Adaptive Initial Users' Overview of a Map Visualisation

by

Zahra Namini Mianji

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2017

Abstract

The ability to customise different aspects of a visualisation system for users is typically lacking in current applications[30]. This results in less immediate and relevant visualisation information to users, as all users are presented with the same overview irrespective of their persona. This thesis proposes a dynamic, adaptive visualisation system that customises the initial view of the visualisation based on the information received from users. This adaptive approach improves the upfront user experience connected with the first stage in Shneiderman's Information Seeking Mantra 'overview first'[57]. This thesis aims to explore whether and how the users' interactions in the first four steps of Shneiderman's tasks [57] can be used to create the appropriate early experience for future users with the same goals or characteristics. The evaluation confirms the possibility of the idea and raises further questions and issues.

Acknowledgments

Firstly I would like to thank Dr. Stuart Marshall, for supervising the completion of my Masters study. He taught me many things in both years of my Masters study especially the second year in which I learnt how a research should be done. I thank him for all of his guidance and support throughout that time.

I thank my friends and the members of the HCI group, for providing feedback and support during my project. I wish them the best with all of their studies.

I would like to thank my parents, my sister and all of my friends for encouraging me during my Masters study and all throughout my time at Victoria University.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions	3
1.3	Methodology	3
1.4	Thesis Structure	5
2	Background Requirements Analysis	7
2.1	Target Users Profiles	8
2.1.1	Users Profiles	9
2.1.2	Developers Profiles	10
2.2	Target User Personas	12
2.2.1	User- First Persona	13
2.2.2	User- Second Persona	15
2.2.3	Developer Persona	16
2.3	Project Requirements	17
2.3.1	Non Functional Requirements	18
2.4	The Project Completion Process	18
2.5	Mash Ups	21
2.6	Related Works	22
2.6.1	Adaptive Visualisation Systems	23
2.6.2	Acquiring Users Information	31
2.6.3	Visualisation Tasks	33
2.7	Summary	34

3	Design	37
3.1	Project Features	37
3.2	User Experience Design	39
3.2.1	End-User Experience Design	45
3.2.2	Developer Experience Design	45
3.3	Discussion	50
3.3.1	Challenges	52
3.4	System Architecture	56
3.4.1	GUI Sub-system	60
3.4.2	Logic Sub-system	63
3.4.3	Database Sub-system	67
4	Implementation	71
4.1	Users' Profile and Logic Component	73
4.1.1	Login Page	73
4.1.2	Sign Up Page	74
4.1.3	Index Page	76
4.2	Developer's Profile	81
4.3	Database	82
4.4	Reporting Component	83
4.4.1	Chart Page	83
4.5	The Previous Versions of the System	93
4.5.1	First Version	93
4.5.2	Second Version	97
4.6	Tools and Languages Choice	100
4.7	Language and Tools Critique	102
4.8	Alternative Implementations	104
5	Evaluation	107
5.1	Performance Metrics	107
5.1.1	Developer's Heuristic Evaluation	110
5.1.2	Cognitive Walk-through	114

5.2	Process and Results	119
5.2.1	Cognitive Walk-through	120
5.2.2	Developer's Heuristic Evaluation	122
5.3	Changes	124
6	Conclusion	129
6.1	Summary	130
6.2	Future Questions	131
6.3	Future of the Project	133
6.4	Contributions	133

List of Figures

1.1	Comparison of visualisation systems	4
2.1	Contai et al. interventions	25
2.2	An adaptive visualisation system model	26
2.3	The first system output	29
2.4	The second system output	29
2.5	The comparison of the current state-of-the-art against the system requirements	31
3.1	A sample database record of my adaptive visualisation system	39
3.2	The user interface story board of my visualisation system . .	41
3.3	The sample user interface of Shneiderman's tasks	44
3.4	The personas' characteristics modification by the developer diagram	47
3.5	The developer's sample interface of personas' characteris- tics modification	48
3.6	The diagram of transitions between different interfaces of the developer's profile	50
3.7	The personas' modification diagram	53
3.8	The global view of the system architecture	56
3.9	Schematic system architecture	58
3.10	The end-users' used case view	61
3.11	The developer's used case view	63

3.12	The logical view of monitoring changes functionality	64
3.13	The logical view of the users' interface changing process . . .	65
3.14	The users' interface changing process view of the system architecture	66
3.15	The relationship between the system components and the system requirements	69
4.1	The system's login page	74
4.2	The system's sign up page	75
4.3	The end-users' map view page	80
4.4	The developer's interface to modify personas' characteristics	81
4.5	The developer's reporting chart which represents the num- ber of updates per personas	85
4.6	The developer's reporting chart which represent number of updates per users assigned to one of the personas	87
4.7	The developer's reporting chart which represents the num- ber of each set of filtering objects used by the end-users . . .	89
4.8	The developer's reporting chart which represents the num- ber of each zooming value used by end-users	90
4.9	The database search component	91
4.10	The relationship between system's components and the sys- tem's requirements	92
4.11	The previous version of the developer's interface represent- ing data-history table	94
4.12	The previous version of the developer's interface represent- ing the visualisation of the number of end-users' interac- tions with their views	94
4.13	The system architecture as a diagram	96
4.14	The Diagram of Saving and Reporting the Changes	97
4.15	The system's previous version's developer interface which includes the number of updates per users' chart and the data history table	99

4.16	The system's previous version's developer interface which includes the number of updates per personas' chart and the tale which represents these numbers	100
5.1	The reference heuristic table [22] (part 1)	108
5.2	The reference heuristic table [22] (part 2)	109
5.3	The developer's heuristic table	113
5.4	The end-users' heuristic table (part 1)	118
5.5	The end-users' heuristic table (part 2)	119

Chapter 1

Introduction

1.1 Motivation

Visualisation systems typically retrieve domain data, convert it into a visual and interactive form, and present a consistent visualisation to all users. However users may be split into different personas, and since they are the main target of a visualisation system their different requirements and preferences should be counted as the first priority in a visualisation system. Customising a visualisation system based on users' preferences plays a significant role in the user's experience of a visualisation system.

A practical example is using users' information, such as their location or their intention of using the visualisation. Users' information can be a reliable resource which users' preferences can be retrieved from. These information can be used in classifying users and producing personas as well. As an example, in a map visualisation a group of people may need a specific map view to access their destination. Therefore, customising map views can result in achieving more users' satisfaction and using the aggregations of all past users' information helps in identifying the appropriate initial view for a new user.

Shneiderman identified an Information Seeking Mantra [57]. This includes seven tasks of users' view of the visualisation which have the high-

est level of importance. The seven tasks are:

- Overview: Gives an overview of the entire collection.
- Zoom: To zoom in on items of interest.
- Filter: Filters out uninteresting items.
- Details on demand: Selects an item or group and shows details when needed.
- Relate: Specifies relationships among data.
- History: Keeps a history of actions to provide undo action.
- Extract: Allows extraction of sub-collections and of the data.

This Master of Science (MSc) thesis aims to explore whether and how the users' interactions in the first four steps can be used to create the appropriate early experience for future users with the same goals or characteristics. The four steps considered are: overview, zoom, filter and details-on-demand.

An already built and tested example is a rule-based system which depends on the user specified queries to choose the suitable type of visualisation. [17]. This gives the opportunity of entrusting the analysis of data types and structures to computers. What is lacking in the system [17] is the role of users' preferences. The type of persona does not affect the system's reaction. A further work that can be done in order to improve the system is to include a new rule based on the type of user that is logged into the system to explore whether the user type has a role in the visualisation type decision making process or not. The ability of customising the system based on users' information could improve the system decisions. As an example in a map visualisation, a local user who is looking for a specific nearby place would prefer a local overview of the map while another user may prefer to access to a more general view to be able to identify several places on the map.

1.2 Contributions

This thesis contributes to the following:

1. An exploration of the issues involved in matching users' information to the initial presentation of visualisations, relating to the first stage in Shneiderman's Information Seeking Mantra [57].
2. Proof-of-concept prototype of the design of an adaptive visualisation system.
3. Evaluation of the users' data factor by using heuristic evaluation and cognitive walk through.

1.3 Methodology

Customising a visualisation system can be done in different sections based on different rules. An adaptive dynamic visualisation system can change the type of visual asset displayed, the surrounding user interface and the initial view (such as filters and zoom-levels) automatically. These customisations can be determined based on the types of users and user queries. Figure 1.1 shows the explanation, as a simple diagram.

Figure 1.1: An adaptable and a non-adaptable visualisation system diagram

This thesis explores how to adjust the initial view of the visualisation with users, based on users' information.

For this purpose I needed some examples. By designing some test personas and studying their goals and scenarios, I was able to extract some requirements for the system by writing scenarios for them and extracting the steps they take during the scenarios and the tasks they needed to use to take those steps. The requirements were then compared against the current state-of-art based on the studied related works in order to identify what are the lacking requirements of the previous systems which are going to be considered in my system requirements.

These requirements were then used to develop a visualisation design. This design was then implemented using ASP.NET, Java Script and the Google Map API. Then the implementation was evaluated to determine its effectiveness using heuristic evaluation and cognitive walk through. Results obtained from the evaluation, were used in improving the system and updating it to the second and last version.

As the terminal step of the system completion, the final version of the system was heuristically evaluated by two of the visualisation field experts.

1.4 Thesis Structure

This thesis is presented in six chapters: an introduction to the project, the background and requirements analysis, the project design, the project implementation, the project evaluation; and a conclusion of the project.

The introduction clarifies what the project is. The introductions also discusses the motivation for the project.

The background and requirements analysis presents the related works and personas related to the target users. The personas have defined goals and scenarios which are used to draw up both the functional and non-functional requirements of the project.

The design takes the requirements from the analysis phase and develops a design for the visualisation system.

Implementation goes through the process of creating the visualisation system based on the design. It will also compare the design tools which are used to build the project.

Evaluation goes through the testing of the visualisation system and the results obtained. This indicates how the project has met the requirements of the project and how well the initial view would perform in front of real users.

The conclusion summarises what has been presented in the former chapters and talks about limitations.

Appendices provide glossaries, the bibliography, and all the parts required for the interviews and evaluation processes. These include the Human Ethics Committee (HEC) forms, raw data used for the scenarios, the protocol used to perform evaluations, and some large tables of the resulting data from user evaluations.

Chapter 2

Background Requirements Analysis

The aim of the thesis is to explore how dynamic adaptive visualisation systems that identify and react to changing users needs, would work. This could provide a richer user specific experience to the stages in Shneidermans Information Seeking Mantra [57]. The goal is to develop a proof-of-concept prototype visualisation, and evaluate its performance. The proof-of-concept prototype is designed to prove the concept that the initial users' overview of the visualisation can be adjusted with the users' requirements.

As previously explained in motivation section, to define users requirements and preferences, users information is needed. By users' information, I mean the information which will affect the users' visualisation view. These information can include their location or other information which specifies how should Shneiderman's tasks affect users' visualisations' view. The needed information may get changed based on the visualisation type and are not strictly needed to define the system requirements, but they clarify users' preferences of the system. Therefore, in the first section, I discuss the different types of users of the project and by analysing their scenarios and goals, I present the requirements of my proposed system, and compare these requirements against the current state-of-the-art. I also

discuss the project's related work, covering similar functionalities with my project.

Examining the related work formed a basis for the project, as while as analyse the project requirements; both functional and non-functional. The functional requirements are informed by the development of personas; both the user and developer personas. By analysing the requirements of the personas, I was able to produce scenarios on how they would use the system. The non-functional requirements were then developed to meet the system needs of these personas.

This chapter also presents the analysis and completion process of the idea, resulting in the specification of design requirements of the project implemented in next chapters.

2.1 Target Users Profiles

Usually users' profiles provide information about the users who are going to use a project; various information such as their age, gender, preferences. This information can be achieved by simply explicit approaches such as interviewing them. Then the information will be used in classifying users into groups. Using these information as the measurement, the people with close characteristics and preferences will form a user group. The groups and their information will then be used to produce personas. A persona is an archetype of sample users upon which the design decisions will be made based on it. What I explained so far was the general approach of forming target users and personas.

In most of the projects, personas are developed by analysing users, but in my project personas are designed at the beginning, in order to classify users. The actual users will be assigned to each of the personas based on their requirements. So the approach I used in my thesis was designing personas and assigning the people who have the same intention of using the map, to those personas. This approach does not necessarily de-

fine the right personas, but the aim of my thesis is to define some persona examples to explore how the users's interaction according to the Shneiderman's four tasks can be used to create the appropriate experinec for future users with same characteristics. Therefore, the persona examples may have some limitations which can be identified and improved while testing the system with actual future users and by the developer's analysis.

There are two classes of users on the client side. One is the general users who will login, access and interact with visualised information, while the second is the developer who will get notified about the high-level patterns in system changes through a reporting component. The developer analyses the changes users assigned to personas, make to their view in order to explore the improvements s/he can make to the designed personas' views in order to make them closer to the views actual users prefer.

The end users who interact with the visualisation's user interface are referred by term "User" and the person who tracks the end users interactions is called the "Developer".

2.1.1 Users Profiles

In my thesis, because my focus is on the initial views which will be presented to users, I had to design several users profiles. As a visualisation example to test the idea on, I chose map to design personas for. Therefore, I needed several types of personas using the map and based on them, the initial views of the map were designed. As an example, a group of people may aim to use a map in order to find the nearest ATM machine or a pharmacy. So a designed persona can be a person who aims to find the local ATMs according to his location. This persona is the representative of the people I explained about, in my example. Based on the persona's target of using the map, their initial view will be designed. To design the system, I had to consider a group of people with general conditions as my users.

Background The users are assumed to be able to work with simple websites and mobile applications. They are not required to have any specific educational level. They are just assumed to be familiar with using electronic map elements, such as buttons.

Motivation As mentioned earlier, users who intend to use the map for a specific target such as finding particular places in a specific area, are the main target users. Adjustment of their initial view of the map based on their aim of using the map, motivates these users to use the adaptive visualisation.

Activities It is intended that users use this project to better evaluate the designed initial view of the map, based on the designed personas. Activities include:

- Profile a map view, so the initial view can be built.
- Make changes to the map in order to improve the initial view based on what they prefer.

2.1.2 Developers Profiles

The second types of actual users are the developers who will not have interaction with the user interface, but will track all the changes that will be made to the users' views as well as users' information and will be recorded in the database server. His aim is to explore the limitation with the designed users and designed initial views in order to improve the final visualisation.

Background The developer would have already moved into industry and worked with transactional databases. He may not necessarily have a university degree, but would have receive a certificate or diploma in

computer programming. As development of web application including databases is one of the first requirements of web developers, a professional developer working in this area could be assumed to have gained a number of years of experience working with either of relational databases.

Software Experience Since the developer is required to be able to connect to the database and read the log files in order to track the changes, he is assumed to be able to write database queries as well as the server language codes which is ASP.NET in this thesis.

Education While I do not assume a professional developer would have a degree, it is likely they would have some kind of qualification, such as a diploma or certificate in computing. I assume instead the years of industry would strengthen his knowledge by providing more working experience.

Motivation As mentioned earlier, the developer seeks to identify the changes which have been made to initial views and extract the must change overview values.

Activities It is intended that the developer uses the proof-of-concept prototype to better evaluate the designed initial view of the map, based on the designed personas. Activities include:

- Access the recorded changes so that the problems of a view can be extracted.
- Compare different versions of the view to understand the persona design problems.

2.2 Target User Personas

A persona is a model of a user that focuses on the individual's goals and specifies what a group of users would expect when using an artifact. The model has a specific purpose as a tool for software and product design. Although a persona models users' profiles, there are some important differences too. It looks like a real user but It is not a description of a real user. The persona represents patterns of users' behaviour, goals and preferences [9].

This section will introduce some sample personas generated based on users' characteristics explained in previous section for the purpose of designing the proof-of-concept prototype. The prototype system uses a map as its example domain. These personas represent the groups of users who the project is targeting for. The user personas were used in designing the system requirements and architecture related with the client side and bonded with the users' overviews and the developer persona was used in designing the server side system architecture and requirements.

Each persona has characteristics, goals and scenarios which define how will this system help users to provide their needs of the system. The personas' characteristics explain about how the initial map's view of the users assigned to that persona is related to him/her. The term 'initial view' can refer to several meanings. Either the first representation of the first time a user signs up the system and gets the view, the views for each time a user logs into the system which are represented to him/her at the beginning before making changes to the view. By the term 'initial view' in my thesis, I mean the view which is represented to the user each time s/he logs into the system.

The personas' goals define the requirements of the users specified to that persona, from the system and the personas' scenarios define the interactions and actions each user assigned to that persona, may do to the system. Scenarios inform how the initial views look like.

There are two approaches of representing the up-front to the users. Each of the approaches has got some costs and limitations. According to the sign-up process of the system, there are two decisions available. Either having the users to answer the questions to specify their map view characteristics, during the sign up process; or signing the user up without asking them to fill the questionnaire and representing the same view to all the users letting them to customise the view later by themselves. The first approach is time consuming since users have to go through the questionnaire while the second approach is lacking the privacy and customised aspect of the users' views. I chose the first approach for my thesis since one of the contributions of the thesis is to define persona examples and built the up-front according to the end-users examples. Without considering this contribution, the system is not considered as an adaptive visualisation system which customises the 'initial views' of the users, since in that case, the initial views would be the same and defining personas is meaningless. Moreover, time cost of the first approach is part of the second approach as well. Because asking questions during the sign-up process, helps in representing a more preferred map view to the users and with skipping this step, users will have to consume more time to change the map view to what they need. Therefore, comparing the time cost and the privacy cost, I chose the time cost which can help in building a more useful up-front for the system's users and designed the personas and their scenarios based on this choice.

2.2.1 User- First Persona

Job: Delivery man

Location: Wellington, New Zealand

The first persona is a delivery man who uses the map in order to find all restaurants in Wellington. Due to the fact that he has to deliver objects to restaurants, he prefers a view of the Wellington map with the only

information of all the restaurants.

Persona Goals

Goal 1 This person wants to have a general view of all restaurants of Wellington, marked.

Goal 2 He wants to have access to their addresses.

Goal 3 He wants to be able to zoom on a specific restaurant in order to get a local view of the area.

Goal 4 He wants to be able to change the map information and re customise it.

Context Scenarios

Scenario 1 Having completed a simple prototype of his initial view, he can see the initial view of the Wellington map. The initial view is automatically set up and the restaurants are marked and ready to be used.

Scenario 2 The delivery man wants to get the address of a specific restaurant, he will click on the restaurant mark and the address will be shown as a tool tip.

Scenario 3 The delivery man wants to focus on a specific area and see the road names. He can zoom in, drag and get a local view of the area. He can zoom out to get the general view again.

Scenario 4 The delivery man wants to change the information of the map. He needs to have ATM machines clarified as well. He can check the check boxes of places which needs to be added to the map and click on the add button, and the map information will change.

2.2.2 User- Second Persona

Job: Student

Location: Wellington, New Zealand

The second persona is a student who uses the map to find the hospitals nearby. Because he wants to identify the close hospital, he prefers a Wellington map with the local view of his location with all the nearby hospitals marked.

Persona Goals

Goal 1 This person wants to have a local view of where he stands with nearby hospitals marked.

Goal 2 He wants to have access to their addresses.

Goal 3 He needs to find further hospitals so he wants to be able to zoom out to get the more general view of the area.

Goal 4 He wants to be able to change the map information and re customise it; he needs the map to mark hospitals too.

Context Scenarios

Scenario 1 Having completed a simple prototype of his initial view, he can see the initial view of the Wellington map. The initial view is automatically sets up and the hospitals are marked and ready to be used.

Scenario 2 The student wants to get the address of a specific hospital, he will click on the hospital mark and the address will be shown as a tool tip.

Scenario 3 The student needs to get a more general view of the area or he needs to go to the next hospital which is not in his view. He can zoom out or drag the map in order to change his view and find further hospitals.

Scenario 4 The student wants to change the information of the map. He needs to have ATM machines and bank marks as well. He can check the check boxes of places he needs to add to the map and click on the add button, and the map information will update.

2.2.3 Developer Persona

The developer persona is a master of computer science student who has access to the server side of the project. He aims to create triggers to copy an old value to a history table if it is changed when someone edits a row. He also stores the user ID and last modification date of the original table every time someone edits that row. Using the reporting component and the visualisations made based on the history table he will be able to study changes and extract the limitations of the designed persona and the initial view.

Persona Goals

Goal 1 This person needs to have access to the system database.

Goal 2 He needs to have the permission of coding and editing the database and the server side codes.

Context Scenarios

Scenario 1 Having access to the server side of the project and the database, the developer will create a trigger to copy the old value of a field to the history table [59] whenever the field is about to get changed. He can read the history table whenever he wants without interfering the system database.

Scenario 2 The developer should be able to make changes to the default values of the tables and the initial views as the result.

2.3 Project Requirements

By designing some test personas and studying their goals and scenarios, I was able to extract some simple functional and non-functional requirements for the system.

Functional Requirements

Functional requirements describe what the system should do [21]. Each square bracket at the end of each requirement indicates which persona goal or scenario, it has been drawn from.

Requirement 1 The process of matching users' preferences and configuring the visualisation systems initial view should be automatic, based on initial seeded personas created by the developer [User's First and Second Personas; Scenario 1].

Requirement 2 The system should support and record subsequent manual configuration by users [First and Second personas; Scenarios 2, 3 and 4].

Requirement 3 The system should react to identified patterns of initial views being manually modified [User's First and Second Personas; Goals 2, 3 and 4].

Requirement 4 The system should support using location information to inform the selection and scope of an initial view [User's First Persona; Goals 1].

Requirement 5 The visualisation developer should be able to get reports of the manual changes made by the users to the initial seeded personas and views originally created [Developer Persona].

2.3.1 Non Functional Requirements

Non functional requirements describe how the system should perform rather than how the program will respond to the user's functional requirements.

Requirement 6 The user interface should be compatible with different screen sizes.

2.4 The Project Completion Process

I built a prototype of an adaptive visualisation system which customises the initial users' overview of the visualisation based on the information received from users. To build the prototype, I chose a visualisation example (map), and designed persona examples at the first step and based on their requirements and scenarios, the initial users' overviews of the visualisation system were designed too. A questionnaire was designed based on the designed personas. The questionnaire is part of the sign-up process and based on the end-user's answers, s/he will be assigned to a specific persona and the the persona's view will be presented to him/her. Moreover, a developer component is implemented as a prototype too. The developer component is for monitoring the changes end-users make to their views and modifying the designed personas and their views based on the end-users' interactions with their user interface.

The system's design and architecture will be discussed by details in chapter 3, but generally describing the prototype, it consists of main sections which are the front-end and the back-end. Each of the sections in-

clude several components. The front-end has two types of users. The end-users who interact with the visualisation (they are referred by the term 'user') and the developer who tracks those interactions.

This project is proceeded through different ideas over the time of studying more about the related works and the project conditions and discussing more with my supervisor. Studying related works of the same research area helped in getting different ideas about developing an adaptive visualisation system.

My first idea was about customising the type of charts which will be shown to users. In this approach, the first step was to interview with people and understand their requirements and preferences from the system. During interviews, I aimed to identify suitable charts for the target tasks. In this approach suitability of a chart depends on several factors, such as types of data related to the task, persona queries, format and frequency of the data which should be represented and the type of charts which suits best to the personas' preferences. As an example, the people who decide for the education policies may be more interested in bar charts which make them able to compare performances annually while a student may prefer a dot map to explore weak points of his own study year. The intended dataset for the first phase was an educational dataset and the system would be a rule-based system. By rule-based system I mean, some rules about the types of data and users' queries and types of personas would be set. So, when the users would sign into the system, based on the query and the type of the data of the query results and their type of persona, the system would propose the chart which suits best according to the rules. To sum up, my first understanding of making a customised visualisation system was about presenting different types of charts to the users, based on the user queries, data types of the query results and users' preferences. But this idea requires more time than my thesis valid time because it requires a more sophisticated system architecture. Then I decided to work on just one type of chart (bar chart) and work on the interface

adaptation by changing the visualisation interface based on users' preferences.

In the second approach, I had to provide questionnaires for the users who sign up in the system, in order to identify users' preferences of visualisation interface, such as if the users are colour blind or what their preferred colour of bars, text fonts and elements are. I aimed to identify if users are familiar with visualisation or the data set or if they require notifications or axis names; and if they prefer the chart to include grids or the bars to be separated and if the names and values of bars be written on the chart. But it could be estimated that the answers to all those questions would be yes, so it would not be a useful approach, moreover it would not be a high level of customization since it was related to only the user interface and not the system architecture.

After achieving certainty about my approach and my project idea, I started the project as a simple prototype including the Shneiderman's tasks [57] integrated with just the end-user's overview and tracking the end-users' interactions with the system. As moving further, I decided to extend the developer's section of the system and add more functionalities and components to it. And, meanwhile change the system's functionalities to more self-adaptable approaches. Improving the system provided the better observation of the end-users' customisations of their views and easier modifications of the personas and the designed views.

The approach of providing data visualisation of displaying changes in system's data over time, may seem similar to Google Analytics which is web analytics service and provides statistics and basic analytical tools for search engine optimization (SEO) and marketing purposes [2]. Google Analytics includes many features such as custom reports, email-based sharing analytics, but the feature which is important to me is that it is a data visualization tool which displays data changes over time using charts [2].

The difference between Google analytics [2] and my system is that it visualises the changes per individual users, but my system visualises

changes per personas. My system includes tracking changes of individual users, as well, but this is a supplementary function of the system which is provided to help the developer to better detect the initial views' needed changes. The core function of the system is based on the personas, since the designed views are based on the personas' characteristics. Additionally, personas' analysis results in improving the initial views. Therefore, even users' interactions' tracking results in personas' improvement. This is the core difference between these two systems. Comparing them against each another cannot clarify which approach is the better approach since each of them are useful for different goals. As an example, if a system is designed based on personas, personas' analysis will help more, on the other hand, in a system designed based on the users, personas' analysis cannot be so helpful.

I did not build my system's prototype on top of Google analytics [2] because of the discussed difference. Moreover, the point of developing my system's prototype was to explore the issues, not to prove any specific user interface is better.

As mentioned before, the visualisation example to implement my system is a map. To choose the map's library which I wanted to use, I searched in order to find the best and easiest option. This research will be discussed in chapter 4, but as the map library, I chose Google Maps API [55] and now I am going to represent some of different mash-ups related to Google Map API before discussing the related works, since mash ups are considered as part of the related works.

2.5 Mash Ups

In this section, I am going to explain a few mash ups of Google Maps API. A mash up is a technique by which a website or web application integrates complementary data, presentation or functionality from two or more sources to create a new service [55]. Apart from the map feature of

the Google Maps API, there are tons of other uses from Google maps. I have listed five of them which are kind of similar to my project because they also restrict the map information to specific items.

The convenient Reel Box mash up shows what movies are currently playing at some specific local area cinemas. The Reel Box finds user's location automatically, and when the user hovers over a desired show time, it provides directions from his current location to the cinema offering the movie [32].

The Google Maps Hotel Finder Experiment mash up uses public transport travel times to generate maps that help to find a hotel within a certain travelling distance. Using a slider, the travel time can be controlled [32].

Housing Maps is a mash up which provides a list of housing for rent and for sale in most major cities. The site also includes filters which can be drilled down to listings in a specific price range [50].

Sam Knows mash up gives the ability to look for a house or a building on a map of the UK and see what types of broadband Internet are available at that address [56].

Trulia Local mash up is a handy Google Maps based tool which allows to search for nearby information such as nearby crime rates, schools (and their ratings), restaurants, banks and gas stations. Data layers are available for most major cities [32].

2.6 Related Works

From my point of view, my thesis consists of several sections: exploring users' information and preferences, designing the visualisation system

and customising users' initial overviews of the visualisation using different tasks. So to investigate the background works of my thesis area, I tried to read papers about all of the mentioned sections. This literature review includes papers on designing adaptive visualisation systems, exploring users information techniques and reviewing and comparing users overviews of the visualisations.

2.5.1 section is about comparison of some similar works with my work and 2.5.2 and 2.5.3 sections are mainly about the points I learnt from other works in order to build my work.

2.6.1 Adaptive Visualisation Systems

The emerging area of adaptive visualisation has resulted in several systems being developed or proposed that enable a more efficient and effective and usable interaction with complex visualisation representations. However, there are always some limitations with the projects. As the first limitation which differentiates completely my system with other adaptive visualisation systems is the consideration of users as one of the customising factors. The solving approach of this limitation is considered as my system's first functional requirement.

An adaptive visualisation system was introduced by Desai et al. Their system [18] is based on an educational dataset with a variety of data types. The system is based on a set of rules that depends on the user specified queries to choose the suitable type of visualisation. The type of persona does not affect the system's reaction. The user queries are distinct from personas and they are not substitutes for each another, because user queries mean any changes each individual user makes to his/her visualisation's data. On the other hand, personas are patterns of a group of users which have common goals and preferences and the visualisation is designed customised to those goals and characteristics. A further work that can be done in order to improve the system is to include a new rule

based on the type of user that is logged into the system to explore whether the user type has a role in the visualisation type decision making process or not. Desai et al. [18] created a query builder which is a web based interface that gives the ability of generating queries to users and the online interface system makes the dynamic visualisation for those queries. They implemented the system on a dataset from Washington States Public Education System. The aim was to test the effective impression of this specific dynamic data visualisation tool.

Similarly, BOZ system [12] which is a dynamic graphic design tool, does not customise the system based on the users and the system works the same for every user. Customisation in this system happens based on the requirements of tasks [12]. The system [12] designs graphics based on the purpose of the task which the graphic will be used for. BOZ system [12] allows users to use graphical interfaces instead of logical interfaces. It also gives the ability to the users to get their search results in a system, in form of charts and graphics instead of raw texts. To test BOZ system [12], it is designed on an airplane dataset. It is used to present the airplane schedule information by graphic designs. The evaluation on the system shows BOZ system [12] was successful to reduce users' performance task times. The evaluation of the system [12] shows that using graphics to represent information is a more successful approach than other kinds of presentations such as text or tables.

Another common limitation is not tracking users' behaviour to study the successfulness of the visualisation system. My proposed system seeks to better support this functionality [functional requirement 2 and 3].

Contai et al.'s visualisation system focuses on identifying important parameters to each of the users and adapting those parameters in the provided visualization [11], however they do not save the changes and study them later. I built on their work and extended it further by studying the initial views used by the designed personas, and how they are subsequently adapted. As a consequence, my system saves the changes that

users make to their views, and forms a report to be sent back to the visualization developer for subsequent adaption.

Conati et al. [11] focus on the parameters that make the system adaptable to users and suggest two important parameters: what to adapt (understanding which features should be adaptable); and how to adapt (how should make those features adaptable). Most other systems answer the how to adapt question by recommending an alternative visualisation based on each of the parameters, but their system identifies important parameters to each of the users and adapts those parameters in the provided visualisation. Figure 2.1 shows a visualisation with different interventions. Although they can be used alternatively for each another, based on the user's parameters, one of them may seem more suitable.

Figure 2.1: Interventions used in a study [11]

Nazemi et al. get content and users' information using two modules: conten and interaction analysis[45]. Their content analysis module extracts data properties, and their interaction module extracts users' information from different sensors. Based on this information, the output component produces the adaptable visualisation. The advantages with the

system is the ability to include several parameters in system decisions and to produce different kinds of adaptable visualisations. In general, the number of adaptive visualisation systems is not high, and many of them either include adapting single parameters of the visualisation or include several parameters for a single visualisation type. In comparison, Nazemi et al.'s system includes several visualisation types adaptable to several parameters. Similarly, one key difference between my system and their system is the fact that there is not a data store behind the system in order to track the system changes later. As a result, there will not be differences in the system's interaction with users as time passes and the system relies on the data achieved using sensors. Figure 2.2 is the system architecture of the system [45] described.

Figure 2.2: The model of the adaptive visualisation system [45]

Saving changes that the users make to visualisations can be useful in tracking users' preferences and adapting the system by them as time passes. Golemati et al. [25] system has a similar approach however it is not designed as an automatic visualisation system. It is not able to add users automatically and requires to interview them beforehand. This sys-

tem suggests the visualisation based on user's information, data which is going to be visualised and system configurations. Users' information are extracted from users' profiles. This system is a rule-based system. Rules clarify the suitability of the visualisation based on the defined parameters. The system includes a database which contains the rules and the visualisation method selection matches features from users, system and data with the visualisation properties. The rule database is made by visualisation experts with users. It also adjusts user's profiles according to users' behaviour as time passes. The disadvantage with the system is that it is designed based on a set of rules which are specific to users and adding more users to the system may produce problems. Adding users may not be able to be done automatically and may require more time and effort to first analyse the users who want to be added. However, solving this limitation has been considered as the system requirement [functional requirement 2].

One of the most important steps in a research is the system evaluation which can greatly examine the system successfulness and declare the improvement approach. Robert et al. [24] system is a system without the previous discussed limitations but it does not include an evaluation or reporting section to understand what were the results of the work. However, evaluation played a great role in my system development process. There are two types of evaluation in my system. The first one is the usual evaluation which measures the successfulness of system performance at the final stage of system completion and can be done using different approaches such as heuristic evaluation, cognitive walk through or user tests. The other type of evaluation which I meant is my system reporting component which gives the ability of evaluating of the personas and their view popularity to the developer [functional requirement 5]. There will be more detailed information about this component in next chapters.

Robert et al. [24] describes the architecture of a rule-based system which dynamically selects modalities for representing information. The

system then specifies techniques for representing information within each of the selected modalities. The system is user adaptable too. It uses models to store users' information and the stored data changes during time as users' needs change too. The system has got two levels of modelling personas. The first phase is the modality selection phase which is about analysing the information which is going to be presented and making presentation model decisions such as being visual or auditory. This phase consists of three steps. Expressiveness step decides which model is suitable to present the information. Effectiveness step determines which information should be represented and adaptation step is the adapting phase of visualisation to users' abilities and needs. There is one more phase in the work of this paper which is technique selection phase. This phase is about techniques of using selected modalities in previous phase. As an example if there should be redundancy with the modality; should there be both textual and graphical presentation to better represent the information or not. These techniques depend on users' preferences. Users' preferences determines the techniques of how to use modalities and if they should be combined [24]. Figures 2.3 and 2.4 are examples of the system outputs based on type of personas using the system.

QUESTION: how many boards in each lot failed each test?

RESPONSE:

Figure 2.3: The response for the assurance manager [24]

QUESTION: how many boards in each lot failed each test?

RESPONSE:

each lot	each test	number of boards
lot_1	final_test	1
lot_1	functional_test	3
lot_1	visual_test	2
lot_2	final_test	2
lot_2	functional_test	1
lot_3	functional_test	2
lot_3	visual_test	2
lot_4	final_test	1
lot_4	functional_test	2
lot_4	visual_test	2

Figure 2.4: The response for the factory manager [24]

In this system [24] users' information is stored in users' models. There are personas in format of user models which determine each group of users' preferences, but the visualisations are adaptable to the individual

users as well. As an example, the system represents multiple presentation for new users but a single most used presentation for older users. This system [24] has been built based on a machine factory dataset.

According to the discussed related works, adaptive systems are mostly rule-based systems, since they have to define a set of rules to make the system adaptable and act according to the set rules. A rule-based system is a system with a set of "if-then" statements that uses a set of rules about how to act upon different inputs of the system [63]. This aspect gives the opportunity to define a set of visualisation's actions to be taken based on the input they get from the users. As the result, my system can be considered as a rule-based system too, because the system represents the views based on the answers which the users provide during the sign-up process. However, it can be considered as a partially rule-based system since not every part of the system is acting automatically and based on a set of rules and one important part of the system is controlled by the developer manually.

By identifying the requirements of my proposed system and the related works, I was able to compare the current state-of-the-art from 2.5.1 section against my system requirements in order to find out what are the lacking requirements of those systems which are going to be considered in my system requirements. Figure 2.5 is indicating such an attempt.

	Desai et al.	Boz system	Contai et al.	Nazemi et al.	Golemati et al.	Robert et al.
Requirement 1	✓	✓			✓	
Requirement 2			✓	✓		
Requirement 3			✓	✓		
Requirement 4						
Requirement 5						✓

Figure 2.5: The comparison of the current state-of-the-art against the system requirements

2.6.2 Acquiring Users Information

Dynamic visualisation systems that are customised based on users should be designed to be adaptive based on users' preferences and actions. Making system automatically adaptive to users, requires information about the individual users. There are several approaches to get the users' information [23]. In explicit modelling, the information is achieved by directly asking questions from users and in implicit modelling, information will be extracted from users' interaction with the system or involving them in special tasks [23].

I chose the explicit modelling for the sign-up process due to the fact that my main focus was on how the information is going to be used, not the approach of getting it. I found the act of direct asking acceptable by referring to other papers too. An example is a user adaptive map based system [61] which presents automatic location facilities using map visualisation. In this system [61] the initial presentation of the system is based

on direct questions and users' behaviours are recorded in order to extract users' preferences in time pass as well. The preferences are saved in user models. The user models change by changing users' interactions and preferences over time.

However, I used implicit modelling, too. Because, while users interact with the visualisation, these interactions affect on the developer's reporting data and the reporting data will be used in customising the users' assigned personas and views. Therefore, users' information are being collected by implicit approach which is using users' interaction with their views.

Also, the more preferred approach was the implicit modelling since there were more papers following this approach. As an example, Nazemi et al. [44] introduces a system which gets users' information automatically by analysing users' performance in interacting events. The system detects users' preferences by involving them by interactive events and analysing the results. The developed system can be used for different applications' user interfaces without needing so much configuration.

BDVR system also [28] monitors users' behaviour, detects their pattern behaviour and based on that recommends a visualisation which suits better to the specific user. This system [28] does not require user specified information because users' behaviour analysis is a dynamic task. Although the first phase of this system [28] is different from mine since my system requires users to provide their preferences and information, both systems use the users' interaction with the system in order to improve the designed personas. The paper [28] recognises patterns from users' behaviours by referring to iterative users' behaviours as patterns.

There is a short paper [20] which introduces a visualisation system that is adaptive to users' designs, available resources (hardware, software), tasks, data form and content which wants to be visualised. It [20] describes the system characteristics using four models: user model, task model, resource and data model. This [20] paper has also used interactive games

in order to extract users' characteristics information. This system [20] gets users' information automatically from users' interactions indirectly.

2.6.3 Visualisation Tasks

A useful starting point for designing advanced graphical user interfaces is studying different users' tasks of a visualisation: overview, history and extract or details on demand. Using those overviews helps in customising a visualisation system based on the users. As an example in a map overview, the level of zooming, the maps' information can be controlled based on users' requirements using these overviews. To better understand the idea, here is an example from the designed personas. A delivery man who has to deliver objects to different stores of a city would probably prefer a map (since the training type of visualisation of the thesis is a map) with a general view of the city map with the filtered information of stores; on the other hand, a student who uses the map to find the nearby pharmacies, will prefers a map with the local view of his location with all the nearby pharmacies marked.

The paper which I referred to in studying user's overviews is written by Shneiderman [57] which brings up a task by data type taxonomy with seven data types which are one-, two-, three-dimensional data, temporal and multi-dimensional data, and tree and network data; and seven tasks which are overview, Zoom, filter, details on demand, relate, history, and extracts. The used overviews will be explained more in further chapters.

There are also more sophisticated types of overviews which has been proposed in recent years. Such as this paper [13] which introduces and compares two of the most common user overviews which are overview+detail and focus+context. Overview+detail makes the users able to see the context of a detailed area. Google map is a good example of this view.

Overview+detail and zooming are the most common and preferred components of user interfaces. The other view which is focus+context

merges focusing and context into a single view beside all other parts which are visible as well. The view is visible within the main interface surrounds with other contexts. The below pictures make the two explained overviews clearer. I have chosen some of these overviews according to my type of visualisation and the designed personas, which will be discussed in advance.

2.7 Summary

There are different examples of customising a visualisation system. Information visualisation which is about adapting what to present, data customisation can be an example. Presentation adaptation which is about how to present information, an instance is adapting charts based on data or users' preferences and Interface adaptation which is about how to interact with this information [11]. The literature review was generally on adaptive visualisation systems and included papers about all these different areas. Although my topic is an instance of presentation adaptation because it adjusts the initial view of the visualisation based on users' preferences, the other papers helped me in getting a more professional view in general ideas of designing, building and evaluating an adaptive visualisation system.

Most of the adaptive visualisation systems change the predefined and customised visualisation based on the interaction of the users with the system during time pass. I learnt it as an approach which is necessary to be used if the system improvement is the goal.

By analysing the related works of the thesis topic, the general limitations of the topic became clear. Using those limitations, some system requirements were extracted. Briefly, the limitations can be categorised as:

1. The systems do not include user customisation (either automated or manual) in the visualisations they represent.
2. The systems do not include tracking and studying users' behaviour

and interaction with the system, because they do not include data store in their back end in order to save and track the system changes.

Of course, each system includes one or more of these limitations and the mentioned limitations were the conclusion of studying several systems. What I am interested in, is designing a system architecture for an adaptive initial views of visualisation system. Considering the mentioned limitations, in next chapter I am going to present a system design and architecture which focuses on providing solutions to the discussed requirements and limitations.

Chapter 3

Design

In chapter 2, by analysing the related works of the thesis topic, the general limitations of the topic became clear. Using those limitations, some system requirements were extracted.

Considering the discussed limitations, in this chapter I am going to present a system design and architecture which focuses on providing solutions to the discussed requirements and limitations.

This chapter discusses the project features in section 3.1, the user experience design which represents a general design of the user interface in section 3.2 and a more specific system architecture in section 3.3. It also presents a table which presents the connection between different sections of the system architecture and the system requirements. Since the covered related works and personas are used to discover the project design and how it should meet the users' requirements.

3.1 Project Features

Considering the listed limitations at the beginning of the section, a few system features will be discussed in this section which later in system architecture section will be extended. Below are my solutions to the listed limitations:

User Customisation The base of this thesis is proposing a dynamic, adaptive visualisation system which customises the initial view of the visualisation based on the information received from users. To design such a system several components are involved. End-user and developer interface component, the back end, the database and the reporting component are different sections of my system which will be discussed more during this chapter.

Tracking of Changes In order to be able to save and track the changes which users make to the visualisation, a reporting component and a database is needed. The reporting component takes care of tracking the changes and the database is needed for saving them.

The database includes users' data and their visualisation's view's data. Users' data such as their username and password. The visualisation's view's data are the data which are related to the Shneiderman's seven tasks [57] such as the zooming level value, the filtering information and the other tasks. Supplementary information can be saved as well, such as the time of user's actions and the type of their interaction with the system. Figure 3.1 represents a sample database schema. It includes the username and the password as characters format for the users' information; the level of zooming as an Integer value and the set of places which should be included in the visualisation (in this example a map), as the Shneiderman's tasks' [57] data. The supplementary information are the time of user's interaction with the visualisation in format of standard time, the type of interaction and the number of his/her interactions with the visualisation.

The date and time column of the database is used for extracting time stamps to be used by the developer. As an example, a specific date such as 20th of January if the user's log in time is 1:30:15 and the first update is 1:30:20, this represents the update which has been initially done to the user's view. Moreover, the time since 'start' of each user's view can be calculated. This allows the developer to calculate time since start of the

view, to identify customisations since initial views.

	action	revision	dt_datetime	idusers	pass	zoom	filter	persona
	insert	1	2016-11-02 14:23:12	user1	123	15	restaurant	local/health
	update	2	2016-11-02 15:09:09	user1	123	14	bus_station	local/health

Figure 3.1: A sample database record of my adaptive visualisation system

As the reporting component, different approaches can be used. What I aimed to achieve using the reporting component was transferring the changes from the front end to the back end and the database. This gives the opportunity of saving them and retrieving them later as tables and visualisations which make possible the ease of use and comparison of users' data.

Next section discusses the user experience design which includes explaining the standard visualisation user and the data they need to provide for the system and how do the data relate to Shneiderman's information Seeking Mantra [57]. And the architecture of the adaptive visualisation system will be proposed which covers the discussed limitations and requirements.

3.2 User Experience Design

The user experience design in connection with information seeking Mantra [57] and without concerning about the type of visualisation, includes a questionnaire to explore how would users interact with Shneiderman's tasks [57]. A diagram is also included to show the relationship between the user and the tasks and a basic user interface design which shows how would it looked like if it would be implemented.

To specify how to implement information visualization systems where the Shneiderman's tasks [57] contribute to the design method, this ques-

tionnaire will be used. In a typical example describing a document analysis visualization tool, the author tries to get information in details how each aspect of the Mantra [57] forms his/her design [15]. In the questionnaire, there should be questions about how each of the tasks are going to affect the user's view of the visualisation.

The discussed user experience in this section belongs to both class of users (end-users and the developer), since both groups of users' profiles include Shneiderman's tasks [57].

Figure 3.2 represents the user interface story board of a visualisation system including Shneiderman's tasks [57] to customise the initial view of the visualisation. The tasks are assumed as the user interface elements such as buttons or textboxes. All these elements are part of the user's profile page. The sing-in/up form is representing an extra page.

Figure 3.2: The user interface story board of my visualisation system

Story board enables the developer to model the high-level relationships between major user interface elements and therefore, ask usability questions. User interface-flow diagram is used for modelling the interactions that users have with the system [5].

The questionnaire is usually provided in the sign-up form. However, there are other story boards and approaches which can be used and each of the approaches have their own advantages and challenges which will be discussed more in details, in the discussion section. In case of providing the questionnaire in the sign-up form, as can be seen in figure 3.2, each of the tasks are providing some specific facilities to the user's view which by changing the task's values, the view will change too.

When logging in to the user account, the user should be able to access his/her view or log out the system. These actions can be taken care by

providing relative buttons.

In case of accessing their view of the visualisation, they are represented by the overview of the visualisation. The overview provides a general context for understanding the dataset [15]. The overview is the section which includes all the other tasks and by changing the tasks' values the customised views will be provided to users. All the questions of the questionnaire are to explore how should this overview be presented. Many things can be different in overviews. The represented information, the zooming value, the user interface and the interactions, the comparison of the two or several types of data and etc. As the result, all the other represented rectangles in figure 3.1 which are connected to the overview are the tasks which are included by the overview and provide the changes to it.

The history task of the Shneiderman's paper[57], is about users to be able to easily return to the previous state in the process of exploring data [15]. To perform such a task, the undo or redo buttons can be provided which takes the view one step back each time it is clicked. The history task can also be used in keeping the history of the changes the users make to their views. Using this ability, the represented view can include those changes next time the users return. Keeping the record of the users' interactions with the system and the history of their views is the approach which is used in adapting the users' views to the ones which are closer to their preferences. Storing data (the updates to the visualisation's elements such as zooming value) and logs and considering them in a specific time periods and aggregating information related to the users of a specific persona makes the scaling possible. The developer will be able to have a high-level view of the persona's view elements and can understand how to change the view to make it closer to a more appropriate one.

The details-on-demand task of the Shneiderman's [57] paper provides supplementary information of the visualisation's data [15]. Since there are limitations of the screen, not every data is available on the first view of the visualisation. To represent the supplementary data, either buttons or the

other kinds of interactions such as hovering and tooltips can be used. The types of interactions can be chosen based on the user's preferences and is customisable.

The extract task is about being able to extract important information [15]. This can be done by saving the view and its related information. An example is considering a crowded dot chart in which by zooming task, the user can find the information s/he was looking for and by saving the view, s/he will be able to access it later, too. This can be possible by providing either a save button or asking the user if s/he wants to save his/her view before logging out. Saving the view, same as the history task can have different approaches. Saving can be used as representing the view with all the changes as the initial view for the next times or keeping the view and representing it when the user wants to use it and representing the designed view as the initial view.

The relate task of the Shneiderman's paper [57] allows the user to view relationships between several data. This task can be performed in different approaches, but an example can be adding several types of data to be represented in a visualisation, next to each another or on top of each another so that the user can compare them. A really simple example is a bar chart which may include single bars of math scores of three classes and the user adds the second data bars of physics score to the visualisation. S/He will be able to have a comparison of the two courses.

Filtering is about controlling which data points should be visible in a visualisation [15]. Filtering can be performed to a visualisation view using different approaches such as providing checkboxes to hide or show data points or information on a visualisation. One example of filtering is customising the initial view of a map based on the user's location. Loading a map based on the user's location is a filtering task since it is filtering other places of the map, out. This task has a great role in customising user's views of their visualisations.

The last task is zooming which is about adjustment of the size and

position of data elements on the screen [15]. This option can be performed using buttons or a slider.

The questionnaire for getting users information and using that information in adjusting the initial view of the visualisation should consist of questions about the discussed tasks. The questions should clarify how the user wants each of these tasks to be used and what values they should include. Questions should be relied based on the data store. Figure 3.3 shows a simple sample user interface including the discussed tasks.

Figure 3.3: The sample user interface of Shneiderman's tasks

There are two classes of users in the system. One is the general users who will login, access and interact with visualised information, while the second is the developer who will get notified about the high-level patterns in system changes through a reporting component. The end users who

interact with the visualisation user interface are referred by term "User" and the person who tracks the end users' interactions is called the "Developer".

3.2.1 End-User Experience Design

The discussed user experience belongs to both class of users since both group of the users' profile include the Shneiderman's tasks [57]. The end-users' profiles include the Shneiderman's task [57] as elements to customise the visualisation. Elements such as zooming buttons or hovering interactions to represent details. The end-user experience design was already discussed in 3.2 section.

The developer's profile is about personas and include the Shneiderman's tasks [57] as elements in order to be able to better analyse the visualisations which represent the personas' and end-users' updates of their view and interactions with their customised visualisation. Elements such as a time slider which filters the time periods of end-users' actions as can be seen in figure 3.3 and will be discussed more in 3.3 section; or the comparison parts which can be seen in figure 3.3 and compares two selected data related to the end-users' views' updates in the visualisation.

3.2.2 Developer Experience Design

But, the developer has one more interface which is the persona modification interface and makes the developer able to modify the personas' characteristics via the interface without needing to access the database directly. Figure 3.4 shows such a diagram. As can be seen in the diagram, when the developer logs into his/her account has access to the interface and can select the persona which wants to change its characteristics and the task of the persona which needs to be changed (such as the zooming value or the filtering objects), and enter the data which should be replaced with the previous value of the task. Figure 3.5 shows a simple sample of such an

attempt. The persona selection drop-down list includes all of the system's personas. In the designed sample I showed the personas by the numbers, but in the actual system, the personas can be listed by their names. In my system they would be 'general/delivery' persona and 'local/health' persona. The names of my personas are based on the characteristics of my designed personas which were a delivery man who needs a general view of the map and the local user who is seeking nearby health centers. The task drop-down list includes the visualisation's tasks used in the system and their values can get changes according to users' preferences. In my system, they are zooming, filtering and location. The value textbox is the place in which the chosen task's value should be entered. In my system the value for zooming would be whole numbers of 0-19 range or the value for the filtering task is the any subset of set of places which are shown as the checkboxes in the end-users' interface, such as (ATM, bank, ...).

Figure 3.4: The personas' characteristics modification by the developer diagram

Select a persona :

Select a task :

The value :

Figure 3.5: The developer's sample interface of personas' characteristics modification

There is an disadvantage with this approach which can be handled via coding. The disadvantage is about if the developer enters a wrong value for the selected persona's characteristic and this error is not predicted and handled in the back-end code, causes database errors and system break down. As an example if the developer aims to change the zooming value of persona one. The zooming value should be a positive integer number and the developer enters a minus number or even a string. In that case, the inserted value does not match the data type of the database and this creates error. The challenge can be handled in the back-end by checking the inserted value in the textbox before sending it to the database. And, if the formats do not match, the developer will get a warning. By this approach, the database error and the system break down will be prevented. Moreover, the person who works with the interface is the developer who knows about the data-base's data formats.

Figure 3.6 represents the transitions from different screens related to the developer profile. The solid lines represent transitions between different interfaces. The figure represents that when the developer log into his/her profile, has access to several visualisation interfaces which rep-

resent different end-user's data as charts, the database search interface which provides the possibility of searching specific data in the database without needing direct access to the database and the personas modification interface which was discussed as figure 3.5. The visualisation overview interfaces include the charts which represent the users' interactions with their views. It includes different Shneiderman's task [57] to better interact with the visualisations. More details will be discussed in implementation chapter.

Figure 3.6: The diagram of transitions between different interfaces of the developer's profile

3.3 Discussion

Time Slider can be used in the system to classify the data which is about the changes users make to their views, based on the time periods. As an example, analysing how many users have changed their view's zooming at the beginning when they entered to their account or at the end before signing out. This helps in better analysing the changes and understand how to adapt the visualisation view for the user better. As an example, if a user logs in to his/her view and the first thing he does is

zooming in the visualisation and then playing around with the visualisation and this pattern is repeating among other users specified to a specific persona, this shows that changing the view to a more zoomed in and local view is a change which is really needed for the users. Or if the users zoom in and then again return to the previous zooming level, this can show that at the first stage although the more local view may seem better but returning to the previous level of zooming shows that the users prefer the more general view, more.

Self-adapting is about automatic systems which is about the ability of the system to self-adapt itself to the changes. My implemented system is not a self-adapting system and the changes will be studied and performed by the developer. The reason is, the system is just a prototype to test the requirements and the possibility of using the Shneiderman's tasks [57] for an adaptive visualisation system; not a full system. Moreover, because the implemented system is a prototype, the number of users is not so many people, however in a real system which is not just a prototype, the number of users is a huge number. As the result, making the system as an automatic adaptive system did not have any specific effects on the results and also was a time consuming process and was not included in my top priorities. But, changing the system to an automatic adaptive system using artificial intelligence or rule-based systems, is considered a great improvement of the system which is one of the planned future works. In a self-adapting system, the changes of all users assigned to a specific persona, will be all aggregated and analysed. As an example, it can be analysed that what are the mostly used values for zooming or what are the mostly picked set of filtering objects. To perform this approach, time periods can be added too. In a real system, which is not just a prototype or a test, the number of users is a huge number. As the result, using all users' information from the beginning, not only can makes the system slower, but also may not be a useful approach. Since, including all data from the beginning

of time, may involve a great period of time and the size of the loading data will be so much and this results in slower system performance. Moreover, the most recent data are more trustable in case of identifying users' taste of their visualisation's view, because they are the most recent views which they preferred to have. As the result, time periods can be included in analysing users' changes.

3.3.1 Challenges

Challenge of same user another persona is one of the challenges of the system, which is about if a user is assigned to a persona when signing up in the system and by accessing to his/her view, s/he changes the view completely in a way that the view's characteristics is not based on his/her persona characteristics any more and is closer to another persona's. In this situation, the system does not recognise this change of persona and uses the updates for the first-chosen persona. This is a gap of the system which can be prevented using two approaches.

The first approach is comparing the views' elements' (such and zoom and filtering objects) values from the previous overview and the current overview when the user is about to log out. In this approach whenever the user wants to log out of the system, every visualisation view's task should be checked in order to get them compared with the previous values of them to see how much changes the user had made to his/her view. If many aspects of the visualisation's view has got changed, this can be resulted that the view has been transferred to a complete different view which may have to be assigned to another persona. A level of changing can be specified and by comparison of the previous and current user's visualisation's view's aspects such as, zooming, filtering, location and etc. this level of changing should be checked. If the value is greater than a specific amount, as an example if zooming has changed for 4 times smaller and the two of the filtering objects have changed as well, and two other

aspects of the view have had changes, so this new view needs checking. A notification can be sent to the developer to notify him/her that there may be changes of persona in one of the user's view. The developer can investigate and if s/he detects the assigned user's persona should be changed to another persona, s/he can do the change. Diagram 3.7 shows such an attempt.

Figure 3.7: The personas' modification diagram

This approach has two limitations. The first one is, if the user leaves the system without logging out; in this case time span, can be used. This will be discussed further. The other limitation is needing a developer in

order to check the process. Although, the process can be implemented as an automatic process without needing a human to take care of the process, it can be a risky task, since it needs analysing and usually human analysis is a better approach than analysing a system using a set of rules. On the other hand, using a developer to take care of this process, there may be lots of notifications from different users in a system with lots of users and this results in increasing time cost and error percentage.

The second approach is when asking the user if s/he wants to change his/her view, this time of using it; and if s/he approved it, s/he will have to fill a new questionnaire and his/her username will be assigned to a new persona based on the answers s/he will give to the questions and when logging in to his/her view, the view of the new persona will be presented to him/her.

Time span of analysing can be set in order to save the latest changes after a specific amount of time if the user closed the system without logging out.

Real-time system is a applications which process data as it comes in the system, without any delays [35]. My thesis prototype is not implemented as a real-time system (since it is a prototype rather than a complete system) but this can be considered as an improvement of the future works of the system. The implemented system is a pull system rather than a push system, because to insert the updates to the system they need to be requested from the database by refreshing the page (as an example). However, in a push system, the updates are imported to the system as they happen without needing any request from the client side. The reason is the number of users is not a big number since the system is just a prototype to test the requirements and the possibility of using the Shneiderman's tasks [57] for an adaptive visualisation system. As the result, making the system as a real-time system did not have any specific effects on the results and also

was a time consuming process and was not included in my system's top priorities. In this approach, each time the developer requests an update for the system, the reporting component's data gets updated and the new updates which users have made recently to their views will be added to the persona's update reporting visualisations. This request can be in form of signing into the reporting component or refreshing the component; therefore, the system is not a real-time system. This is considered as a system's limitation and is one of the future works which should be taken care of. When the system is not a real-time system and the number of users is a huge amount, there is a risk of neglecting the updates and as the result the changes the developer decides to make to the personas and their views is less trustable, unless s/he requests for updating the reporting component in small periods of time.

3.4 System Architecture

Once requirements engineering uncovers the characteristics and constraints of the system to be built, the architectural style or combination of styles that best fit those characteristics and constraints can be chosen [51]. This section proposes an architecture for the adaptive visualization system with the discussed characteristics. Figure 3.8 shows the global view of the system architecture. The browser

Figure 3.8: The global view of the system architecture

This adaptive visualisation system is a client/server architecture. It consists of a client side and a server side and a database. The server side and the database are located on the web server and are connected to the client side (the browser) via Internet.

The web server uses HTTP to respond users' request and serve the files which form web pages.

As mentioned before, there are two classes of users on the client side. One is the general users who will login, access and interact with visualised information, while the second is the developer who will get notified about the high-level patterns in system changes through a reporting component. The end users who interact with the visualisation user interface are referred by term "User" and the person who tracks the end users' interactions is called the "Developer".

Therefore, there is a developer component which takes care of interacting with the reporting component and observing changes and is connected to the database and the logical component of the system (the application server).

Using the database and the logical component of the software, the developer can observe the changes users make to their view via client side of the system. The required changes can be occurred by the developer to the database data using the browser (the developer interface).

All the changes that will be made to the users' views as well as users' information will be recorded in the database server and the reporting component that reports back the changes to the developer works using database functionalities (functional requirement 5 and 6).

The application server as can be seen in the schematic system architecture (figure 3.9) has the responsibilities of providing authentication, monitoring and reporting changes ability as well as communicating with the database. Figure 3.9 is the schematic system architecture.

Figure 3.9: Schematic system architecture

As the first step in this process, a visualization designer/developer may identify initial personas that they feel capture the needs of their target users. Typically, designers may only design one initial view for a single persona. These seeded initial personas are stored in the database with a set of initial configurations.

When a user signs up in the system and answers the sign up page's questions, the information will be recorded in the database.

Users can self-identify with these personas, and if they are only pro-

vided with one option, then the first users of a new visualisation will be shown a single common initial view similar to the current state of play in most visualisation systems.

Comparing these answers of the questionnaire and matching them with the initial seeded designed personas, the default values of such configuration options as the zooming level and filtering objects will be specified and saved; and by reading those values from the database, the initial view of the map will be presented to the user, automatically (requirement 1). Any changes to the zooming level and filtering objects of the map will be written on the database. The changes which the user makes to his/her initial view will be saved and reported back to the developer using reporting component, in order to help the developer to understand how accurate the original designed views were compared to the actual preferences of real users. The real benefit of this then becomes that the system can respond to common patterns of early interaction and propose new configuration sets for users that seem to have common qualities or needs. This may be done automatically, or via reports to the developer for manual consideration.

To build the project, two main parts are the general sections of the system. The front end and the back end. Each section consists of several components. The reporting and monitoring changes component (developers profile), the user experience component (users profile), the database server, the software logic component.

The front end of a system is part of an information system which directly interacts with users. It enables users to access and request the features and services from the back end of the system [60]. The front end in this project includes end-user's and developer's profiles.

The back-end consists of the software logic component and the database server in order to save the data and control the front end. The system tracks the changes which users make to the initial view in order to understand the problems of the initial view and report it back to the developer. Also all users' data should be saved too.

3.4.1 GUI Sub-system

As mentioned before, there are two types of users for the system. The end-users (referred by the term "user") and the developer.

End-Users' Profiles

To design the initial views for users, I needed some examples in order to understand what users expect the map to include. So as explained earlier in chapter 2, two types of persona are designed for the project. To understand which persona suits best to an actual user, questions will be asked from them when signing up in the system. For example: are they are looking for specific places such as stores or pharmacies, or do they already know their end destination and trying to determine the right path; how do they intend to reach those places: using a car or by walking?

These are the sample of questions which should be asked because the answers to these questions will clarify which of the designed user's goals are closer to the actual user's goal of using the map. As an example if the user is walking then a more local zoomed-in view with the nearby places will be more useful to him/her, as they will not be able to physically travel further afield.

Based on the user's answer to the questions discussed earlier, the initial view of the map will be presented. There are several factors which will make the initial views distinguished from each other. By controlling the level of zooming, the local or general views will be made (zooming). By filtering the information and marking the places which are the target of a user, the view of the map will be customised (filtering). By clicking on the marks, the details such as address of places can be added too (details on demand). There are seven tasks of users view of the visualisation which have the highest level of importance [57]. In this thesis four them are used which are:

- Overview: Gives an overview of the entire collection.

- Zoom: To zoom in on items of interest.
- Filter: Filters out uninteresting items.
- Details on demand: Selects an item or group and shows details when needed.

Figure 3.10 shows the user's functionalities with the different parts of the user interface using used case view [31]. The represented tasks are derived from the Shneiderman's Visual Information-Seeking Mantra [57]. As explained in the 3.2 section, each of the represented tasks, influence on the view to customise it based on the persona's characteristics.

Used case view defines the set of scenarios that represent some significant functionalities of the system with the end-users [31].

Figure 3.10: The end-users' used case view

Developer's profile

Developer's profile is the page which the developer can observe the personas and users' map view changes' report via reporting component. By analysing the results, he can make the changes to the personas and their assigned map views or the sign up questions, through his profile. In this way, he does not have to directly access the database or server application code to conduct the changes.

The Shneiderman's tasks [57] are used in the developer's profile, too. The developer's user interface includes the visualisations which include the persona's and users' views' data. The visualisations are used to help the developer to better analyse the users' interaction with their visualisation interface and identify the personas' characteristics which need changing. These visualisations also include Shneiderman's tasks; such as zooming, details-on-demand and etc. The system's developer's interface includes the overview task which gives an overview of the entire dataset of the system and by using zooming task, the developer can analyse a specific persona's data and filtering out other personas' data. Moreover, details-on-demand provides more supplementary information of the users' views' changes, so that developer can analyse them better. Details such as the time of a change.

Figure 3.11: The developer's used case view

3.4.2 Logic Sub-system

Logic component as can be seen in figure 3.9 takes care of the connection of the front-end and the database. The key roles of the server application are authenticating users and the developer to sign in to the system, transferring data to the database from the front-end and monitoring and analysing users' map views' changes and present them to the developer with contribution of the browser and Java Script. The logic component has different responsibilities which can be seen in figure 3.9.

Monitoring Changes Component

Figure 3.12 shows the monitoring changes functionality as a logical view [52]. The logical view is concerned with the functionality that the system provides to end-users [52]. Figure 3.12 just includes the main functionality of the system which is the map functionality. The "monitoring changes"

service uses the "Changing UI" sub system and will be instantiated by it because the visualisation changes will be submitted to the database and will be viewable later by the developer using database functionalities.

Figure 3.12: The logical view of monitoring changes functionality

Changing Process Component

The process of user interface change which includes two functionalities of applying and saving the changes, can be seen in figure 3.13 as a logical view [52]. The logical view represents what entities the component

includes and also what other components are connected to the changing process component. Before saving and recording the changes, the changes should be declared and monitored. Therefore, the changing process component uses the monitoring changes component.

Figure 3.13: The logical view of the users' interface changing process

The figure 3.14 shows the hierarchy of the discussed components connecting to one another as the process view [31]. The process view describes those activities that a system passes through when it exchanges messages with other systems or sub systems [31].

Figure 3.14: The users' interface changing process view of the system architecture

Reporting Component

The reporting component is about the component which observes the designed personas and users' views' changes. By saving all the updates users made to their views, the visualisation of the updates per the personas users are assigned to, is provided in the reporting component. The assigned personas to each of the users are saved in the database too. Moreover, the visualisation of the users' updates and the visualisations including the Shneiderman's tasks' updates per personas and users are provided in the component. The visualisations are provided using the history table of the database which keeps all the users' views' updates.

Using the reporting component, the developer is able to track the updates and decide about the changes each of the designed personas and their views need. The reporting component involves the developer experience which was discussed in section 3.2.2.

Authentication Component

Authentication is about identifying eligible users to enter to their accounts using a username and a password. In a customised visualisation system, since the views are customised base on the users' answers to the questionnaire, the users' accounts are private. To make the users' account private, the authentication component is used. The authentication component follows the approach of checking the entered username in the database to see if it exists. In case of existing, the password will be checked to make sure that the entered password is equal to the saved password for that username in the database.

3.4.3 Database Sub-system

In my system database, I include a table for users' information and a table for keeping the history of changes users make to their map view. When users change their map view information, these new information will be

saved in users' table instead of older information. So, the previous information will be transferred to the history table and will be saved there. To summarise, my designed database for my system includes two tables, one for keeping the most updated of all users' information (so that every time a user logs in, the most recent will be presented to him/her), and the other one is for keeping all of the history of all users and personas' information and map view changes.

Using users' table, each time a user logs into his/her account, s/he will have access to his/her most recent updated map view.

Later the history table will be used for the reporting component. Using the history table and the reporting elements, the developer can observe the sequence of changes of each persona.

Figure 3.15 compares the discussed system architecture design with the list of system requirements and clarifies which section of the system design is related to which requirement. By providing this table I could make sure that I have included all of the system requirements in system architecture design.

	User's Profile	Developer's Profile	Authentication	Monitoring Changes	Applying and Saving Changes	Reporting Component	Data base
Requirement 1	✓		✓				✓
Requirement 2	✓			✓			✓
Requirement 3	✓			✓	✓		✓
Requirement 4	✓						
Requirement 5		✓		✓		✓	✓

Figure 3.15: The relationship between the system components and the system requirements

The discussed architecture design is then used to subsequently develop a web-based map visualisation system as a proof of concept prototype, using a web server, a database and a server-side web application. I used a client/server architecture. The system uses a Google Map JavaScript API to generate user interface widgets that creates the easy-to-use drag and drop functionality. Next chapter explains the system implementation process in details.

Chapter 4

Implementation

This chapter explains the implementation process of the system. The system implementation is divided to two phases. After the first version of the system completion, I performed a heuristic evaluation that will be explained in detail in the next chapter. But, according to the first heuristic evaluation, I conducted a few changes in the system functionality and its database. In this chapter the implementation of the final version of the system will be discussed first and then I will review and compare the previous versions as well. This chapter also addresses the technologies used for the system implementation.

According to the defined limitations in chapter two, a list of expectations can be helpful in order to define some implementation priorities and adjust a to-do schedule.

Overall in terms of front-end, the system consists of several sections which are the log in section, the sign-up section, the user interface section which is called the index page and the developer and the reporting pages. All the pages have two sides of front-end and the back-end parts which include their client side and server side codes.

Front End is the user interface which consists of the map as the visualisation example and the user and developer's interactive elements. I chose

map as the system's visualisation example according to my designed scenarios, any other type of visualisation is replaceable. The scenarios were about a delivery man who needed a general view of Wellington in order to be able to deliver goods to different places and a student who needed a local view of his/her location to access the nearby health centers.

The concerns with the user interface are being a responsive, easy-to-use and interactive user interface. Because the visualisation which I chose is a map, the initial location of the user should be based on his/her actual location.

As my map visualisation I used the Google Map JavaScript API, but it needed some customisations in order to be usable as my intended user interface map. So as the result the list of the critiques of the system front-end based on the system requirements are:

1. Responsive user interface (directly according to requirement 6).
2. Easy-to-use and easy-to-interact user interface (indirectly according to requirement 2).
3. Auto find users location (latitude and longitude) (requirement4).

Back End consists of the server side programming and the database. Most of the system requirements should be covered in this section.

1. Fetching users' interface data from the database each time a user signs in and saving new data in the database each time the user logs out has a great role in the system (according to requirement 3 and requirement 2 and requirement 1).

[this is the most important ability of the system which makes all the requirements and adaptivity of the system available].

2. The process of users' sign up and customising their sign-up information is the core functionality of the system (according to requirement 3).
3. The database view and trigger to save and transfer all the updates and using them to produce charts in order to help the developer to mon-

itor updates, has the great role in evaluating the system and its design performance (according to requirement 5).

The sections of this chapter are based on the 3.8 figure of the third chapter which represents which system components are providing the system requirements of the system architecture design. As the result, implementing those components provides the designed system I required. Therefore, this chapter explains the implementation of those components respectively.

4.1 Users' Profile and Logic Component

Users' profile consists of their log in and sign up pages and the page which includes users' map view. Because the users' profile includes the users' map view which is strongly connected to the logic component of the system, this section explains these two components together.

4.1.1 Login Page

The login page takes care of checking user's username and password correctness in order to pass them to their profile. The connection between the database and the server side of the system is ado.net connection.

I chose ADO.NET connection because ADO.NET data components in Visual Studio encapsulate data access functionality in different ways that helps to program more quickly and with fewer mistakes [48].

Also, in terms of scalability, ADO.NET provides scalability by conserving limited resources. Because ADO.NET applications provide disconnected access to data, it does not hold database locks or active database connections for long durations [48].

Figure 4.1 shows the login page of the system. As can be seen in the picture, there are two sections in the login page. The users' section which leads them to their map view and the developer's section which leads

him/her to the chart page which the page s/he can monitor and analyse the personas and users' map view changes.

Users' Login Section	Developer's Login Section
<input type="text" value="username"/>	<input type="text" value="Developer password"/>
<input type="text" value="password"/>	
<input type="button" value="Log in"/>	
<input type="button" value="click to sign up"/>	<input type="button" value="I am the developer"/>

Figure 4.1: The system's login page

If a user has not signed up the system yet, will be directed to the sign up page.

4.1.2 Sign Up Page

In previous chapters, I explained about the users' information which are needed to form the map view. The sign-up process is such an attempt. The questions provided in sign-up page are made based on a purpose. The map's zooming level and the marks are specified based on the provided indirect questions on the sign-up page. The sign-up questions and the provided options can be seen in figure 4.2.

How do u intend to reach to your destination?

Walking ▾

Where are you looking for?

Restaurant ▾

username

Username required

password

Password required

Repeat password

Sign up

Figure 4.2: The system's sign up page

The validation of users' chosen id and password is checked using ASP.Net's regular expression validator. The `RegularExpressionValidator` class validates whether the value of an input control, which is the username here, matches the pattern specified by a regular expression or not [41]. The pattern I specified as the username is the combination of 'a' to 'z' capital and small letters and 0 to 9 numbers. If the entered username does not match with the pattern, the validator class provides an error message. Also, by using `RequiredFieldValidator` class, I can make sure that the username, password and confirmation of the password input fields did not remain empty.

The questions and choices of the sign-up page is loaded from a text file. The back end reads the input from the text file into a string array. Then the question label fields and drop down lists will be initialised. Using this

approach, if the questions and the information received from users need to be changed, just the text file content will need editing, not the code.

Question number one asks about the approach of the user to reach to his/her destination and based on the user's answer, clarifies if the user will need a local or a general view. The chosen option will be compared to the characteristics of the local and general views and based on the comparison answer, the level of zooming will be specified. Local users have more zoomed in views and general users have more zoomed out views. The second question's answer specifies the information which will be shown on the map. Based on the users' choices, the marked places and their detailed information will be specified.

When a user signs up into the system, the username, password, zooming value and sets of chosen places will be saved in the database using ADO.NET connection. If the user chooses 'walking' option as the first question's answer, that means the user is looking for local and nearby places. So, the zooming level will be a smaller number to make his/her view more zoomed in. I used zooming value of 5 for local users because a smaller number would be so restricted and detailed view and a bigger number would not be a local view. The second option for this question is 'By car', which specifies that the user can be determined as a general view user and has access to further places. As the result, the presented view would be a view with the zooming value of 10. I set the general view's zooming value to 10 because, a bigger number would make the city view small and could not include all the needed places.

Answering the second question specifies what type of places should be presented on the user's map view.

4.1.3 Index Page

This is the main page of the system which involves users' view. Both the sign up and log in pages redirect the users to this page. This page includes

a map user interface (based on Google map JavaScript API) and a set of places which can be selected to be shown on the map.

Libraries used for this page are the JQuery [34], Bootstrap [10] and Google map API library [26]. Bootstrap library is used in order to have a responsive user interface. Responsive user interface is kind of user interface which is adaptable based on the size of the screen. Using Bootstrap library, in my user interface, both the size of map and the page elements are adaptable according to the screen size change. This is obtained by dividing the index page into two rows and each row, into two columns using Bootstrap approach.

I also used an addDomListener [7] which adds a listener to the window object, which as soon as the resize event is triggered executes the function of resizing the map.

The Google map JavaScript API library requires a key in the library string which is obtained by signing up into the Google map website and identifying the type of map I intend to use. Without this key the map is not loaded. This library also includes all the cities different places and their detailed information such as addresses. To customise the map, I clarified the signed-in user's latitude and longitude as the initial coordinates of the map using Geolocation API. So, using the user's current position, I pass the latitude and longitude as the map's initial latitude and longitude and set it as the map centre.

I changed the map feature in order to involve the name of streets but not places. Because identifying every detailed information is not going to help users and makes their map's view crowded and confused too. Accordingly, I used a 'styler' array to involve map features of 'featureType' and 'elementType' and make their visibility off. A 'Styler' array is the style rules to apply to the selected map features and elements [8].

An 'elementType' is the element to which a 'Styler' should be applied. An element is a visual aspect of a feature on the map. Example: a label, an icon, the stroke or fill applied to the geometry, and more [8].

And a 'featureType' is the feature, or group of features, to which a 'Styler' should be applied [8].

I specified 'poi' as the 'featureType' which involves all the names of different places as the labels and set their visibility off. In this order, the map only involves the street names and just the places which users want to be specified.

I also restricted the map coordinates and set the bounds to only Wellington city and disabled the search capability. So, the other parts of the world became inaccessible since my test area is just the Wellington city. So, whenever users try to drag the map to other cities or parts of the world, they will be returned within defined bonds.

When each user signs up, his/her information is recorded in the dataset and whenever s/he logs in and is referred to the index page (either directly after signing up or logging in), his/her map's information is read from his/her data record. This way the level of zooming and user's specified places are specific to each user and also can be easily updated each time the user performs changes.

So, when a user signs up and answers the questions, the zooming value and set of specified places will be passed to the index page using session. These values will be substitute as the map's zoom value in the initialisation method.

The session variables are variables maintained on server side by ASP.NET runtime. Each user is identified by a unique Identification number which is called Session ID. This session is stored in a cookie on client side after the first user request. when the client posts back a page, this cookie is available in the request header, so the server can know this user request is coming from which user. Moreover, user specific information can be stored in session variables, which will be available on server side [39].

The set of users' preferred places will split based the Comma signal and saved in a variable after it is retrieved from the database and received from the back end using session variable. Then a request which contains

the latitude and longitude of the map centre, the radius within which the requested place should be specified, and the array which contains user's requested places, will be made. Then using Google places services, all the places with identified characteristics in request array will be identified and passed to create marker function. The function creates marker for the specified locations and there is an add-listener [6] which listens to clicking function in order to show the place details as a tool tip.

The users specified marked places can be changed using checkboxes. I have provided a number of checkboxes which several of them can be clicked at the same time. By writing a function I was able to check several checkboxes at the same time and insert all those places into an array. And, later they will be passed to a different request in order to be passed to Google places services function and creating markers for the new request list. This will change initial marked places and as the result, the user's view will also change.

Zooming value can also change using either plus and minus buttons on the map, or using a mouse.

Whenever the user clicks log out button, all the changes to zooming value and set of selected places will be passed to the back end using session. Using ASP.NET and ADO.NET, the changed values will be replaced by the previous values in the database. This way, the next time user logs in his/her view, s/he will have the changed view.

The client side of the system can be seen in the prototype proof of concept system demonstrated in Figure 4.3. As can be seen, there are marks of users' preferred places on the map and they can be changed using checkboxes as well as the maps' zooming level (functional requirement 2 and 4). The user interface can responsively change according to the screen size (non-functional requirement 1). This is the user interface designed for a test persona.

Figure 4.3: The end-users' map view page

4.2 Developer's Profile

Developers' profile is the page in which the developer can make the changes to the program and persona characteristics without needing to have direct access to the code behind. There is an interface as can be seen in figure 4.4 in which the zooming level value and the filtering object of personas can be changed. In this way, updating the system is more easier and the developer does not need to understand and change the code behind.

Persona One	Persona Two
<input type="text" value="Zooming Level Value"/>	<input type="text" value="Zooming Level Value"/>
<input type="text" value="Object Set"/>	<input type="text" value="Object Set"/>
<input type="button" value="Submit"/>	<input type="button" value="Submit"/>

Figure 4.4: The developer's interface to modify personas' characteristics

Moreover, providing the text file of sign up process questions gives the same opportunity to the developer to edit a part of system which has a great probability of changing without needing to change the system's code behind.

4.3 Database

The database consists of two tables and a view. One table includes users' information and the other table is used to hold all the updates users make to their information each time there is a change in their map view.

The users' table holds information about users' id, password, the persona which the user is assigned to ,zooming value and the set of objects which are marked on the map. At first I intended to make two tables to separate users' information from their map's view information, but this approach did not have any specific effect on system functionality and I realized can result in making the connection between the back end and the database more sophisticated. So, one table holds both information of users and their map's view.

The zoom column of the users' table refers to the zooming attribute of the map's view and the zooming task of Shneiderman's Information Seeking Manta [57] and the filter column refers to the set of information which will be declared on the map which comes from the filtering task of the Shneiderman's paper [57].

The data-history table includes all the users' updates in form of information of their user id, the persona they are assigned to, the level of zooming, their set of objects, date and time of the update and the action if it is an insert or an update and the revision. Saving the map's view information in the data-history table is used for analysing how this information every time changes and the comparison between them which can help the developer in analysing how the personas' information should be changed in order to be more preferred by the users. The user' id is for identifying users from each other and the action column identifies what type of update has been made to the map view. Revision column shows the number of times which a user has made changes to his/her map's view information and the date and time of the changes will be saved too. Using revision, date and time and the action columns helps the developer to

better track the updates personas and users make to their views. As an example, if a user's revision is a big number, this can be a clue that user has a higher possibility of dissatisfaction and if this pattern is repeating within the persona with other users who refer to the same persona, this can be concluded that the referred persona needs changes. To do this a bar chart is provided which directly gets the data from the data-history table and shows the number of changes each persona and user has made to his/her view. These changes are just the ones which their type of action is 'update'. Because 'insert' action refers to the first time a user makes his/her account.

The data-history-changes view joins the two tables in order to list all the users who have updated their information. This view is used for separating the users who has made changes to their map's views from those who did not. This view gives the ability of identifying the personas which the related users have mostly changed based on the number of updates of their related users.

4.4 Reporting Component

4.4.1 Chart Page

This page includes a chart which represents the numbers of times each persona has made an update to his/her map's view. Usually charts are helpful in providing a quick review of an important factor of a database. This page's bar chart is also trying to introduce the persona which requires the most number of changes in its views and as the result, the personas who need to be changed in order to become closer to their users' preferences and requirements.

In the first page, the bar chart includes the personas and number of updates their users made to their map view. This visualisation also includes zooming and details on demand. The details on the demand is

activated by hovering on the bars which gives more specific information and tells the exact number of updates of each persona. This gives the possibility to the developer to analyse all the personas' updates and specify which persona has the most number of updates. As the result which persona needs changing the most. As an example of my system, figure 4.5 which is the chart of number of updates per each persona, represents that 'general/delivery' persona's view had been modified more by its users than 'local/health' persona. It can be interpreted from the chart that 'local/health' persona's users were more satisfied with the persona's designed view. By using other charts, the 'general/delivery' persona characteristics which need to be updated to make a more preferable map's view for the persona's users, will be clarified.

Figure 4.5: The developer's reporting chart which represents the number of updates per personas

To make the chart I used the Post function of Ajax [33] to transfer data from the back-end to the browser and make the chart using Google chart library. The ajax post method loads data from the server using a HTTP POST request [33].

There is a zooming function in the bar chart which shows all the users who are assigned to each persona when zooming on each persona bar in the bar chart. Because the map views are designed based on the persona characteristics the overview of the visualisation represents the interaction of personas with the map view, generally. Then zooming on the chart changes each bar which represent a persona to the users who are specified to that persona. This gives the possibility of analysing users' interaction with the user interface, too.

Using the chart in figure 4.6, the users which had more number of updates can be clarified. These users are the ones which can be studied more to detect their views issues. The updates can be on the zooming level, filtering objects or the users' location. If many of the users modified all the three visualisation's tasks, this can mean the designed persona and its view have major limitations. In this case using the charts which represent the number of each task's values used by users can be helpful.

Number of Updates per users of Persona One

This chart shows the number of updates each user who are assigned to persona one have.

Figure 4.6: The developer's reporting chart which represent number of updates per users assigned to one of the personas

Figure 4.7 and 4.8 charts which show the updates of zooming level and filtering object for each of the personas, separately. The charts will be represented by clicking on the zooming and filtering tabs in the navigation bar. This gives the possibility of comparing all zooming and filtering values to each another and analysing the most favourite one for each group of users specified to a persona.

As can be seen in the figures, zooming value 15 and the set of filtering

objects which includes 'Restaurant', have been mostly used by the users of persona number one. If the default values of these tasks are the same with the mostly used values, it can be interpreted that the chosen values were suitable for the designed persona. Other chosen values can be studied as well to check if they are the final chosen values or the ones which were chosen first and changed later to the default values. To do such a study the time stamps can be used.

Filtering Objects Used by the Users Assigned to Persona One

Figure 4.7: The developer’s reporting chart which represents the number of each set of filtering objects used by the end-users

Figure 4.8: The developer’s reporting chart which represents the number of each zooming value used by end-users

There is also a search component in which the developer can search in the data-history table. The searching option is not restricted to a specific data or a specific column. As can be seen in figure 4.9 , the column and the data can be chosen in the searching interface. I did not include the data-history table in the developer's interface since when the the number of users is a great amount, the table can not fit into the developer's interface page nicely. The searching component provides the possibility of accessing and searching the data-history table for the developer without needing to include it in the interface.

Search Page

▼ action	▼ revision	▼ dt_datetime	▼ idusers	▼ pass	▼ zoom	▼ filter	▼ persona
update	2	2016-11-02 15:09:09	user1	123	14	bus_station	local/health
update	2	2016-12-22 14:23:43	user10	321	16	atm	local/health
update	3	2016-12-22 14:24:04	user10	321	16	bank/atm	local/health
update	1	2016-10-10 14:38:14	user11	123	3	atm	general/delivery
update	2	2016-11-02 15:07:39	user2	123	18	atm	general/delivery
update	3	2016-11-02 15:08:31	user2	123	19	atm	general/delivery
update	4	2016-12-22 10:51:07	user2	123	19	atm/bank	general/delivery
update	5	2017-01-12 12:52:31	user2	123	19	atm	general/delivery
update	1	2017-01-01 21:48:51	user20	123	15	restaurant	local/health

Search a specific data:

From

Figure 4.9: The database search component

There is also a table which shows the visualisation information in text and is derived from the data-history table. For making the table I used the ASP.NET datatable [38] and after connecting to the database and fetching

the data, I made a table and insert the data. The table includes all information and history of updates of each user.

As mentioned before, the date and time column of the database is used for extracting time stamps to be used by the developer. As an example, a specific date such as 20th of January if the user's log in time is 1:30:15 and the first update is 1:30:20, this represents the update which has been initially done to the user's view.

Figure 4.10 shows the relationship of the system's requirement and the system's pages which include the functional components. It represents that in implementation of each of the system's requirements, which pages and sections of the system were involved.

	Login Page	Sign-up Page	Index Page	Developer's Profile	Database	Chart Page
Requirement 1	✓	✓	✓		✓	
Requirement 2			✓		✓	
Requirement 3			✓	✓	✓	
Requirement 4			✓			
Requirement 5				✓	✓	✓

Figure 4.10: The relationship between system's components and the system's requirements

4.5 The Previous Versions of the System

4.5.1 First Version

The first version of the system did not include the persona which each user is assigned to, in the database. And, as the result, did not include personas in the reporting component and the charts. When I did the first evaluation, I realized to evaluate whether the personas need to be changed or not, and what factors of personas need changing, I should have saved the persona which each each user in assigned to, and should have designed the reporting visualisations based on the persona. Because initial map views are designed based on personas' needs and preferences and only evaluation of users' changes does not help in understanding whether the personas' desinged initial views need changing or not. As the result, I changed the database and the reporting visualisations. I changed the charts from number of updates, number of zooming value and filtering object per each user to number of updates per persona and by zooming on the chart, number of updates of each user can be seen too. I also changed the zooming and filtering visualisations from per users to per personas. The below figure is presenting the previous version of the system's visualisation page. Figure 4.11 also represents the previous data-history table of the system.

user2	insert	1	15	restaurant	2/11/2016 3:07:07 PM
user2	update	2	18	hospital	2/11/2016 3:07:39 PM
user2	update	3	19	atm	2/11/2016 3:08:31 PM
user1	update	2	15	bus_station	2/11/2016 3:09:09 PM
user3	insert	1	5	hospital	2/11/2016 3:09:27 PM
user3	update	2	16	hospital	2/11/2016 3:09:54 PM
user4	insert	1	15	hospital	2/11/2016 3:10:18 PM
user4	update	2	18	hospital	2/11/2016 3:10:32 PM
user4	update	3	14	bank,atm	2/11/2016 3:10:58 PM
user4	update	4	14	school	2/11/2016 3:11:22 PM
user4	update	5	14	movie_theater	2/11/2016 3:11:41 PM
user4	update	6	14	park	2/11/2016 3:12:00 PM
user5	insert	1	10	restaurant	2/11/2016 4:01:38 PM
user6	insert	1	12	restaurant	2/11/2016 4:02:38 PM
user7	insert	1	14	restaurant	2/11/2016 4:03:25 PM
hihihi	insert	1	15	restaurant	11/12/2016 5:35:43 PM
fer2	insert	1	15	restaurant	20/12/2016 5:54:43 PM
fer2	update	2	15	restaurant	20/12/2016 5:54:55 PM
fer2	update	3	15	restaurant	20/12/2016 5:55:54 PM

filter	filter_count
Restaurant	11
hospital	6
stores	6

Figure 4.11: The previous version of the developer's interface representing data-history table

Figure 4.12: The previous version of the developer's interface representing the visualisation of the number of end-users' interactions with their views

In this pages, the bar chart's data comes from data-history table of the database and fills a datatable in Microsoft Visual Studio. The table includes the history of every user's actions. The actions can be either 'insert' or 'update'. Therefore, using an algorithm, I made a two-dimensional array to insert all the users' id who the 'update' action was assigned to them and the number of updates they had. I used serialization [42] to send the array to the front end in order to show the data on the chart. Serialization is the process of converting an object into a form that can be transported. On the other side, deserialization converts a stream into an object. Together, these processes allow data to be easily stored and transferred [42].

At the front end of the chart page, the sent objects are used after changing them to variables. Using Google visualisation API, the variables are added to the chart as the columns and the chart was generated.

Figure 4.13 presents the previous system architecture as a simple diagram and shows the previous relationships between the database tables and the system components.

Figure 4.13: The system architecture as a diagram

Figure 4.14 shows the process of reporting the changes to the developer in previous system architecture. It had the same approach in terms of each the user would have signed in, the changed user interface would be presented to him/her.

Figure 4.14: The Diagram of Saving and Reporting the Changes

4.5.2 Second Version

One of the most important changes of the second version of the system to the final version, was deleting the data-history table of the pages and building a search page. In the search page, the developer can search in

the data-history table. The searching option is not restricted to a specific data or a specific column. As could be seen in figure 4.9 , the column and the data can be chosen in the searching interface. Therefore, including the whole data-history table in the interfaces is not needed any more, since searching component has made the possibility of accessing to data, for the developer. Moreover, including the data-history tables when the number of users is a great amount, is not a good approach since it can not fit into the developer's interface page nicely (In case of future scalability of the system).

Therefore, I added the searching component and also a navigation bar in order to be able to easily move between reporting component's pages. The filtering and zooming pages are also included in the navigation bar as well as every single page of the reporting component. This has made the access to different pages of the reporting component, easier for the developer. Moreover, the navigation bar changed the pages to nicer and cleaner pages. The previous developer's interface can be seen in figures 4.15 and 4.16. Figure 4.15 shows how would the developer's interface looked like including data-history table and figure 4.16 shows the developer's interface before including navigation bar and with using hyperlinks to navigate to other pages, instead.

action	revision	dt_datetime	idusers	pass	zoom	filter	persona
update	1	10/10/2016 2:38:14 PM	user11	123	3	test1,test2	p1
update	2	17/10/2016 1:19:10 PM	user25	222	15	restaurant,bank	p1
update	3	18/10/2016 2:26:51 PM	user6	222	17	restaurant,bank	p1
update	4	18/10/2016 2:28:36 PM	user6	222	15	gym,	p1
update	5	18/10/2016 2:29:55 PM	user6	222	15	gym,	p1
update	6	18/10/2016 2:49:20 PM	user6	222	15	gym,	p1
update	7	18/10/2016 2:50:23 PM	user6	222	15	gym,	p1
update	8	18/10/2016 3:13:39 PM	user6	222	15	gym	p1
update	9	18/10/2016 3:24:21 PM	user6	222	15	park,bar	p1
update	10	18/10/2016 3:38:52 PM	user6	222	15	park,bar	p1
update	11	18/10/2016 3:41:29 PM	user6	222	15	bar	p1
update	12	18/10/2016 3:56:12 PM	user6	222	15	gym,spa	p1
update	2	2/11/2016 3:07:39 PM	user2	123	18	hospital	p1
update	3	2/11/2016 3:08:31 PM	user2	123	19	atm	p1
update	2	2/11/2016 3:10:32 PM	user4	123	18	hospital	p1
update	3	2/11/2016 3:10:58 PM	user4	123	14	bank,atm	p1
update	4	2/11/2016 3:11:22 PM	user4	123	14	school	p1
update	5	2/11/2016 3:11:41 PM	user4	123	14	movie_theater	p1
update	6	2/11/2016 3:12:00 PM	user4	123	14	park	p1
update	2	22/12/2016 10:51:07 AM	user8	123	15	restaurant	p1
update	4	22/12/2016 10:51:07 AM	user2	123	19	atm	p1
update	7	22/12/2016 10:51:08 AM	user4	123	14	park	p1
update	2	22/12/2016 10:51:08 AM	user5	123	10	restaurant	p1
update	2	22/12/2016 10:51:08 AM	user7	123	14	restaurant	p1
update	1	1/01/2017 9:48:51 PM	user21	123	15	restaurant	p1

This chart shows the number of updates each users who are assigned to persona one have.

Figure 4.15: The system's previous version's developer interface which includes the number of updates per users' chart and the data history table

This chart shows the number of updates each persona had overall, to get the number of updates each user had, please click on the bar each persona bar. You can also get the number of each [zooming level value](#) and [filtering object](#) that has been used by each persona.

Figure 4.16: The system's previous version's developer interface which includes the number of updates per personas' chart and the tale which represents these numbers

The navigation bar is a horizontal one, since in case of using a horizontal navigation bar instead of a vertical one, the size of the charts can easily increase by the increase of number of users.

4.6 Tools and Languages Choice

The first requirement of the project was a map API. Later, in the alternative implementation section, all the tested APIs will be explained, but the Google Maps API was the one I chose and built the project on top of it. To develop the API, I used JavaScript and JQuery programming languages.

These all were used in building the front end of the project.

Google Maps API The Google Maps API gives developers several ways of embedding Google Maps into web pages or retrieving data from Google Maps and allows for either simple use or extensive customization [26]. There are several types of API. The web APIs are:

- Google Maps JavaScript API
- Google Static Maps API
- Google Street View Image API
- Google Maps Embed API

What I used for the project is the Google Maps JavaScript API.

Google Maps JavaScript API is a web page or application which displays a map using the Google Maps JavaScript API. It displays a Google Street View Image API panorama using the Google Maps JavaScript API. The Google Maps JavaScript API is simple to use and provides a wide range of services for data visualization, map manipulation and directions [27].

JavaScript is a lightweight and interpreted programming language with first class functions [43]. First class functions mean functions and objects are treated as the same thing. Functions can be stored as a variable inside an object or an array as well as it can be passed as an argument or be returned by another function. That makes function "first-class citizens in JavaScript" [58]. JavaScript is a dynamic scripting language which supports object-oriented style [43].

jQuery is a fast, small JavaScript library. It makes HTML document event handling, animation, and Ajax much simpler with an easy to use API that works across all the browsers [34].

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites [10].

As the back end programming language, to connect to the database and write on or read from it, I used ASP.NET.

ASP.NET is a web development model that includes the services which are necessary for building enterprise class web applications with a minimum of coding. ASP.NET is part of the .NET framework. Coding ASP.NET applications requires access to classes in the .NET framework. The coding can be in any language compatible with the common language runtime (CLR), such as Microsoft Visual Basic [37].

I aim used MySQL as my database because it is a common database which can be used for web applications and reading the table's data and reporting it in a readable format is possible too.

MySQL is a most commonly database used for web applications and embedded applications and has become a popular alternative to other database systems because of its speed and reliability. MySQL can run on UNIX, Windows and Mac OS [53].

4.7 Language and Tools Critique

The first challenge I faced for my system, was passing values from back end to front end and vice versa. My system consists of different components and each component has a duty to do. Each component includes several variables and these variables may be used by several components. As

the result passing variables and values was one action which should have repeated many times. I performed passing them not only from database to front-end through back-end, but also from front-end to the back-end in order to be saved in database.

There are several approaches to pass information between pages. I could use a query string. A query string adds information to the URL of the target page. The query string can be used when using a Hyperlink control to navigate to another page or when using the Redirect method to redirect to another page [62]. So, this approach cannot be used always.

Using session makes it possible to store information in a way that is then accessible to all ASP.NET Web pages in the current application. On the other hand, session takes memory, and the information is stored until the session expires, which can be more than the time which is needed for simply passing information to the next page [40].

The other approach is serialization which is the process of taking an object and converting it to a format in which it can be transported across a network or be stored. Deserialization is the process of reconstructing the serialized information to the original object. Therefore, the process of serialization allows an object to be serialized, transferred across the network or saved in a storage location such as the ASP.NET cache, and then be reconstructed for use later [14].

I used all these approaches according to the situations in which they were suitable for the situation.

The second challenge of the implementation process using a Linux server with my ASP.NET application. Since I implemented the system in ASP.NET, It was better if I could use a windows server to upload the program to the server. On the other hand, university servers are Linux servers and to upload the program to the server I had to search and find the best and easiest approach to be able to upload the program. A good option, which I got familiar with, is Docker.

Docker containers wrap the software in a complete filesystem that con-

tains everything which is needed to run the system. This includes, code, runtime, system tools, system libraries and anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment [19]. So using Docker, there will not be problem of using windows server side programming language with a Linux server.

4.8 Alternative Implementations

The first choice of the map API of my project was the Leaflet API instead of Google Maps API.

Leaflet is an open source JavaScript library for friendly user interface and interactive maps. It has all the mapping features which most developers ever need. Leaflet is designed with simplicity, performance and usability. It works efficiently across all major desktop and mobile platforms and can be extended with lots of plug ins. It has a beautiful, easy to use and well documented API and a simple, readable source code that is nice to be worked with [4].

Leaflet includes tiles as the basement of its map. The tiles are customisable and so the map can be restricted to a certain geographical area. The adjustment of the initial geographical point when the map loads and the initial level of zooming are all customisable using JavaScript programming language in Leaflet. To control the presenting information on the map, there are several kinds of tiles which can be replaced by the default tile. I replaced the default tile with a tile which included less information on the map in order to be able to add a JSON dataset to the map and be able to control the information which should be shown on the map and so be able to filter the map's information. There are many kinds of tiles in OSM project. I used a simple tile from the OSM tiles collection.

OpenStreetMap is a project with the aim of creating a free world map. Over 200,000 mappers collect worldwide data on footpaths, parks, cycle ways, roads, railways, rivers, forests, houses and anything else that is commonly shown on maps. Because they collect the data themselves and not from existing maps, they possess all the rights to use and distribute it. OpenStreetMap data can be used without licensing costs or notifying anyone and seeking permission [49].

I picked one of the simple tiles of the OSM collection with just names of streets and roads and replaced it with the default tile of the Leaflet API, because the default tile of the Leaflet included all the building names and all locational information of different places on the map which prevented me to be able to filter the information based on users' preferences for their initial view of the map. Then I added a GeoJSON dataset to the map which included different building names of Wellington city with their geographical locations.

GeoJSON is a format for encoding a variety of geographic data structures in form of a Json file. GeoJSON supports different types of geometry shapes such as polygon or line strings, to clarify a specific area on the map. In GeoJSON, an object consists of a collection of name/value pairs. For each member, the name is always a string. Member values are either a string, number, object, array or one of the literals: true, false, and null [29].

I searched for a complete GeoJson dataset of places of Wellington city. The best one which I found still did not involve most of the places. So I thought about finding a different approach and looking for a map API which does not need to change the map tiles in order to change the map information. I wanted to find a map API which the map information could be controlled using map features and JavaScript programming language. As the result I switched to Google Maps API.

Comparing Leaflet and Google Maps API clears that using the growing collection of plugins and their updates, makes the level of control over

Leaflet unlimited [16]. Leaflet has the ability to be integrated with free and open mapping alternatives such as OpenStreetMap, MapQuest Open or MapBox. Leaflet provides direct tile access which Google does not. As the result Leaflet gives the ability to users to load the map in various types such as including landscapes, green areas, volcanoes or many other types of maps. There are many options of layers for Leaflet library which makes the users able to find the one that best fits their usage and budget [36]. Leaflet also gives the ability to design aspects of the map implementation without worrying about licensing [16]. However, I had to switch to using Google Maps API because I did not need using various types of tiles in my project but needed the detailed and complete data points of Wellington city and controlling them. Google Maps comes with complete set of data [54] about the city points.

To use the Leaflet library, I had to use a complementary GeoJSON file which contains detailed data about city places and all geographical points of Wellington city. Because Leaflet map library does not include detailed information about the city points by default and the dataset needs to be added as a tile on top of the map. I tried several datasets but they were not complete datasets. As the result, I chose Google Map API instead, which contains information about every points of the city by default.

Chapter 5

Evaluation

This chapter explains the evaluation process of the system. I conducted a heuristic evaluation in two steps and a cognitive walk through which will be explained further in this chapter.

Evaluation is a sequence of acts with the aim of identifying the limitations and the improvement a system or a prototype can have. Evaluation is a key research challenge and Heuristic Evaluation is one recognized method.

5.1 Performance Metrics

Heuristic Evaluation is a well-known and popular evaluation method within Human-Computer Interaction (HCI). The method is used to judge how well the system meets its requirements and goals of some predefined heuristics by the usability experts. The method is easy to learn and apply, it can be used during all phases of development and it requires little time and other resources. Furthermore, it can be modified in various ways to improve the result by, for example, including domain experts or end users, including specific user considerations, or by performing specific tasks [22].

There are six groups of heuristics which can be seen in figure 5.1 and 5.2. They are the reference heuristics which are classified based on the

papers they belong to.

Heuristic
Visual Information Seeking Mantra [6, 17]
Overview first
Zoom
Filter
Details on demand
Relate
Extract
History
Ergonomic Criteria for Hierarchical Information Visualization Techniques [8]
Limitations
Cognitive complexity
Spatial organization
Spatial orientation
Information coding
State transition
Orientation and help
Navigation and querying
Data set reduction
Ten Usability Heuristics [12, 20]
Visibility of system status
Match between system and the real world
User control and freedom
Consistency and standards
Error prevention
Recognition rather than recall
Flexibility and efficiency of use
Aesthetic and minimalist design
Help users recognize a, diagnose and recover from errors
Help and documentation

Figure 5.1: The reference heuristic table [22] (part 1)

Perceptual and Cognitive Heuristics [23]
Ensure visual variable has sufficient length
Do not expect a reading order from color
Color perception varies with size of colored items
Local contrast affects color and grey perception
Consider people with color blindness
Preattentive processing benefits with field of view/use preattentive visual variables wisely
Quantitative assessment requires position or variation in size
Preserve data to graphic dimensionality
Put the most data in the least space
Remove the extraneous (ink)
Consider gestalt laws
Provide multiple levels of detail
Integrate text (with graphics) whenever relevant
Ergonomic Criteria for Interactive Systems [16]
Prompting
Grouping and distinguishing items by location
Grouping and distinguishing items by format
Immediate feedback
Legibility
Consiseness
Minimal actions
Information density
Explicit user actions
User control
Flexibility
User's experience
Error protection
Quality of error messages
Error correction
Consistency
Significance of codes
Compatability
Knowledge Tasks [1]
Expose uncertainty
Concretize relationships
Formulate cause and effect
Determination of domain parameters
Multivariate explanation
Confirm hypothesis

Figure 5.2: The reference heuristic table [22] (part 2)

As explained before, there are two types of the users for the system and the system has two parts. The end-users and the developer. Therefore, there should be an evaluation for each of the system sections. As the result, I performed the evaluation from two perspective and based on two types of personas.

One of them evaluates the system from users perspective and the other one from the developer's perspective. The developer's evaluation is a heuristic evaluation based on the system requirements defined for the developer's part of system. The users' heuristic evaluation is related to the users' front-end. To evaluate whether the user interface works fine I performed a cognitive walk-through.

5.1.1 Developer's Heuristic Evaluation

I derived most of my developer's heuristics from the system requirements. Not all the reference table's heuristics were suitable for my thesis. As an example, the ones from 'Perceptual and Cognitive Heuristics' section are mainly concerned about the final visualisation and the user interface. However, my system is a prototype and my thesis focuses on the core functionalities and architecture of the system rather than the final visualisation and user interface. The reason for this is that we are exploring the requirements and possibilities in providing adaptive persona-driven visualisations, rather than proposing a final visualisation design. Additionally, although some of the other heuristics were related to my system but their meaning in their reference papers are different by how they were interpreted in my thesis. The reason for that is that my evaluation is of the entire prototype, and not just the final UI design. The heuristics are then useful for exploring other aspects of the architecture.

Match between system and the real world The definition of this heuristic in Nielsen's paper [46] is that the system should speak the users' language, with words, phrases and concepts familiar to the user, rather than

system-oriented terms [47]. However, in my system, the concern about matching between the system and the real world is mostly about data. Data should be relevant with the actual users' information. The earlier discussed problem about assigning different personas to the same user is an example of this problem. In that case because of the user's changes to his/her view, the user's view's updates are being assigned to a persona which is not related to the user any more and this causes inconsistency between the real world and the system. As the result the heuristic is addressing an issue of the system, of which the solutions were described earlier.

Consistency and standards In the Nielsen's paper [46], consistency means users should not have difficulty in understanding whether different words, situations or actions mean the same thing or not [47]. On the other hand, what my system concerns, as the system's consistency, is that the database transactions should change affected data only in allowed ways and the written data to the database must be valid according to all defined rules [3]. To prevent the inconsistency issue, I provided data checking before submitting it to the database. An example is the developer's personas' modification interface. In case of sending data to the database without checking it, there is the possibility of database error and system break down.

Error protection There are heuristics from the reference table [22] which do not belong to only end-users' component or the developer's component, but to the whole system. Error protection is about preventing a problem from happening in the first place [46]. Preventing errors from happening is an important factor in the system and I tried to handle the errors prevention by checking the format of data before submitting it in the system's database. Checking the entered usernames, passwords and other data formats to make sure that the entered data has a compatible

format with how it is defined in the dataset, is such an attempt.

Help users recognize, diagnose, and recover from errors Moreover, in case of detecting the error before letting it happen, the error can be announced to the users and guide them with its correction. I provided this option by announcing hints to users in case of detecting an error and ask them to re-enter the data considering the hints.

Figure 5.3 represents the heuristic metrics of the developer's evaluation. They are extracted from the system requirements.

Heuristic Number	Heuristic Name	Related Requirement
Heuristic One	System's Consistency	
Heuristic Two	Identifying Common Changes	Requirement 3: The system should react to identified patterns of initial views being modified in consistent ways.
Heuristic Three	System's Help and Documentation	
Heuristic Four	System's Control	
Heuristic Five	Usefulness of a Particular Persona	Requirement 5: The visualisation developer should be able to get reports of the manual changes made by the users to the initial seeded personas and views originally created.

Figure 5.3: The developer's heuristic table

System's Consistency is about being able to give response to concurrent users. It Refers to the requirement that any given database transaction must change affected data. This metric can be tested by posting the system on the server and get several users to connect to the system and make the

changes. A developer can also sign in to the system simultaneously and observe the changes.

Identifying Common Changes is one of the system's obligation which is about being able to record the changes to study the changes later and learn how to improve the initial map views.

System's Help and Documentation is a metric which is considered in every section of the system. Because, following a system and using it requires it to be either so simple and user-friendly to be used by itself, or provide documentations and guidance to get familiar with the system.

System's Control is about being able to control the system easily, since being able to perform changes or control a system via an interface and without needing to have direct access to the code behind or the database is a system advantage.

Usefulness of a Particular Persona is based on requirement number 6 which is about being able to get reports of the manual changes made by the users to the initially seeded personas. The reporting component and the results obtained from the reporting visualisations should make sense to the developer and s/he should be able to use them in order to understand the best changes s/he can make to the personas to make the initial views closer to users preferences.

5.1.2 Cognitive Walk-through

As mentioned many times before, the reference paper of my thesis is Shneiderman's Information Seeking Mantra paper [57].

In the field of information visualization, an important theoretical concept is the visual information seeking mantra described by Shneiderman [57] which offers two contributions for understanding information visualization methodology, the Visual Information Seeking Mantra and the

task-by-data-type taxonomy. The task-by-data-type taxonomy offers useful categorization of data types in the context of information visualization. The Mantra, Overview first, zoom and filter, then details-on-demand, describes how data should be presented on screen so that it is most effective for users [15].

So to do the heuristic design, I picked five of the heuristics which are related to the tasks I am using in my system. They are: overview, zooming, filtering, details-on-demand and history. Moreover, according to the system's requirements three more metrics are added.

Overview acts like a picture which shows the whole data entity the visualisation represents and provides a good context to understand the dataset. Data patterns usually can be seen from a view which includes and comprises the whole view [15]. Therefore, according to requirement 1 which talks about the process of the matching users' preferences and configuring the visualisation system's initial view, the view is considered as the overview task of the system. Providing users' preferred overview is considered a heuristic metric.

Zooming and Filtering can be regarded of navigation and change of representational advantage point. Using zooming and filtering, extra information will be removed and therefore, they both reduce the complexity of data representation [15].

'Zooming' is the adjustment of the size of data element on the screen, by user. By zooming-in, the size of smaller data elements increases and the size of larger data elements which are not needed reduces. 'Zooming-out' does the opposite thing [15].

Filtering does such a representation complexity reduction as well, but without changing user's view. Filtering allows the control of which data be visible. Filtering allows for adjustment of visible data points by users and thereby users can selectively hide or show the data they are interested

[15].

These two metrics are related to the system based on requirement 2 and 3 which are about the system support of view's manual configuration and modification of the initial views.

Filtering can also be referred to requirement 4 since using users' location to customise their initial view is also a way of filtering other information and locations which are not needed to be shown on the view.

Details-on-Demand provides the supplementary information. Usually, in an information visualisation, many points are visible in the overview and many of them are not because of limitation of the screen. The details-on-demand does this job on a point-by-point basis without requiring change of view. This can be useful for quickly identifying a specific data element among many by relating the detailed information to the rest of data. Providing these details using simple actions such as mouse-over allows to access of the information without needing the representational context to changes [15].

Details-on-demand refers to requirements 2 and 3 as well as zooming and filtering tasks.

History also refers to requirements 2 and 3, the same as previous customising tasks. Users should be able to easily return to the previous status when exploring the data. Comparing the current state of representation to the previous state can help in better understanding of the data. In addition, if the user makes a mistake, s/he should be able to easily recover from it [15].

As the result, five of my heuristics come from Sheidermans' paper [57]. And, by reviewing the table of figure 5.1 and 5.2, I selected some of the heuristics such as flexibility. Moreover, there are some characteristics which my system should have according to the system requirements. Therefore, I added some heuristics based on the system requirements to

the end-user section of the system heuristic evaluation, as well.

Flexibility refers to the availability of users to customise the interface in order to make it adaptable to their habits. Requirements 1, 2 and 3 are related to this metric. Requirement 1 requires the system to match the initial view automatically to users' preferences. This is act of flexibility of the user interface to users. Requirement 2 requires the system to support and record manual configurations of users which again refers to a flexible user interface. By providing this requirement, the user interface will be presented more flexible when user access their map view next times. Requirement 3 wants the system to react to identified patterns of initial views being modified in consistent ways; which is again about making the user interface flexible enough to record and affect the changes each time users make to their views.

System's Help and Documentation as discussed before, is a metric which is considered in every section of the system. Because, following a system and using it requires it to be either so simple and user-friendly to be used by itself, or provide documentations and guidance to get familiar with the system.

Responsive User Interface According to requirement seven of the system, the user interface should be responsive to be able to fit on any browser.

Figures 5.4 and 5.5 shows the discussed heuristics of the end-user section of the system.

Heuristic Number	Heuristic Name	Related Requirement
Heuristic Six	Flexibility	<p>Requirement 1: The process of matching users' preferences and configuring the visualisation system's initial view should be automatic, based on initial seeded personas created by the developer.</p> <p>Requirement 2: The system should support and record subsequent manual configuration by users.</p> <p>Requirement 3: The system should react to identified patterns of initial views being modified in consistent ways.</p>
Heuristic Seven	Overview	<p>Requirement 1: The process of matching users' preferences and configuring the visualisation system's initial view should be automatic, based on initial seeded personas created by the developer.</p>
Heuristic Eight	Zooming	<p>Requirement 2: The system should support and record subsequent manual configuration by users.</p> <p>Requirement 3: The system should react to identified patterns of initial views being modified in consistent ways.</p>
Heuristic Nine	Filtering	<p>Requirement 4: The system should support using location information to inform the selection and scope of an initial view.</p> <p>This can be referred as a filtering option.</p>

Figure 5.4: The end-users' heuristic table (part 1)

Heuristic Ten	Details-on-Demand	
Heuristic Eleven	History	<p>Requirement 2: The system should support and record subsequent manual configuration by users.</p> <p>Requirement 3: The system should react to identified patterns of initial views being modified in consistent ways.</p>
Heuristic Twelve	System's Help and Documentation	
Heuristic Thirteen	Responsive UI	Requirement 6: The user interface should be compatible with different screen sizes.

Figure 5.5: The end-users' heuristic table (part 2)

5.2 Process and Results

Now that evaluation metrics are discussed, the evaluation processes will be discussed in continues. I performed each evaluation in two steps. As

the first step, I did the cognitive walk-through and the developer's heuristic evaluation by myself and got some useful results. According to those results I made some changes to the system and database architecture which will be discussed in advance. As the second step, two visualisation experts performed the evaluations and their results are presented and discussed as well, continuously.

5.2.1 Cognitive Walk-through

The cognitive walk-through is a usability evaluation method in which one or more evaluators work through a series of tasks and ask a set of questions from the perspective of the user [1].

In cognitive walk-through, the developers and designers perform the test themselves instead of the users. So, I designed several actions as the cognitive walk-through to be performed and at last, they check each heuristic from the Front-end users' evaluation table. A cognitive walk-through is based on the assumption that a user will explore the interface to learn how to use it.

End-users' Persona's Task In this task the user performs the below actions.

Action 1 Signing up into the system.

- **Step1** When the browser opens, the user clicks the sign up button to be redirected to the sign up page.
- **Step2** The user selects the options to declare his/her preferred persona from the drop down menus and fills the username and password input textboxes. To test the system's sign up obligations, the user can leave the required inputs empty and get the system warning.
- **Step3** After signing up, the user redirects to his/her map view page.

Action 2 Changing the map view to his/her more favourite view.

- **Step1** When entering to the map view page, the user starts playing with his/her map view by changing filtering information and level of zooming till make the view as close as possible to his/her favourite view.
- **Step2** After changing the map view, the user logs out of the system.

Action 3 Logging in to the map view.

- **Step1** The user enters his/her username and password to sign in to the system. In case of wrong username or password the system gives a warning. When signing in to the map view, the view should be presented as the last time the user signed out.

After taking the first phase of cognitive walk-through, these are the results of the test:

Flexibility My system presents the map view based on the information got from users at the beginning. Although, the presented view may not be the preferred view to a user and is far from what s/he wants, the changes s/he makes to the view will be recorded and the next view which will be presented to s/he next times, is the changed view.

Overview The system represented the map's overview based on the chosen options of the sign-up page's questionnaire.

Zoom The zooming function worked well.

Filter The map's information could be changed and filtered using provided checkboxes.

Details on demand The place's details such as, their addresses and the opening hours, were available by the hovering interaction.

History When signing up the next time the map's view was represented including the changes the user made to his/her view last time, before s/he closes the system. Therefore, the history function works fine.

Needing help and documentation My system's audience are ordinary people with different level of education so the front-end is designed so simple to be followed.

Responsive user interface My user interface is responsive using bootstrap library.

5.2.2 Developer's Heuristic Evaluation

According to the discussed heuristic metrics, the below discussions of the evaluation represent the changes which were needed to be performed to the system as the result of first step.

Consistency There will not be a problem in giving respond to concurrent users since the database is in its most simple design and there are not inter connections between tables which usually creates database errors. On the other hand, one problem detected is that each time the developer has already signed in and users make updates to the database, the reporting component does not show the real-time updates, but just the ones which were made before the developer signed into the system. The reason is each time the developer signs in, the system connects to the database and imports data to the reporting page. This is not a problem in this stage, since the developer is seeking the changes on a specific time periods and does not need to monitor changes online and always. But real-time reporting component can be an improvement which can be added in future works.

Identifying common changes When trying to mark this metric as a satisfying or dissatisfying metric, I realized in identify the improvements, not only the changes of users should be considered, but also thee more important aspect is the changes which are assigned to a specific persona. Because, the initial views are designed based on the persona's characteristics, therefore to improve the initial views, the personas need to be studied. As the result, the reporting component should be based on the changes per personas. So, I added the assigned personas to each of the users' record in the database and added the reporting visualisations which represent the changes per personas.

Needing help and documentation The developer's interface is simple to be used. There is just e few elements on the page and the explanations are provided to guide the developer with the provided visualisations.

System Control My system has the advantage of changing the sign-up questions through a text file without needing to have access to the code behind. Also, the developer interface provides the possibility of performing changes to the database without having to directly access the database.

Determination of usefulness of a particular persona From a developer perspective, I was able to understand which persona requires changes and what filtering objects and zooming values are more preferred among the users assigned to that persona, using the visualisation which are made based on the data-history table of the database.

To summarise, by performing the fist phase of the developer's heuristic evaluation, I understood the personas assigned to the users should be recorded in the database in each of the users' records to be used later in the reporting component. Therefore, I included the visualisation's of the users' views' updates based on their chosen personas in the reporting component to be able to track the needed changes of the designed per-

sonas and their designed views.

By performing the second phase of the developer's evaluation by my supervisor, there were some changes mostly to the developer's interface. One of the most important changes was deleting the data-history table of the pages and building a search page. In the search page, the developer can search in the data-history table. The searching option is not restricted to a specific data or a specific column and the column and the data can be chosen in the searching interface which was represented in chapter 4. Therefore, including the whole data-history table in the interfaces is not needed any more, since searching component has made the possibility of accessing the data, for the developer. Moreover, including the data-history tables when the number of users is a great amount is not a good approach since it can not fit into the developer's interface page nicely. Therefore, I added the searching component and also a navigation bar in order to be able to easily move between the reporting component's pages.

The filtering and zooming pages are also included in the navigation bar as well as every single page of the reporting component. This has made the access to different pages of the reporting component easier for the developer. Moreover, the navigation bar changed the pages to nicer and cleaner pages.

The other change was about changing the navigation bar from a vertical bar to a horizontal bar after discussing with my supervisor, I realized changing it to a horizontal navigation bar is a better decision, since the size of the charts can easily increase by the increase of number of users.

5.3 Changes

My thesis aims to explore whether and how the users' interactions in the first four steps can be used to create the appropriate early experience for future users with the same goals or characteristics. To investigate the answer to this question, I built a prototype to explore approaches to do such

a task. Although the prototype has faced limitations, during the process of implementation and evaluation, I learnt how changes can be applied to the prototype to improve it to a more complete system.

Self-adapting Not only Submitting the changes of the system, but also analysing them can be done both manually and automatically. Each has got some advantages and limitations. Manually analysing and changing the system, can be more trustable when the number of users are limited; on the other hand, in a systems with lots of users, this approach is time consuming and prevents the system to be updated frequently. This results in affecting the further analysis of the system and may distract the developer to make the best decisions for the system, since s/he is not aware of the missed updates.

Persona's update In continuous of the previous point, when a user has made lots of changes to his/her view in a way that the view is not based on his/her assigned persona's characteristics any more. Therefore, the assigned persona needs changing based on the updates s/he has done to his/her view. This causes the problem of inconsistency of the system, since the system's data is not compatible with the real data. When facing this problem, there were two approaches, self-adapting and manually adapting which were explained in chapter 3. When implementing the manually adapting in the prototype, the limitations were not detectable because of the small number of end-users, but later when discussing it as a standard system with a huge number of users, I faced the problem of great time cost. Moreover, because of the amount of time the developer needs to consume on analysis and updates, there may be other changes which will affect the previous updates and this results in less accurate system. I learnt transferring the system to a self-adapting system is a top priority action which should be considered in system improvement.

Using time slider During the project process, I learnt one of the improvements for the system can be not only customising the initial views of the users' visualisations, but also customising the analysis of those visualisations. Customising in visualisations' analysis is like classifying the analysis factors of the visualisations. Using time periods in visualisations' analysis is one of the things which can make the analysis results more accurate. As an example it can be analysed to check how will the users change their views at the beginning when they enter to their maps' accounts and how will the updates change at the end before signing out of the system. This helps in better realise how to change users' view to make it more preferable. As an example, if the user changes the zooming level at the beginning but returns to the default zooming value again, and this pattern is being repeated by so many other users of the same persona, this can be resulted that although zooming level may not seem suitable at the beginning but finally it is realised as the best zooming level by the users. And, unlike all the updates assigned to the zooming level, it does not need changing. To do this, time column is provided in the database. The developer can use a slider to filter the updates based on specific time periods and represent the personas' analysis visualisations based on a specific time period. This is one of the system's improvements which can result in better analysis of the designed personas and their views.

Real-time system An important improvement of the system is about changing it to a real-time system. My system is a pull system which means to import updates to the system, the developer needs to pull them in the system using approaches such as refreshing the pages. But, in a push system, the updates are imported to the system automatically and the system gets updated by itself. This is an important factor in the systems since by converting a system to a real-time system, the analysis updates are more trustable. In a non real-time system the developer needs to be worried about updating the data each time and this is a limitation of the system.

Therefore, one of the important improvements of the system is changing the system to a real-time system.

Automatic customisation In the signing-up process, the cost of time (in case of so many questions in the sign-up questionnaire) can be so much. On the other hand, I learnt every approach has got its limitations and advantages. Representing a common map view to all the users and redirecting them to the map view without asking them questions to customise the map view is another approach. In this approach the initial view is not customised and can get customised later by the users. But sometimes the cost of each approach needs to be calculated and the one with the least limitations and the best fit one should be chosen. Comparing the time cost of asking questions from users when signing up, with the act of not asking questions and presenting a non-customised initial view, the time cost is more suitable for the project, since initial view customisation has the greater priority.

There was a minor change in database, since during the process of evaluating the system, I figured out that personas should be used in not only designing views, but also in the database, directly. Before evaluating the system, I used personas to just designing the views for the users and I did not affect them directly in the system by including the type of personas, users are assigned to, in the database. Therefore, I was not able to figure out if a persona is designed preferable enough for the users or not. By including the designed personas in the database and the reporting visualisations, it got easier to detect the design changes which each persona and its map's view needs.

The prototype helped me in answering the thesis question of whether and how the users' interactions in the first four steps of Shneiderman [57] can be used to create the appropriate early experience for future users. However, the process of finding the answer to the question has raised other questions which will be discussed in chapter 6.

Chapter 6

Conclusion

Once the evaluation is completed it was possible to determine the issues with the design and implementation of the system according to requirements and goals from section 2.3.

Using the cognitive walk-through helped in validating if the persona scenarios were possible and what are the system's limitations which can be considered in future works of the project. Using the evaluation, the issues of the prototype were determined. The prototype is a proof-of-concept of the design to explore the issues and identify ways to improve the system

As stated in chapter one, my research question is about exploring whether and how the users' interactions in the first four steps of Shneiderman [57] can be used to create the appropriate early experience for future users with the same goals or characteristics. I attempted to design solution and implemented a proof of concept prototype. According to the evaluations, I believe the users' interaction in the Shneiderman's tasks [57] can be used in creating the suitable future initial views for users with same preferences. Using the idea is possible and helpful in improvement of the visualisation's systems' initial view. However, I faced limitations during the process of the project. I was able to find solutions for some of them and I could not find solutions to some of them because of lack of time. Even the proposed solutions may not be the best ones and by putting more time

and effort, the better solutions may appear.

By using the performance metrics defined in evaluation chapter it was possible to show that almost all of the defined requirements had been met by the project. Although, there are some major limitations with the system which will be discussed as the future works of the project.

6.1 Summary

Requirement 1 The sign-up process of the users' system and assigning the user to a designed persona based on his/her answers and presenting the persona's view to him/her is an automatic process. Therefore, using the sign-up component, the process of matching users' preferences and configuring the visualisation system's initial view can be done automatically, based on initial seeded personas created by the developer.

Requirement 2 The system represents and saves the changes users make to their views in the database via index page and database components. Therefore, The system supports and records subsequent manual configuration by users

Requirement 3 The system saves the users' views' changes to their assigned personas and represents them in their views for next times of their login. Therefore, The system reacts to identified patterns of initial views being modified manually.

Requirement 4 The users' maps' views are loaded based on their current location. Therefore, the system supports using location information to inform the selection and scope of an initial view.

Requirement 5 The visualisation developer is able to get reports of the manual changes made by the users to the initial seeded personas and

views originally created through reporting component. S/He is also able to make the changes to the initial seeded personas and views using the developer's profile.

The developer is able to get reports of the manual changes made by the users to the initial seeded personas and views originally created via reporting component and is able to make the changes via the developer interface.

Requirement 6 The system's pages are compatible with different screen sizes.

6.2 Future Questions

By answering the main question of thesis which was discussed earlier this chapter, a set of other questions are discovered. These questions are risen over time during the project design, implementation and evaluation. My main goal was answering the main question of the thesis rather than trying to answer everything by myself.

Every approach has got some limitations and advantages. The signing-up process has cost of times which increases by increasing of the sign-up questions. Eliminating the sign-up questionnaire is another approach which has the disadvantage of presenting non-customised view to users. The new question is about which approach fits better to the system. Deciding about approaches needs calculating the weight of advantages over disadvantages. The approach which provides the project and its users' goals better, is the better approach. Therefore, one of the risen questions in continuous of the thesis is about whether asking the users to answer sign-up questions or presenting a common view to users and customise it over time? In other words, should we prefer the time cost over initial customisation or not?

The other discovered questions after answering the main thesis questions, to improve the system, are:

1. What approaches can be used to change the system to a real-time system?
2. What artificial intelligent approaches can be used to make the process of reacting to identified patterns of initial views being modified in consistent ways, automatically? (directly according to requirement 3)
3. How can other Shneiderman's tasks [57] affect the system improvement?

These questions can help in planning for the future works of the project. They are derived during the prototype's implementation and evaluation process. During the process of evaluation, the limitations of converting the prototype to a complete system were cleared. Changing the prototype to a real-time system is an approach to solve some of the limitations. When the prototype converts to a real system, the number of users increase. A pull system in which the developer needs to pull updates from the database into the system by himself, is hard to be controlled and the developer's analysis are less reliable since the analysis may be based on the data without updates. Therefore, question one is one of the important questions in case of converting the prototype to a full system. Moreover, one further improvement of the prototype can be converting the system to a more self-adapting system. Again, during the evaluation process with several users, the act of updating personas' characteristics manually by the developer was not a major limitation, but in the full system when the number of personas and users increase, this approach is not suitable any more. Using artificial intelligent approaches to change the process of reacting to identified patterns of initial views being modified to an automatic process, can be so helpful. The last question is about improving the prototype to a more professional system. In my thesis just four of the Shneiderman's tasks [57] were involved due to the limited time. However, more Shneiderman's tasks [57] can be used in future works of the project to make the system

more complete.

6.3 Future of the Project

Both the evaluation and the things I learnt during the process of the project, uncovered a number of limitations that will need to be solved to improve the system to a better system.

Self-adaption Comparing the cost of customising the system manually by the developer and self-adapting approach, when the number of users is a great amount, self-adapting approach becomes a future plan for the project which can improve the system to a better one.

Time periods As discussed before, inserting time periods into the developer's reporting component's visualisations, can help him/her to better analyse the end-users' views' updates and focus on a specific time period. Inserting time periods is one of the further actions to improve the project.

Real-time system As one of the further improvements of the system, is changing the system to a real-time system. Since, my system now is a prototype not a complete system and the number of end-users was a small number to just test the requirements of the system, it was a pull system rather than a push system. But, as a future work of the project changing the system to a real-time system is one of the most important improvements which should be done as discussed before.

6.4 Contributions

The completion of the adaptive initial users' overview of a map visualisation system resulted in three contributions:

1. An exploration of the issues involved in matching users' information to the initial presentation of visualisations, relating to the first stage in Shneiderman's Information Seeking Mantra [57].
2. Proof-of-concept prototype of the design of an adaptive visualisation system.
3. Evaluation of the users' data factor by using heuristic evaluation and cognitive walk through.

Bibliography

- [1] Cognitive Walkthrough. <http://www.usabilitybok.org/cognitive-walkthrough>, 2010. [Online; accessed 19-January-2017].
- [2] Google Analytics. <http://searchbusinessanalytics.techtarget.com/definition/Google-Analytics>, 2011. [Online; accessed 19-January-2017].
- [3] Consistency. <https://www.techopedia.com/definition/27386/consistency-databases>, 2017. [Online; accessed 19-January-2017].
- [4] AGAFONKIN, V. Leaflet. <http://leafletjs.com/>, 2015. [Online; accessed 22-July-2016].
- [5] AMBLER, S. W. User Interface Flow Diagrams (UI Storyboards): An Agile Introduction. <http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm>, 2014. [Online; accessed 17-January-2017].
- [6] APIs, G. M. Events. <https://developers.google.com/maps/documentation/javascript/events>, 2016. [Online; accessed 04-January-2017].

- [7] APIs, G. M. Listening to DOM events. <https://developers.google.com/maps/documentation/javascript/examples/event-domListener>, 2016. [Online; accessed 04-January-2017].
- [8] APIs, G. M. Style Reference. <https://developers.google.com/maps/documentation/javascript/style-reference>, 2016. [Online; accessed 04-January-2017].
- [9] BLOMKVIST, S. Persona an overview. *Department of Information Technology, Human-Computer Interaction, Uppsala University, Sweden*, 2002.
- [10] BOOTSTRAP. Bootstrap. <http://getbootstrap.com/>, 2017. [Online; accessed 04-January-2017].
- [11] CARENINI, G., CONATI, C., HOQUE, E., STEICHEN, B., TOKER, D., AND ENNS, J. Highlighting interventions and user differences: Informing adaptive information visualization support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2014), CHI '14, ACM, pp. 1835–1844.
- [12] CASNER, S. M. Task-analytic approach to the automated design of graphic presentations. *ACM Trans. Graph.* 10, 2 (Apr. 1991), 111–151.
- [13] COCKBURN, A., KARLSON, A., AND BEDERSON, B. B. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.* 41, 1 (Jan. 2009), 2:1–2:31.
- [14] CODEGURU. Serialization/Deserialization in .NET. http://www.codeguru.com/csharp/csharp/cs_syntax/serialization/article.php/c5821/SerializationDeserialization-in-NET.htm, 2003. [Online; accessed 04-January-2017].
- [15] CRAFT, B., AND CAIRNS, P. Beyond guidelines: what can we learn from the visual information seeking mantra? In *Ninth International Conference on Information Visualisation (IV'05)* (July 2005), pp. 110–118.

- [16] CREATIVE, B. Is Leaflet a better tool than Google Maps? <http://www.creativebloq.com/web-design/leaflet-google-maps-121413738>, 2014. [Online; accessed 22-July-2016].
- [17] DESAI, A., MIAN, M., HAZEL, D., TEREDESAI, A., AND BENNER, G. Data visualisation in educational datasets using a rule-based inference system. *IEEE International Congress on Big Data*, 2014, pp. 462–469.
- [18] DESAI, A., MIAN, M., HAZEL, D., TEREDESAI, A., AND BENNER, G. Data visualization in educational datasets using a rule-based inference system. In *2014 IEEE International Congress on Big Data* (June 2014), pp. 462–469.
- [19] DOCKER. What is Docker? <https://www.docker.com/what-docker>, 2016. [Online; accessed 04-January-2017].
- [20] DOMIK, G. O., AND GUTKAUF, B. User modeling for adaptive visualization systems. In *Proceedings of the Conference on Visualization '94* (Los Alamitos, CA, USA, 1994), VIS '94, IEEE Computer Society Press, pp. 217–223.
- [21] ERIKSSON, U. ReQtest. <http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>, 2012. [Online; accessed 21-July-2016].
- [22] FORSELL, C., AND JOHANSSON, J. An heuristic set for evaluation in information visualization. In *Proceedings of the International Conference on Advanced Visual Interfaces* (New York, NY, USA, 2010), AVI '10, ACM, pp. 199–206.
- [23] FRÍAS-MARTÍNEZ, E., CEBRIÁN, M., PASCUAL, J. M., AND OLIVER, N. Explicit vs. implicit tagging for user modeling. In *Proceedings*

- of the Workshop on Personalization in Mobile and Pervasive Computing, PMPC@UMAP 2009, Trento, Italy, June 22, 2009.* (2009).
- [24] GARGAN, JR., R. A., SULLIVAN, J. W., AND TYLER, S. W. Multi-modal response planning: An adaptive rule based approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1988), CHI '88, ACM, pp. 229–234.
- [25] GOLEMATI, M., HALATSIS, C., VASSILAKIS, C., KATIFORI, A., AND O. PELOPONNESE, U. A context-based adaptive visualization environment. In *Tenth International Conference on Information Visualisation (IV'06)* (July 2006), pp. 62–67.
- [26] GOOGLE, D. Google Maps API. <https://developers.google.com/maps/faq#whatis>, 2016. [Online; accessed 22-July-2016].
- [27] GOOGLE, D. Google Maps API. <https://developers.google.com/maps/documentation/javascript/tutorials/>, 2016. [Online; accessed 22-July-2016].
- [28] GOTZ, D., AND WEN, Z. Behavior-driven visualization recommendation. In *Proceedings of the 14th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2009), IUI '09, ACM, pp. 315–324.
- [29] HOWARD BUTLER, MARTIN DALY, A. D. S. G. T. S. C. S. The GeoJSON Format Specification. <http://geojson.org/geojson-spec.html#definitions>, 2008. [Online; accessed 22-July-2016].
- [30] HU, Z., NARAYANASWAMY, M., RAVIKUMAR, K., VIJAY-SHANKER, K., AND WU, C. Literature mining and database annotation of protein phosphorylation using a rule-based system. Oxford Journals, 2005.

- [31] INC, Y. Sample Software Architecture Document. <http://cs.uns.edu.ar/~mc/disenio/downloads/Otro%20Material/SAD-OnlineCateringService.pdf>, 2016. [Online; accessed 11-August-2016].
- [32] IONESCU, D. 10 cool and useful Google Maps mashups. <http://www.pcworld.idg.com.au/slideshow/419990/pictures-10-cool-useful-google-maps-mashups/>, 2016. [Online; accessed 22-July-2016].
- [33] JQUERY. jQuery.ajax(). <http://api.jquery.com/jquery.ajax/>, 2017. [Online; accessed 04-January-2017].
- [34] JQUERY, T. F. What is jQuery? <https://jquery.com/>, 2016. [Online; accessed 22-July-2016].
- [35] JUVVA, K. Real-Time Systems. https://users.ece.cmu.edu/~koopman/des_s99/real_time/, 1998. [Online; accessed 19-January-2017].
- [36] MACWRIGHT, T. From Google Maps to Open Source With MapBox and Leaflet. <https://www.mapbox.com/blog/open-source-with-leaflet-and-mapbox/>, 2012. [Online; accessed 22-July-2016].
- [37] MICROSOFT. ASP.NET Overview. <https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx>, 2016. [Online; accessed 22-July-2016].
- [38] MICROSOFT. Adding Data to a DataTable. [https://msdn.microsoft.com/en-us/library/z16c79x4\(v=vs.110\).asp](https://msdn.microsoft.com/en-us/library/z16c79x4(v=vs.110).asp), 2017. [Online; accessed 04-January-2017].
- [39] MICROSOFT. ASP.NET Session State Overview. <https://msdn.microsoft.com/en-us/library/ms178581.aspx>, 2017. [Online; accessed 04-January-2017].

- [40] MICROSOFT. How to: Pass Values Between ASP.NET Web Pages. [https://msdn.microsoft.com/en-us/library/6c3yckfw\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/6c3yckfw(v=vs.85).aspx), 2017. [Online; accessed 04-January-2017].
- [41] MICROSOFT. RegularExpressionValidator Class. [https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.regularexpressionvalidator\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.regularexpressionvalidator(v=vs.110).aspx), 2017. [Online; accessed 04-January-2017].
- [42] MICROSOFT. Serialization in the .NET Framework. [https://msdn.microsoft.com/en-us/library/7ay27kt9\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/7ay27kt9(v=vs.110).aspx), 2017. [Online; accessed 04-January-2017].
- [43] MOZILLA, D. N. JavaScript. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>, 2016. [Online; accessed 22-July-2016].
- [44] NAZEMI, K., STAB, C., AND FELLNER, D. W. *Interaction Analysis for Adaptive User Interfaces*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 362–371.
- [45] NAZEMI, K., STAB, C., AND KUIJPER, A. A reference model for adaptive visualization systems. In *Proceedings of the 14th International Conference on Human-computer Interaction: Design and Development Approaches - Volume Part I* (Berlin, Heidelberg, 2011), HCII'11, Springer-Verlag, pp. 480–489.
- [46] NIELSEN, J. Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems* (New York, NY, USA, 1994), CHI '94, ACM, pp. 413–414.
- [47] NIELSEN, J. 10 Usability Heuristics for User Interface Design. <https://www.nngroup.com/articles/>

- ten-usability-heuristics/, 1995. [Online; accessed 19-January-2017].
- [48] NULLSKULL.COM. ASP.NET - What are the advantages and drawbacks of using ADO.NET? <http://www.nullskull.com/q/10292797/what-are-the-advantages-and-drawbacks-of-using-adonet.aspx>, 2011. [Online; accessed 04-January-2017].
- [49] OPENSTREETMAP. OpenStreetMap Foundation. https://wiki.osmfoundation.org/wiki/Main_Page, 2016. [Online; accessed 22-July-2016].
- [50] OSTROW, A. 13 Must-See Google Maps Mashups. http://mashable.com/2007/07/10/google-maps-mashups-2/#m6GlM5I_VZq4, 2007. [Online; accessed 22-July-2016].
- [51] PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*, 5th ed. McGraw-Hill Higher Education, 2001.
- [52] PROGRAMMERS. Mapping between 4+1 architectural view model UML. <http://programmers.stackexchange.com/questions/233257/mapping-between-41-architectural-view-model-uml>, 2014. [Online; accessed 11-August-2016].
- [53] QUINSTREET. What is MySQL? <http://www.webopedia.com/TERM/M/MySQL.html>, 2016. [Online; accessed 22-July-2016].
- [54] ROBINLOVELACE.NET. Testing web map APIs Google vs OpenLayers vs Leaflet. <https://news.ycombinator.com/item?id=7451375>, n.d. [Online; accessed 22-July-2016].
- [55] ROUSE, M. What is a mash-up? <http://whatis.techtarget.com/definition/mash-up>, 2016. [Online; accessed 22-July-2016].

- [56] SAMKNOWS. The global platform for internet measurement. <https://www.samknows.com/>, 2016. [Online; accessed 22-July-2016].
- [57] SHNEIDERMAN, B. The eyes have it: A task by data type taxonomy for information visualisation. Department of Computer, Science Human-Computer Interaction Laboratory and Institute for System Research, University of Maryland, 1996, pp. 336–343.
- [58] STACKOVERFLOW. What is meant by 'first class object'? <http://stackoverflow.com/questions/705173/what-is-meant-by-first-class-object>, 2009. [Online; accessed 22-July-2016].
- [59] STACKOVERFLOW. How to log all changes in a mysql table to a second one? <http://stackoverflow.com/questions/6787794/how-to-log-all-changes-in-a-mysql-table-to-a-second-one>, 2011. [Online; accessed 26-September-2016].
- [60] TECHOPEDIA. Front-End System. <https://www.techopedia.com/definition/3799/front-end-system>, 2016. [Online; accessed 22-July-2016].
- [61] VAN TONDER, B., AND WESSON, J. *Design and Evaluation of an Adaptive Mobile Map-Based Visualisation System*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 839–852.
- [62] W3SCHOOLS.COM. ASP QueryString Collection. http://www.w3schools.com/asp/coll_querystring.asp, 2017. [Online; accessed 04-January-2017].
- [63] WEBOPEDIA. rule-based system. http://www.webopedia.com/TERM/R/rule_based_system.html, 2017. [Online; accessed 16-January-2017].