University of Glasgow | School of Computing Science

Urban Data Timeline

Yordan Yordanov

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 28, 2016

Abstract

We show how to produce a level 4 project report using latex and pdflatex using the style file l4proj.cls

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name:		Signature:	

Contents

Chapter 1

Introduction

1.1 Motivations

Smart cities bring a lot of advantages. They can opt for better urban planning and development by making more efficient use of the infrastructure to improve productivity and services but also to reduce the waste of fuel and energy. Furthermore their intelligence can change and enhance the way authorities respond to changing circumstances. On the other hand, achieving all these goals is a challenging process. Huge amounts of information, created by social feeds, environmental sensors, traffic, news and many more, have to be gathered, stored and analysed. This project aims to combine and visualise all these data strands so that some can look at the cities from a different angle and possibly extract new tendencies and patterns, not possible to discover by following only a single source. This will help us identify problems or challenges that we had not been aware of and provide effective solutions that we all as citizens can only benefit from.

1.2 Aims

This project aims to build a tool for the fusion and visualisation of timely data collected from the Urban Big Data Centre for their Integrated Multimedia City Data project. Past data from various urban data streams, such as social media posts, news, blogs, traffic information, environmental sensors and many more has been provided. The tool should be able to query all this data and come up with a timeline representation of observations that might be of interest to the user. This application is initially build to be used by the general public but may as all be valuable to local researchers who follow or compare peoples opinions, businessmen who want to know how their business is developing, media that want to report what are the public impressions about an important event and even politicians while running their campaigns.

1.3 Background

Smart cities today represent a perception to integrate both information and communication technologies in an acceptable and maintainable fashion to manage their infrastructure and facilities. The optimal goal is to enhance the quality of life and satisfy the residents needs. ICT gives to the people, who maintain these cities, the great opportunity to directly follow what is happening and take actions accordingly. This can be done only by processing the huge amounts of information, collected from sensors, in real time and providing knowledge and information: the keys for discovering inefficiencies.

However, the biggest asset of a smart city is not the number of cameras or sensors but the people that occupy it. A third of the internet users[9] represent their smart-phone as the most important device for going online. A third of the population owns a smart device and use it every day to bank, shop and access social media. Social media streams, such as Twitter, have a reputation to be extremely useful source of information. Anybody can understand what is happening all around the world in real time. With that comes various opportunities for developers to implement systems that automatically detect and track events as they happen. Smart cities can benefit from that as what better way to improve efficiency than for example identifying and reporting road accidents to the emergency services in real time.

Aiming to help with analysing these massive amounts of data, the Urban Big Data Centre was established by the UK Economic and Social Research Council to address social, economic and environmental challenges facing cities. Their researchers are undertaking innovative projects, covering topics from big data management to linking and analysing the multi-structural urban data. Such project is the Integrated Multimedia City Data.

1.3.1 Integrated Multimedia City Data

Integrated Multimedia City Data[4] (iMCD) is one of the Urban Big Data Centres inaugural projects, funded by the Economic and Social Research Council. It is designed to provide the UBDC with innovative primary data sources. The project consists of four strands: representative household survey, tracking of real-time urban sensors, internet based visual and textual media collection.

The core research strand is the representative household survey that aims to gather data about peoples attitude and behaviour when it comes to information and communication technologies, travelling and learning. Some of the participants are given GPS and life logging sensors that record their activities and travel. Meanwhile the visual and textual data, referring to Glasgow and surrounding areas, is to be collected from the internet.

All of these strands, together, are able to show how Glasgow performs as a smart city. The Terrier IR team provides various data web services so that all these sources can be browsed and queried.

1.3.2 Terrier IR

Terrier IR[1] team consists of researchers, working on large-scale textual information retrieval. They are part of the Computer Science Department at the University of Glasgow. The Terrier IR Platform[2] is where they concentrate their research. It is an open source search engine that is highly effective and ready to be used on large-scale collections of documents.

1.3.3 RESTful Web Services

Web technologies are not meant to only deliver HTML pages between HTTP clients but also, with their technical fundamentals of URIs, HTML and HTTP, to provide a widely deployed information delivery and service platform. REST, on the other hand, is a set of constraints that guide the design of such systems. The general

[1]`http://terrierteam.dcs.gla.ac.uk/`
[2]`http://terrier.org/`

claim of RESTful systems, implementing these constraints, are that they are highly scalable and that the interlinking of self-describing representation formats allows such a system to grow organically and in a decentralized way[6]. As the time of writing, REST is one of the most important technologies that is used in many Web and Mobile applications.

1.3.4 Web Application

Web application runs in a web browser and follows the client-server model(1.1). It is based on splitting the tasks or workloads between a provider, called server, and a requester, called client. Web applications can run on every platform as far as there is a web browser. This is a great relief for the developers as there are far less constraints on the implementation. Web applications usually are built from a combination of server-side and client-side scripts. The server-side scripts take care of storing, processing and retrieving information while the client-side is responsible for presenting it. Furthermore, the separation of concerns allows for updates to be made independently.

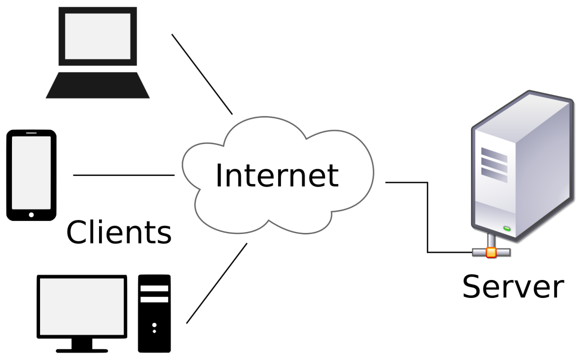


Figure 1.1: Client-Server Model

1.4 Outline

The rest of the report is structured as follows:

- **Chapter 2** provides an overview of related work with examples of similar applications.
- **Chapter 3** discusses the project planning by separating it into project approach and requirements engineering.
- **Chapter 4** explains the design by showing the initial paper prototypes and architectural diagrams. It also provides a comparison between the possible technologies, that are available and suitable for such a project.
- **Chapter 5** goes through the actual implementation of each component and describes the challenges faced.
- **Chapter 6** describes the two phases of the evaluation process and explains the testing strategy for the components.
- **Chapter 7** concludes the report and discusses the implication of the evaluations in terms of future directions

Chapter 2

Related Work

There are applications that have been developed to use a timeline for presenting some kind of data. This chapter is reveals a short description of related products, along with key differences and correlations in comparison with the main ideas behind this project.

Google Maps Timeline

Google stores a history of where anybody that uses its location services goes. All this data is collected by your device sensors and the navigation you use. Then it is visualized with the Google Maps Timeline [13] feature. This application is advertised to be an easy way to view and remember places somebody has been on a given day at a given time. Without any input, the timeline shows predictions of when you have arrived or left a place. One of the key features is highlighting when you have visited the most places and how you had travelled. The application does not offer sharing as Google wants your information to remain private. As far as human interaction goes, you are allowed to correct, confirm or delete a place where Google thinks you had been but even after deleting, it can still be seen that you had passed by that area. In 2009, the company released a similar feature called Google Latitude that offered sharing of location history but the project was closed down [10].

This product has the same initial goal to visualize timely data originating from different streams like navigation history, pictures, travel and walking routes. Some of the key features are using a vertical timeline to display the events, allowing searching based on a specific day, month, year and having a way to show (via bar chart) how active you have been every day in the past few weeks.

Social Media Timeline

All social media websites in the likes of Twitter and Facebook use some sort of a timeline to visualize their user's personal information. However they differ on the way they organize their posts. Facebook describes a timeline to be a place on your profile where you can see your own posts, your friends' activities and stories you're tagged in, sorted by the date and time they were posted. On the other hand, Twitter displays a stream of tweets from accounts that you have chosen to follow. Making use of machine learning algorithms, posts that you are likely to care about more are displayed first.

Despite the fact that social media applications have as a main goal the delivery of a secure and reliable tool for communication, they also provide their users with the ability to visualize their personal data streams like photos, events, group activities and accomplishments. Both Facebook and Twitter use a vertical timeline and allow for searching based on keywords. One of the key features is infinite scrolling that make the illusion of one endless stream of events by making use of the million users, posting every day.

Tech City Map

Tech City Map pulls streams of social media posts for all the businesses in the East London area and tries to analyse their influence. The idea lies on using a map to display all the corporations in the area and linking them together by using different coloured lines for any tweet from one business to the other. The tool allows searching for a specific company as some points represents more then one corporation and manually searching though the map can be difficult. From 2010 to 2012[1] the number of businesses in East London had risen from 200 to 600. This growth was welcomed even by the Prime Minister at that time, David Cameron, proving that such tools are even followed by politicians.

Conclusion

Some of the previously described applications are proven to be successful by millions of users all around the world. However, they focus only on their user and do not really visualise and offer a big variety of data sources. On the other hand, Urban Data Timeline targets the smart city by increasing the user's understating about it. Furthermore, with the help of the Urban Big Data Centre, it has plenty data streams to offer: querying a tweet collection based on a hashtag or specific tweet terms, examining the events, detected by automated systems, related to specific terms, obtaining information about busy venues in an area of a city, finding train stations within a radius of a specific location, searching for delayed trains and accessing past weather data records. Also, all these big companies like Google, Facebook and Twitter provide their own APIs. This brings a great opportunity for this innovative project to become a centralised system where, for example, busy venues can be listed with the tweets from their own Twitter or Facebook accounts and events can be reviewed in depth with the help of news and youtube videos.

[1]`http://flowingcity.com/visualization/tech-city-map/`

Chapter 3

Project Planning

3.1 Software design approach

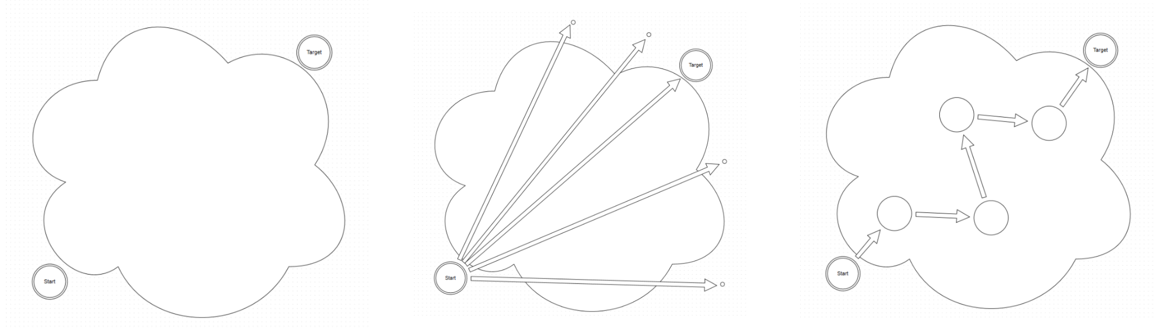
When a new product is developed, it is not clear how it will end up and if it will fulfil the user's requirements. The developers can see the first steps but there are plenty of problems or challenges that can not be predicted from the beginning (Figure 3.1a). That is true for any project, no matter how much planning is put in. However there is a possibility that everything is done the right way but there is a high probability to drift away from the initial target (Figure 3.1b). There are several methodologies that can be followed when developing a software but the two most common are Waterfall and Agile.

3.1.1 Waterfall model

The Waterfall model is an example of a plan-driven process - in principle you must plan and schedule all of the process activities before starting work on them [12]. Here iterations can be costly and involve significant rework. Therefore, after a small number of iterations, it is normal to freeze parts of the development, such as the specification, and continue with the later development stages. Problems are left for later resolution, programmed around or completely ignored. These implementation tricks may also lead to design problems and badly structured systems.

3.1.2 Agile model

This project, on the other hand, follows the Agile methodology (Figure 3.1c) for software development. It is an interactive approach that splits the work into a number of iterations (sprints). This allow the project supervisors to inspect the work of the developer and monitor how well the software development is progressing [12]. Each of these sprints last one week. It starts with a meeting and demonstration of the work done in the previous sprint. As an outcome of these meetings feedback from the supervisors is received, based on the extent to which their requirements are met. Moreover, these meetings allow discussions of the issues that are experienced and compare the available workarounds. At the end of a meeting, it is agreed upon exactly what work will be done during the next sprint. Following this scenario, at least one new feature is introduced after each sprint and changes to previous features are performed as early as possible. By the end of this project, 23 iterations are to be performed.



(a) Start of a project. (b) Waterfall methodology. (c) Agile methodology.

Figure 3.1: Software development methodologies

3.2 Requirements Engineering

The requirements for a system describes the functionality of the services it provides and the constraints of its operation. The process of gathering, analysing and documenting these services and constraints is called requirements engineering.

At the beginning of the project, a few weeks were taken to envision the high-level requirements and to understand the scope of the required system. For the initial requirements use cases and user stories were extracted to help with exploring how users will work with the system.

3.2.1 Use cases

Use case is a way to describe how a real user interacts with the system. They should not be perfect and can only show the action and not go into much detail, in order to stay close to the agile methodology. The developer will implement the system but will have to work close with the supervisors so that the end product will meet their needs.

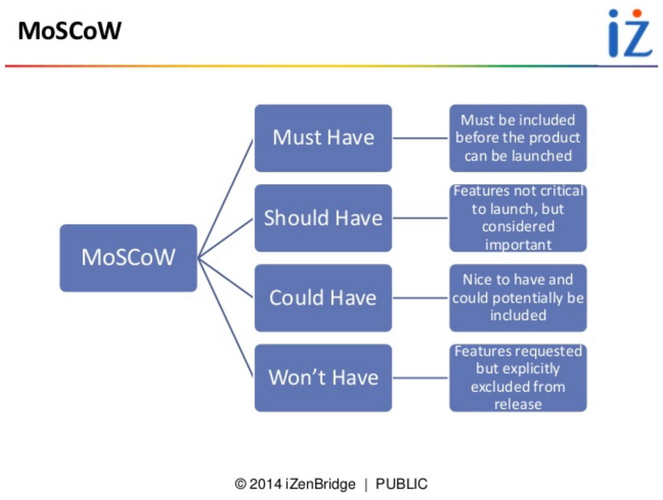
The following use cases were identified in the beginning of this project:

- see how tweets about a specific hashtag are distributed over a period of time
- see popular hashtags that are twitted together with a specific hashtag
- get information about a certain area of the city and find out if the venues(restaurants, train stations, cinemas) there get high attendance
- browse tweets based on time and hashtag
- use the system on a mobile device
- check out what the clients post about a venue, they had visited
- check how many people attend specific venues in a given area
- compare people's opinions about an event
- see if a political party is likely to win the elections

3.2.2 Functional requirements

Functional requirements represent services the system should provide, how it should react to particular inputs and how the system should behave in some situations. More specific requirements can describe the system functions, its inputs and output or exceptions in detail [12].

Adding new requirements may change the project direction. Projects usually have fixed duration so sometimes it may not be possible to include all the features. Therefore all use cases that are identified in this chapter must be prioritised.



MoSCoW
iZ
MoSCoW
Must Have
Must be included before the product can be launched
Should Have
Features not critical to launch, but considered important
Could Have
Nice to have and could potentially be included
Won't Have
Features requested but explicitly excluded from release
© 2014 iZenBridge | PUBLIC

Figure 3.2: MoSCoW rules prioritisation scheme for requirements

Figure 3.2 shows the MoSCoW rules that are typically used as a guideline for prioritising requirements. All the use cases should be sorted, using these rules. The ones that are ”Must Have” and ”Should Have” should be feasible in the duration of the project. Table 3.1 represents the extracted requirements after their prioritisation. It has now been modified according to the final list of requirements.

3.2.3 Non-Functional requirements

Non-functional requirements represent constraints on the services or functions offered by the system. They often apply to the application as a whole, rather than individual components [12]. Table 3.2 represents the extracted non-functional requirements for this project.

3.2.4 High Level System Diagram

The outlined requirements and use cases start to reveal what the system architecture is going to be. In order to achieve the goals, the application has to consist of three main components (Figure: 3.3): client to handle the interaction with the user, middleware to handle the application logic, and a database, in our case external services, to store the data. This architectural model is also called Three Tier Architecture. Tiers enable separation of concerns and encapsulate complexity as they can be broken down into layers and sub-tiers. Furthermore, they can be distributed across a number of machines to provide flexibility and can be replicated across a number of machines to provide the really important scalability factor[5].

Functional Requirements	
MoSCoW	**User Interface**
Must	be able to display a fusion of different kinds of timely data (tweets, weather, traffic, train delays etc.).
	be able to show specific as well as general information based on the request.
	display events in a timeline.
	display the time of events.
	be able to take date input.
	be able to receive word based input.
	be able to take location based input.
	be able to show venues in specific area. (using a Map)
	be able to show two events, side by side, so that they can be compared.
Should	dynamically add data to the layout. (using AJAX)
	be able to visualise statistic data on a graph.
Could	show a summary of the made request.
	show the link of an event with other events.
	Server
Must	be able to fetch data from provided services.
	be able to fetch data from external sources. (Twitter)
Should	provide additional RESTful APIs. (TODO: explain what is REST)
	support caching.
Could	store pre-render events.

Table 3.1: Functional Requirements

Non-Functional requirements	
MoSCoW	**Requirement**
Must	be universal to support all kinds of users(citizen, scientist, researchers, local authorities).
	be extensible so that different things can be rendered on the views.
	be able to quickly process the data from the services.
Should	be accessible from and compatible with desktops, laptops, tablets and smartphone devices.
	be testable.
	be scalable so that new services can be added.
Could	be compatible with tools that can measure code quality. (Sonar)

Table 3.2: Non-Functional Requirements

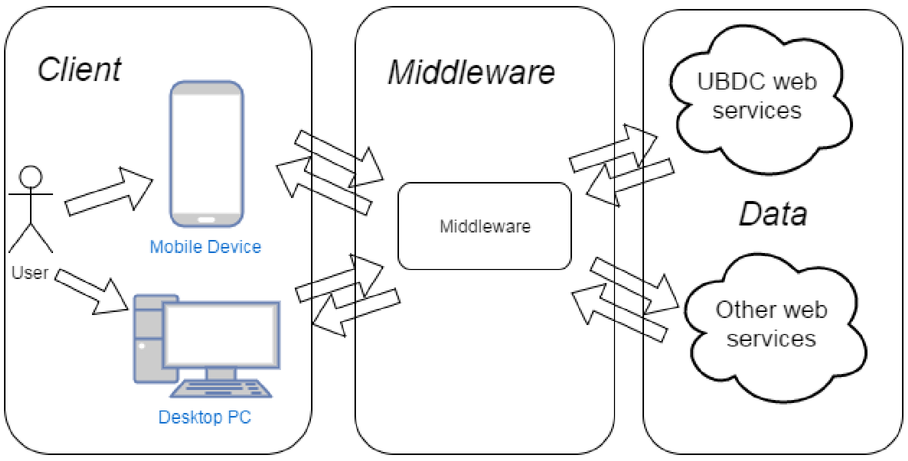


Figure 3.3: High level architecture diagram based on requirements and use cases.

3.2.5 Challenges

After summarising the requirements(Section: 3.2.2) and the high level diagram(Section: 3.2.4), some challenges can be identified. The biggest on is building a system that can be used by people without any experience in computing like citizens and satisfies the needs of researchers. Furthermore, the system should be valuable and usable on mobile devices. Taking into account how many services are to be used, the user interface must be designed carefully and efficiently. Additionally, the services are not implemented specifically for these application so a lot of unrelated data has to be filtered out. Moreover, there are multiple services and all the data, received from them, has to be sorted quickly.

3.2.6 User Stories

User stories are one of the primary development artefacts for Agile project teams. A user story is a high-level definition of a requirement, containing just enough information so that the developer can get a feel of what the user wants to achieve when using the system and performing a task. It represents a functionality that will be of a value to the user. A suggested template for a user story is:

As a ***ROLE***, I want to ***ACTION***, so that ***GOAL***. [11]

The role represents the user that interacts with the system. The action shows how the user wants to use the system and the goal- what the user is trying to accomplish. All user stories are written in a way that they are understandable by both developer and customer. Ideally the customer should write the user stories while discussing them with the developer [11]. The following list represents the user stories, captured for this project based on the identified target users:

- ***Researcher***.
  - As a Researcher, I would like to see how the tweets for certain query are distributed though out a period of time, so that I can see the pick of the number of tweets.
  - As a Researcher, I would like to know other popular twitter hashtags for a certain day, so that I can see what was interesting for the Twitter users for that day.
  - As a Researcher, I would like to know if specific area of the city was busy on a specific date, so that I can see if specific event or weather conditions had affected that area.
  - As a Researcher, I would like to see a timeline representation of the tweets for a specific query, so that I can follow and see if the users opinions change.
  - As a Researcher, I would like to use the app on a mobile device, so that when I travel, I can continue working on my research.
  - As a Researcher, I would like to know what were the weather conditions on a specific date and see traffic information, so that I can use the data as an experiment and measure the likelihood of using the public transport when the weather conditions are bad.
- ***Business owner***.
  - As a Restaurant owner, I would like to see if posting a tweet on my timeline affects the number of people that attend my restaurant.
  - As a Restaurant owner, I would like to see other venues in my area, so that I can check their attendance and try to improve mine.

  - As a TV channel owner, I would like to compare how many people are talking about my channel at specific time, compared to other channels so that I can change my TV guide and increase my audience.

- ***Politician***.

  - As a politician, I would like to compare how popular is my party in the social media in comparison to other parties, so that I can change my campaign and increase my chances of winning.

- ***Citizen***.

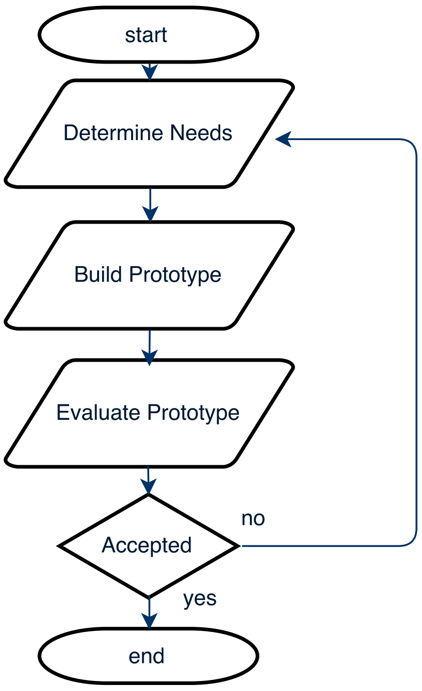
  - As a citizen, I would like to compare two different public opinions about a specific event, so that I can make a decision of my own.

Chapter 4

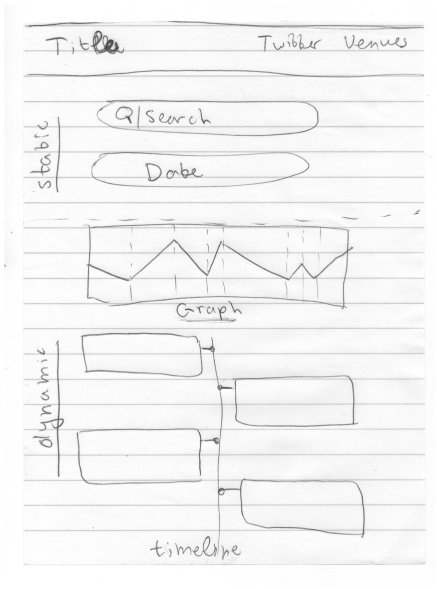
Design

4.1 Paper prototypes

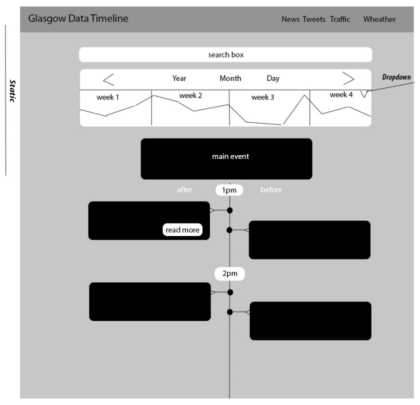
As shown in Figure 4.1a, designing an interface prototype is an iterative process that consists of four steps. The fist step is determining the needs of the users. This is already discussed in the previous chapter. The second step is building the actual prototype. It is good to start from sketches. Once a sketch (Figure: 4.1b) was completed, a wireframe (Figure: 4.1c) was prepared to be evaluated during the next meeting so that changes can be made before the start of the implementation.



(a) Prototyping process



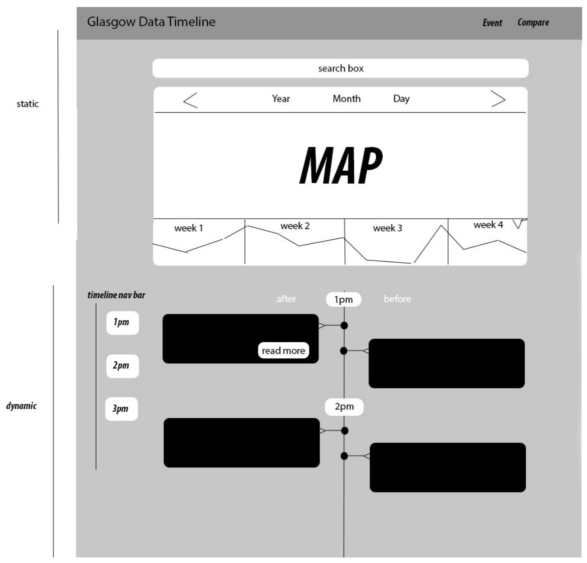
(b) Initial Paper Prototype.



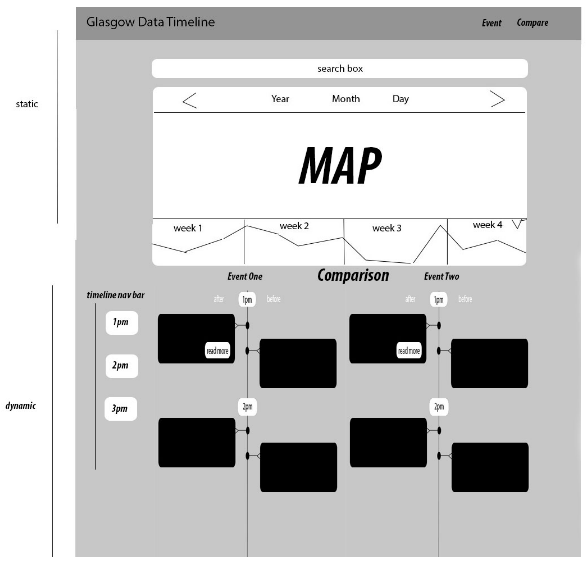
(c) Wireframe from the paper prototype

Figure 4.1: Paper Prototypes

During the implementation process, there had been a few changes to the design. The first few iterations were enough to visualise the initial prototype and it was time to think about what the user was going to extract from using the system. Key limitations of the design were identified as it narrowed the user experience by not providing visual representation of data, that was otherwise available for use to the system. Figures 4.2a and 4.2b show the new version of the design. A map component was included so any service, that uses location based input can be used. Also a Comparison Screen(Figure: 4.2b) was designed so that users will be given the opportunity to compare events side by side. The list of requirements and user stories was updated to reflect on these changes.



(a) Wireframe for event exploring



(b) Wireframe for comparing events

Figure 4.2: Wireframes

4.2 Architecture diagrams

The Urban Data Timeline makes use of the Three Tier Architectural model as described in Section 3.2.4. Based on the diagram 3.3 a more in-depth diagram was designed to outline some of the key components of the system (Figure: 4.3).

Client

The Client consists of the users and either a mobile device or a desktop PC. The users are going to access the application from the browser, running on their device. The browser talks to both the Google Charts API and Google Maps API, so that it can render the map and the chart, that appear in the wireframes(Figures: 4.2a and 4.2b). On the other hand, it talks to the Middleware, or in this case, the Controller component in order to get the information, required by the user.

Middleware

The Middleware consists of four components - Controller, View, Cache, and Model. The Controller talks to all of the other components. First of all, the required data is obtained by either the Cache or the Model. If the data is not in the Cache, the Model tries to get it from the external sources. If the communication was successful, the data is stored in the Cache and returned to the Controller. From there, the Controller sends that data to the View so that the page can be prepared and returned to the user.

Data

The Data currently consists of two services. The first one is the UBDC web service and it is key to this project as it separates into Twitter, Busy Venue, Delayed Transport, Train and Weather services(brief description of the services can be found at Section 1.2 and 2). As for now, only the first four services are used. The second one is the Twitter API. It is used to compensate for the limitation of the Twitter service, provided by the UBDC, which only allows for queering the results based on hashtags. The additional API will broaden the possibilities of tweets querying by adding the option to search by username and get access to tweets from specific twitter account.

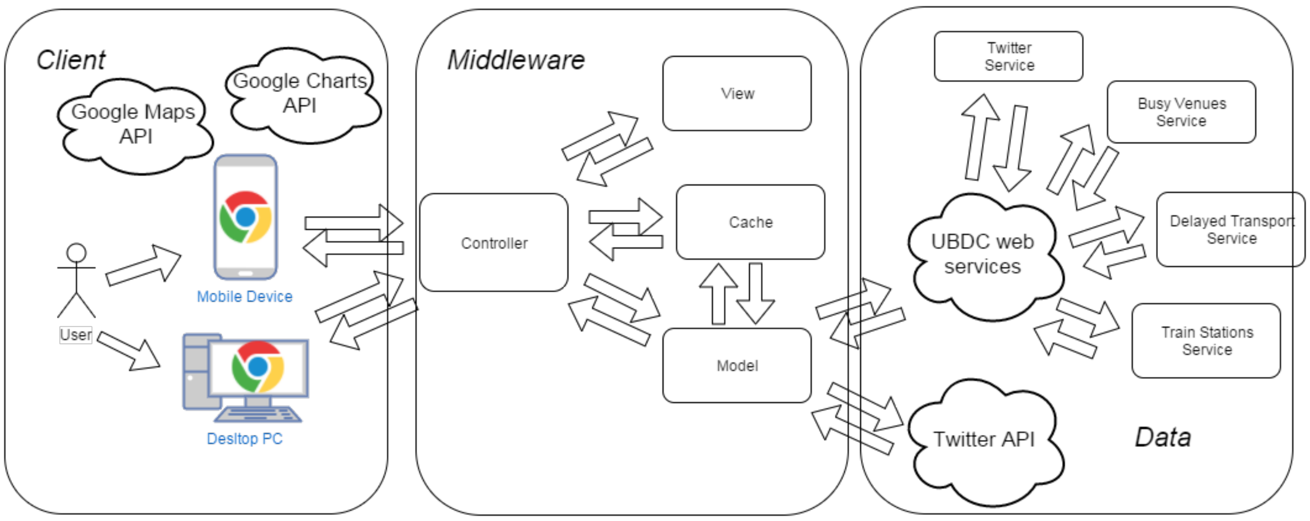


Figure 4.3: Component level architecture diagram.

4.3 Components communication

This section describes how actually the components are communicating between each other. There are two scenarios, based on what request is made by the browser.

4.3.1 HTTP Request

The first one(Figure: 4.4) follows the traditional pattern when a user wants to access a page, which visualises data, using the application. Taking as an example one of the user stories from Section 3.2.6, a researcher is on the home page and wants to know what are the popular tags for a given day. First step is to fill all the required inputs and then press on the search button. Then a new page is rendered on the screen and the researcher can start interacting. However, for all this to happen, the browser has to communicate with the application's server. To begin with, it makes a HTTP GET request to the controller, that maps to the action, triggered by the search button. Then all the inputs are validated against a set of rules and analysed so that the right Model can be invoked. Then the data is requested from the Model. The Model checks the Cache to see if the data is available there and if yes, returns it to the Controller. On the other hand, if the data is not available, a XMLHTTP POST Request is made to a specific Web Service and a XMLHTTP Response is received in the form of a Json object. Depending on the initial request by the researcher, some of the data from the Json is filtered out and the rest is stored in the Cache for a specified amount of time. Cache is not only part of the requirements but it is important as users, like researchers, are expected to do many similar requests by changing only one or two variables at a time. Once the model has finished, manipulating the data, it is sent back to the Controller that redirects it to the View. The

View formats the data, according to its type, and sends it back to the Controller. Finally the Controller returns an HTTP Response with the generated HTML.

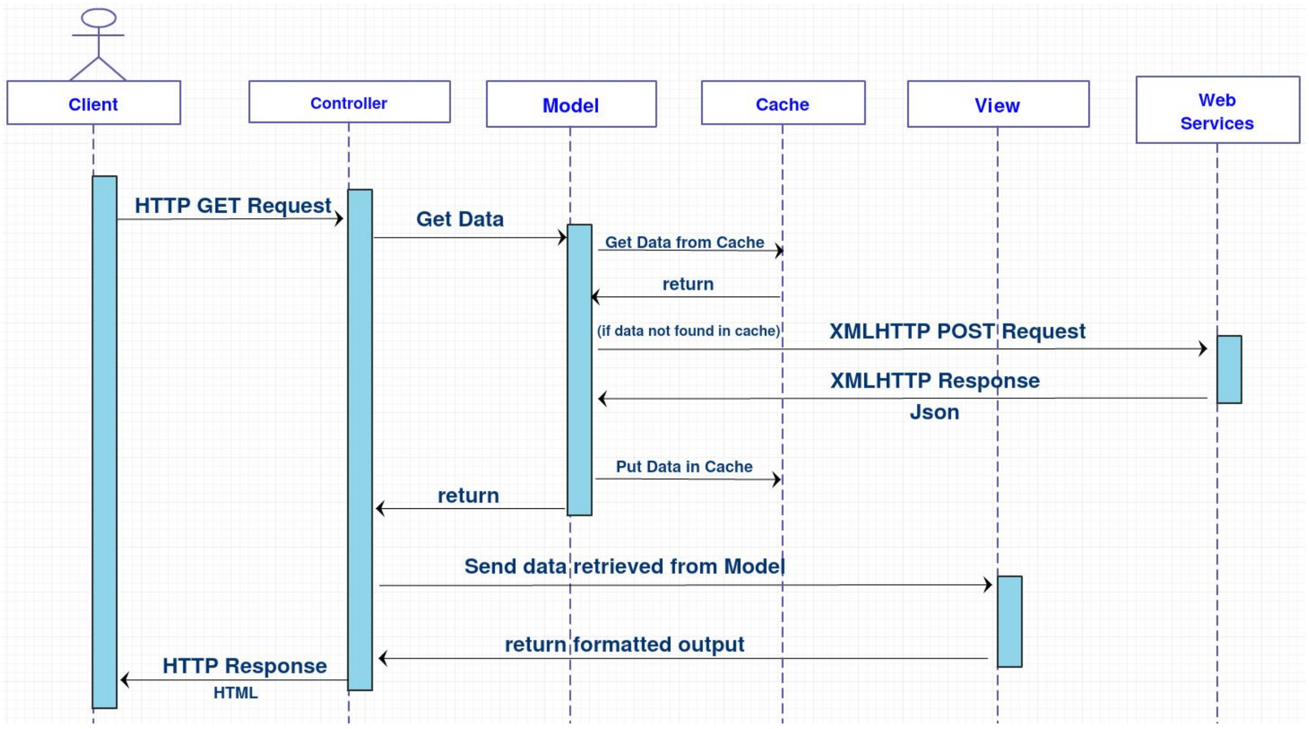


Figure 4.4: HTTP Request Sequence diagram

4.3.2 XMLHTTP Request

Following from the requirements(Section: 3.2.2) and the paper prototypes(Section: 4.1), some of the elements(Map and Chart) can be rendered by the View but in order to make them responsive, these elements have to use Ajax in order to be populated. Taking as an example one of the user stories from Section 3.2.6, a restaurant owner wants to view venues that are nearby but also may want to get the same information but for a place, which is not the same as his current location. In order not to reload the page every time, Ajax performs XMLHTTP Requests instead of HTTP Request and the received response will be used to update the page dynamically. In order to handle such requests, the system must provide REST endpoints that are compatible. If the browser makes a XMLHTTP request(Figure: 4.5) to one of the endpoints, the Controller, similar to the previous case, performs the same steps but in this case there is no communication with the View. The data, returned from the services, is filtered and sent, via a XMLHTTP Response in a Json format, back to the client.

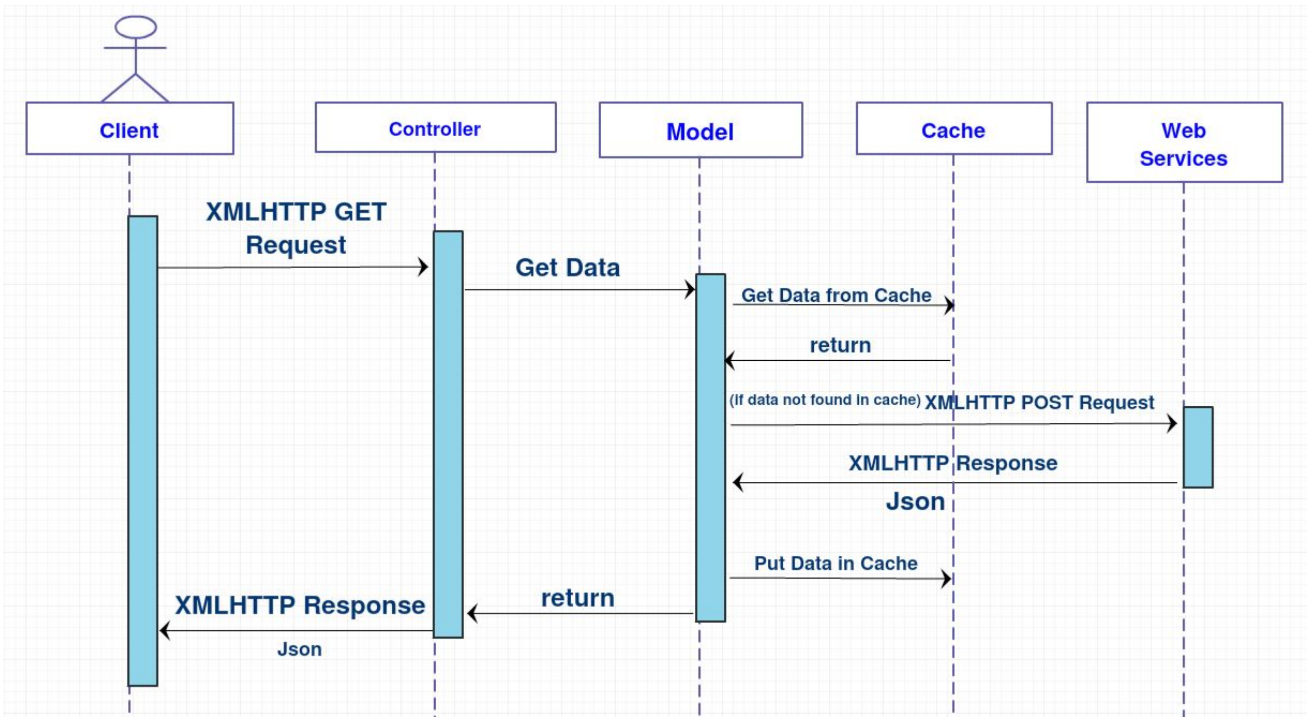


Figure 4.5: XMLHTTP Request Sequence diagram

4.4 Choice of technology

One of the requirements (Section: 3.2) states that the application should be accessible and compatible with desktops, laptops, tablets and smartphone devices. This limits the options in terms of platforms that can be used. A common tool for all these devices is the web browser. For desktops and laptops, this is a great choice as there is a big variety of browsers. On the other hand, this introduces another challenge to the development as some technologies are not compatible with some older browsers like Internet Explorer 7 or lower. Additionally, native applications, developed specificity for a mobile platform like Android or iOS tend to perform better than their web alternatives but will take longer to implement so they are not suitable for such project. This section will present technologies, which will fulfil the requirements.

4.4.1 Security

Security is top priority when developing a web application. As this project will be used only for displaying data, there are not many vulnerabilities that can be targeted by attackers. First of all, the system will not make use of a database so any attempts for performing SQL Injection, Cross-Site Request Forgery, Sensitive Data Exposure or Cross Site Scripting (XSS) will not lead to a security breach. However, as a safety precaution, all inputs that are received must be validated before being passed to any of the external services shown on Figure 4.3. Furthermore, the application will not require authentication so it is not vulnerable to Insecure Direct Object References or Broken Authentication via stolen cookies. The only possible vulnerabilities are Denial of Service and Man in the middle attacks. For this project, no defence mechanisms will be developed against these attacks. The first one can be mitigated by following the web traffic and blocking suspicious requests. The result of a successful Denial of Service attack will be that users will not be able to access the system due to the server, on which the application is running, has be overloaded with requests and is not able to respond. Man in the middle is a form of attack where the user's network has been breached and an attacker is modifying their traffic. The way to defend against this attack is by introducing the HTTPS protocol that ensures the uniqueness of a website but it requires an expensive certificate. However, all of the following technologies have to be compatible with the HTTPS protocol in case that it is decided to be included in a future iteration.

4.4.2 Web Application Framework

Play Framework

Play Framework 2[1] is is an open source web application framework, written in Scala and Java, which follows the modelviewcontroller (mvc) architectural pattern.
Pros:

- It dramatically improves the productivity of a developer compared to other Java based web frameworks like Jersey or Spring MVC. The server does not have to be restarted manually in order to see the changes. *Hot reload* is available for all Java classes, templates and configurations allows for much rapid development. This is available in many dynamic languages, but it is not provided in any other Java framework.
- Play 2 is open source. If required, everything can be seen how it works. It has a relatively large community, represented by questions on StackOverflow and developers that contribute with developing plug-ins.
- Java: type safe language. Also JVM performs great and can scale to support many developers and users. Furthermore, Java by itself has a large community and wide variety of IDEs[2] and available libraries.

Cons:

- Play 2 is a relatively new framework so the community is not as big compared to other Java frameworks.
- The framework is immature. Good practises are still no clearly defined.
- Java does not provide features like closures to keep asynchronous code clean. Play is built on top of asynchronous input and output. There are ways to go around this but the application will end up with lots of inner anonymous classes, which will reduce the maintainability and readability of the code.
- Java does not support Json natively so a third-party library must be used. This, together with its strict types, will lead to big overheads when parsing Json, which is going to be the main source of data for this project.

Django

Django[3] is free and open source web application framework, written in Python. Its main focus is helping developers to write code, without the need to reinvent the wheel.
Pros:

- Django follows the DRY principle: Do not Repeat Yourself. The frameworks is designed so that developers can get the most out of a little piece of code. In addition, this automatically leads to less hours spent in developing and a lower chance of introducing bugs.

[1]`https://www.playframework.com`
[2]Integrated Development Environment
[3]`https://www.djangoproject.com`

- Good documentation. Django provides sufficient documentation for every release with plenty of code examples. Furthermore, if something is not documented, the code is publicly available on GitHub so it can be directly inspected from there or from an IDE like PyCharm.
- Django provides build-in Admin panel, that is generated automatically for each project. It allows users to manipulate and control users or database objects, specific to the application.
- Django is scalable as it is designed on a component based architecture. All the components are decoupled and does not depend on each other so they can be easily unplugged and replaces. Likewise, new components can be simple to introduce.
- Django provides a set of tools that can be useful when writing tests. Tests can be perform not only on the Models and the Controllers, but also on the Views without a third-party library. It provides a build-in request factory, which uses URL resolution to trigger the views.

Cons:

- Django is using Python, that is an interpreted language, and it is often slower than compiled languages.

Laravel

Laravel[4] is a free, open-source PHP web framework, intended for the development of web applications following the modelviewcontroller (MVC) architectural pattern.
Pros:

- Composer - packaging system for PHP and is used for dependency management. Laravel is designed using a component based architecture and the whole framework is available as individual Composer packages.
- Blade is the Laravel's template engine that is lightweight and provides clean syntax for views. It supports template inheritance that reduces the duplication of code and re-usability of specific template components.
- Resourceful controllers: generic routes can be made that directly map to resources in the controller. Makes developing REST a bit easier.
- Laravel is built with testing in mind. It supports PHPUnit directly out of the box. Same as with Django, it provides helper methods that allows for expressive testing.
- Error messages are easy to understand and points directly to the error.
- PHP has built in Json support.

Cons:

- PHP: inconsistent function names in the standard library(for example: isset() and isnull())
- Same as with Python, PHP is an interpreted language so that it is slower than compiled languages. However, PHP 7 is advertised to have a big performance improvement over PHP 5.

[4] `https://laravel.com`

Choice

Laravel was chosen for this project. It is very similar to Django and offers the same features. In addition, PHP 5 and Python are both interpreted languages with similar performance and have built in Json support. However, updating in a future iteration to PHP 7 with its improved performance and speed, twice as fast as PHP 5.6[5], will reduce the computation time that is taken by sorting and filtering the data from the services. Finally, Laravel is compatible with HTTPS requests and will allow for a future security update. On the other hand, Play Framework 2 is not suitable for this project as it urges on using Java, which will bring a big overhead when processing Json objects.

4.4.3 Javascript Library

In order to improve the user experience and embed a chart and a map into the design, an asynchronous Javascript library is required to simplify the calls to the REST endpoints. Two options were researched: jQuery and AngularJS.

jQuery[6] is a lightweight Javascript library, which comes with many features. It broadly simplifies the use of Javascript for client-side scripting. It can:

- manipulate the content of a web page.
- make use of built in effects and animations.
- easily make Ajax requests.
- traverse though the DOM.

AngularJS[7] is a MVC framework, developed by Google. Compared to jQuery it has more features but being a framework emphasis on following its rules. It can:

- make use of templates.
- validate forms.
- perform Ajax requests.
- be tested.

Choice

JQuery was chosen for this project as it is a fast and feature-rich Javascript Library. It allows for more flexibility in the implementation as it is not a framework like AngularJS.

4.4.4 CSS Framework

With the rise of mobile devices, making responsive websites and keeping up with the latest technologies is very time consuming and hard to maintain. There are many CSS frameworks that solve these problems but Bootstrap[8] and Skeleton[9] were the one compared for this project. They both are very simple to get started with: just reference the framework into the header of the page. Both offer 12 column grid system as well as many layouts and components. One of the most important features, that is offered by both frameworks, is providing responsive utility classes that can automatically rearrange a page based on the screen size of the device. However,

[5]http://php.net/releases/7_0_0.php
[6]https://jquery.com
[7]https://angularjs.org
[8]http://getbootstrap.com
[9]http://getskeleton.com

Bootstrap was the one, chosen for this project as, first of all, it has the best documentation. Bootstrap also comes with Javascript plug-ins like drop downs, tool-tips, pop-ups, sliders and many more.

4.4.5 Graph API

Following from the requirements(Section: 3.2.2), the application should provide graph visualisation. For this purpose, two APIs were considered: D3.js and Google Charts.

Google Charts[10]:

- contains a wide varsity of charts.
- has a great documentation.
- offers fully implemented examples for all charts.
- provides extra features like exporting charts as images.

D3.js[11]:

- offers flexibility.
- does not have a limit on how much data to display.
- provides extra features like zooming and clicking.
- can be used for creating very complex graphs.

Choice

Google Charts Graph API was chosen for this project due to its simplicity and great documentation. Furthermore, there is no intention of building big and complex graphs in the time, allocated for this project.

4.4.6 Map API

Going back to the requirements (Section: 3.2.2) and the paper prototypes(Section: 4.1), the application must be able to show venues on a map and also to take location based input. To simplify the implementation, it will be useful to find an API that provides functionality to fulfil both requirements. Google Maps API[12] was chosen for this project as it is the only one that is free and has built it support for all the required features. In addition, it offers 25,000 map loads per 24 hours for 90 days[8]. It also provides wide variety of customisable options like colours, shapes, markers and many more. Furthermore, Google Places[2], part of the Google Maps API, features more than 100 million businesses and points of interest that are updated frequently through owner-verified listings and user-moderated contributions. It gives the ability to search for specific place and can be configured to display details like ratings and reviews.

[10] `https://developers.google.com/chart/`
[11] `https://d3js.org`
[12] `https://developers.google.com/maps/`

Chapter 5

Implementation

5.1 Prototype Implementation

The first three iterations of the project focused on the background research, project planning, requirements gathering and choosing the technologies. Furthermore, the fourth iteration involved producing a prototype out of the wireframes(Figure: 4.1c). A simple static HTML page was developed to represent the key components: search box, date picker, chart and timeline. The search box(Figure: 5.1) was a simple input html field. Its purpose was to get user's input. Also a hint was included(*input a query*) in order to make it clear what this field was supposed to take as parameters. The date picker was implemented using dropdown menus for the day, the month and the year. Two additional buttons were added on both sides to help the user to quickly adjust the date input without the need to go and use the menus again.

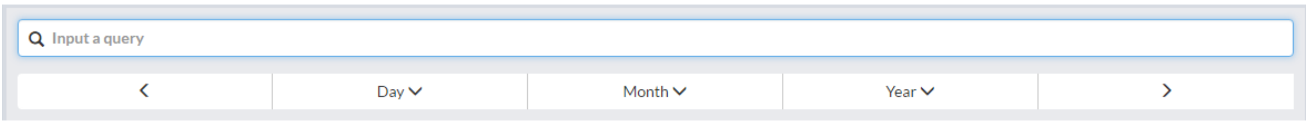
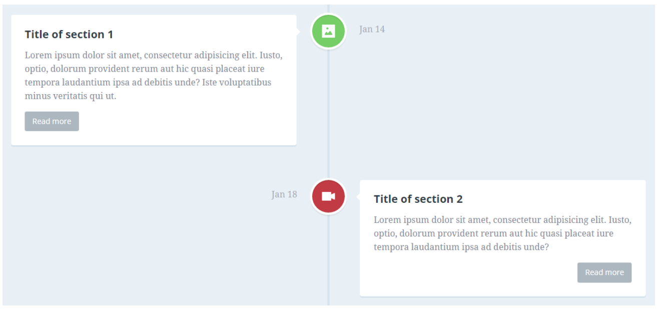


Figure 5.1: Search Box and Date picker

For the purpose of the prototype, an open-source vertical timeline was used. It was implemented using jQuery and CSS3, which perfectly suited the technologies(4.4), used for this project. With its responsiveness and clear design, it was a perfect choice for representing the initial outline of the interface. During this iteration, no back-end logic was introduced, as main focus was evaluating and finalising the design before the start of the next iteration.



Title of section 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Iusto, optio, dolorum provident rerum aut hic quasi placeat iure tempora laudantium ipsa ad debitis unde? Iste voluptatibus minus veritatis qui ut.

Title of section 2

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Iusto, optio, dolorum provident rerum aut hic quasi placeat iure tempora laudantium ipsa ad debitis unde?

Figure 5.2: Initial timeline

5.2 Product Implementation

Accordingly, during the fifth iteration, it was decided to move the prototype to a the Model-View-Controller framework, that was chosen in Section 4.4 so that communication with the services can be introduced.

5.2.1 Model

The models, as shown on Figure: 4.3, were represented by an external service. At the start, in order to learn how the framework worked, each model was implemented separately. However, this lead to a lot of repetitive code so another approach was considered. As PHP supported Object-Oriented programming, the common features from all the models were extracted and an Abstract class was implemented.

```
abstract class AbstractService
{
/** holds the service url*/
protected $url;

/** holds the input query*/
protected $query;

/** holds the request parameters*/
protected $postData;

/** holds the input date*/
public $queryDate;

/** holds the response from the services*/
public $response;

abstract public function getCount();

abstract public function getData();
}
public function sendRequest($arrayToSend)
```

Using the abstract class mentioned above made including another service really easy. The public methods *sendRequest* was common for all models as it was used for communication with the services. The parameter, that it took, *$arraytoSend* represented the array of parameters, that the service required as inputs. The methods *getCount()* and *getData()* were implemented in each Model accordingly. The idea behind them is to process, filter and format the response from a service and return either the count or an array of entries, that the user had required.

In addition, the Busy Venues service was returning information about the venues, which in some cases included twitter account. However, the Twitter service did not allow searching for tweets, based on user's account. In order to compensate for this limitation and increase the amount of data sources, it was decided to include communication with the genuine Twitter API. With Laravel's flexible design and high separation of concerns, it was possible to integrate an additional component that would perform the required communication. Thujohn/twitter[1] was a php component that could have be integrated with Laravel and allowed for extracting a collection of the most recent Tweets, posted by specific user. It allowed for searching base on screen_name or user_id parameters. The only requirement for it to work was creating a Twitter account and enabling the API by generating the appropriate keys and tokens.

[1]https://packagist.org/packages/thujohn/twitter

```
return Twitter::getUserTimeline(['screen_name' => 'thujohn', 'count' => 20, 'format' => 'json']);
```

Challenges

At the beginning of the project, the services were still under development. A few problems were identified at the beginning of the development. First of all, the twitter service was not returning a timestamp for each tweet. As a timeline was built, the time was a key factor. To go around this issues, until the services were fixed, a random time generator was developed so that the tweets can be listed on the timeline. Furthermore, the Busy Venue service was behaving not as expected as some venues, despite the fact that they were returned with score higher than 0, they did not seem to have any entries (all entries were set to 0). On the other hand, some venues with score 0 had entries greater than 0. Again, another random generator, in this case generating attendance, was introduced so that venues could be visualised on the timeline.

Consequently, after introducing the communication with Twitter, registering to use the API for free came with some limitations. Twitter restricted the usage to 180 requests in 15 minutes and advised the developers to cache the responses locally. As Laravel supports caching, it was decided to implemented this extra functionality and use file system cache in order not to fill up the memory of the server, as well as, not to increase the overhead of the project by designing and introducing a database.

```
    public function sendRequest($username)
        {
        $user = $username;
        $cacheTag = 'twitterTimeline'; //config timeline twitter
        $cacheKey = $user . "-" . $cacheTag;
        $cacheLimit = 15;
        $tweets = null;

        /* caching */
        if (Cache::has($cacheKey)) {
            $tweets = Cache::get($cacheKey);
        } else {
            $tweets = Twitter::getUserTimeline(['screen_name' => $user, 'count' => 10, 'format' => '↩
                object']);
            Cache::put($cacheKey, $tweets, $cacheLimit);
        }
        return $tweets;
    }
```

5.2.2 View

chart

 Designed it using Bar chart instead of Line graph Designed my own bar chart so that I can keep up with the clean design so far. Managed to implement some design features: o Showing value of the bar while hovering over

 Currently the charts shows the number of tweets for a period of 10 days On the y-axis there is a scale, computed from the maximum value On the x-axis are the corresponding dates

 Also made the chart to draw any number of charts passed from the Controller so if a new service is add, there will be no need to change the way the chart works. o HOW IT WORKS controller returns a response in the form:

”yesscotland”:”twitter”:[”date”:”2014-08-21”,”count”:126,”date”:”2014-08-22”,”count”:629,],”venues”:[”date”:”2014-08-21 12:08:00”,”count”:7,”date”:”2014-08-22 12:08:00”,”count”:7] - client jQuery code reads this and for each key it creates a button and stores the array corresponding to that key. On button click, jQuery gets the name of the button and draw the data corresponding to that name.

Compare screen A screen where two separate events can be queried at the same time so that they can be compared side by side Challenges: o Currently my design was for single search space so that I had to come up with approach that will separate both search spaces that we introduce Solution: Pass ID parameters from the Controller to the View so that it can be appended to each ID of each valuable element from it. E.g. id=day will become id=dayFirst and so on. Positives: it can be extendable to support as many search spaces as desired. o Had to become with standard way of passing data from Controller to View and migrate my current code to support it.

Scroll-To-Top Implemented that feature because at some scenarios the response might be quite long and it will take you a lot of time to scroll to the top of the page and perform another search.

Problem My current choice for a timeline doesnt scale properly when used in the comparison screen so I may have to look for alternative component.

5.2.3 Controller

5.2.4 Challenges

Validation Research Javascript validation o Advantage live update if a field is valid or not o Disadvantage update appear only under the input field. Also cannot modify error messages style, only if I dig in the implementation of the plugin and make the changes. Furthermore the request can still be manipulated. PHP Validation component that comes with Laravel o Advantage can use any style and can place the error messages anywhere on the page. Works on the server which makes it even more secure as it validates the request. o Disadvantage needs a page reload to show the errors Implementation Every controller starts with validating the request. There are custom rules for every field from the request. If the validation passes, the controller does its job, otherwise the user is redirected to the same page with error messages under the fields that are required or have wrong input. Error messages fields are displayed using Bootstrap CSS in a red colour, so that the user knows that there is an error.

Chapter 6

Evaluation

6.1 Product evaluation

6.1.1 Planning

The first step from the evaluation planning was clarifying the goals and objectives. Main goal was finding out if the implemented features are actually of use to the users. In addition, it was important to get feedback on how usable is the system and extract possible improvements. The second step was identifying the techniques that were going to be used so that the experiment can be designed. To address the main goal, a carefully selected list of tasks was formed so that the participants can go through the whole application and use most of the features. Furthermore, to try and measure the usability of the system, a System Usability Scale[1](SUS) questionnaire was prepared. It consists of a 10 questions with five response options: from Strongly agree to Strongly disagree. It is very beneficial for this project as it can be used on small sample sizes and there is a detailed algorithm how to calculate the results[2]. Additionally, a Think Aloud was decided to be performed in the end of the experiment. It is a form of observation where the participant is asked to talk through the steps, needed to complete the tasks[1]. Think aloud has the advantage of simplicity. It requires little expertise to perform and can provide useful insight into problems with an interface. It can also be employed to observe how the system is actually used and outline possible improvements.

Finally, the evaluation process looked as follows: first the participants performed the list of tasks, then they were given a SUS questionnaire and in the end was the Think Aloud. The actual evaluation was decided to be spilt into two parts: initial and final phase.

6.1.2 Initial phase

The initial phase was crucial as it allowed for piloting the evaluation. It represents a small scale preliminary study conducted in order to evaluate feasibility, time, and cost in an attempt to predict an appropriate sample size prior to performance of the final full-scale evaluation[7]. Piloting allowed for improvements upon the design of the experiment by fixing bugs in the software and see if the initially planned evaluation would have given the desired results. However, the final version of the system was not used in this phase as some of its features were still under development.

[1]`http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`

[2]`http://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-y`

Results

Four students from the University of Glasgow participated in the Initial phase. Two of them were studying Computer Science, one Music and the last one, Sociology. The average time for completing the whole process was 9 minutes and 27 seconds. Three tasks (the list can be seen at Table: 6.1) were given to each participant, followed by the SUS questionnaire. Based on research, a SUS score above a 68(out of 100) would be considered above average. The score that the system got at these stage was 75. This proves that the participants liked the system overall and felt that it was easy to use. However, 3 out of 4 participants disagreed on the first question of the questionnaire that states: I think that I would like to use this system frequently. When asked during the Think Aloud, main reason for that was they do not use twitter or they are not doing a research, as this application would have been great for researchers. Proof of that was the sociology student that agreed on this question. When asked what they think about the application as a whole, 2 out of 4 said that it is quite unique and they have not used any similar system so far.

The Think Aloud was also beneficial in terms of feedback on the design and the features of the system. For three of the participants, the buttons under the graph were not clear if they are for the graph or for the timeline and they suggested for better separation of the components. Two of the participants did not notice some of the features(the navigation bar for the timeline and the information box that pops up from the top) and when asked afterwards, said that these were useful but their main focus was on completing the tasks and did not look for hints. All of the participants liked the clearness of the design and the graphics. Also one of them suggested alternative use case about using the application in television shows to present people's opinions about a specific topic. Furthermore, two of them said that they could use the system during a political event and see how other people think, which completely overlaps with one of the use cases in Section 3.2.1

List of tasks: Initial phase of the Evaluation	
Number	**Task**
1	Explore the timeseries of Ubiquitous Chip(Venue, located on Ashton Lane) for 23th of August 2014 and find out how many check-ins the venue had at 1pm.
2	Find how many tweets with the query word - "noscotland" are tweeted on the 27th of August 2014.
3	Compare the number of tweets with query words - "noscotland" and "yesscotland" on the 29th of August 2014 at 2am.

Table 6.1: Initial Evaluation tasks list

6.1.3 Final phase

The feedback from the initial phase was positive but did not fully achieved the main goal of the evaluation: proving that the features were actually of use to the users. In order to improve, the first section of the evaluation process had to be redesigned. Two hypothesis were introduced. The first one was that the time of completing a task was affected by the interface. The second one tested if the interface had an affect on the number of clicks, performed for completing the task. In order to complete the experiment, A/B testing was used. It is a technique that compares two versions of a single variable and finds out which of the two variables is more effective. The idea, when testing a web application, is to show two versions of the interface to similar participants at the same time. In the case of this project, the desktop and mobile versions offer similar features. To solve this, a third system was extracted, where some of the features were disabled. However, they were selected in a way that will not prevent the participants from performing their tasks. The information box, timeline navigation bar, sections headings, which show in what timeslot is the current event on the screen, bar graph and infinite scrolling were all of the removed features. On the other hand, the full system was improved, as suggested in the initial evaluation, and components were separated by following Google's Material Design[3] and visualising them as Cards[4]. Cards are a convenient means of displaying content composed of different elements. They are also well-suited for showcasing elements whose size or supported actions vary(**add screenshot**).

Number of Participants	Interface	Task	Interface	Task
2	1	1	2	2
2	2	1	1	2
2	1	2	2	1
2	2	2	1	1
2	1	1	3	2
2	3	1	1	2

Table 6.2: Distribution of Participants per Task and Interface after Counterbalancing

In addition, counterbalancing method was used in the designing of the final experiment. It can be defined as using all of the possible orders of conditions to control order effects[3]. 12 participants were allocated for this experiment and Table 6.2 shows their distribution according to the task and interface. The participants first had to complete a task, using one version of the interface, and then fill the questionnaire about that particular interface. Then, they were given another version and a different task with the same questionnaire in the end. This technique was used in order to try and minimize the disadvantages, such as learning effect and boredom, of paired experiments. The Think Aloud was kept in the end, trying to focus more on comparing the evaluated systems. Additionally, the list of tasks was modified (Table 6.3) to try and make the participants interact more with the system. Due to time constraints, the functionality behind the third task was not fully implemented and it was given as optional for the participants, who were keen on discussing it.

List of tasks: Final phase of the Evaluation	
Number	**Task**
1	Explore the time-series of Ubiquitous Chip(Venue, located on Ashton Lane) for 23th of August 2014 and find out how many check-ins the venue had at 1pm.
2	Compare tweets with hash-tags - ”nosctoland” and ”yesscotland” on the 29th of August 2014 at 2am by writing down one or two sentences about what were the people's opinions at that time.
3	Find out if there were any delays in Glasgow Central on the 5th of September 2014 and if there were, write down a sentence, summarising people's opinions about them(if there were any).

Table 6.3: Final Evaluation tasks list

Results from Task

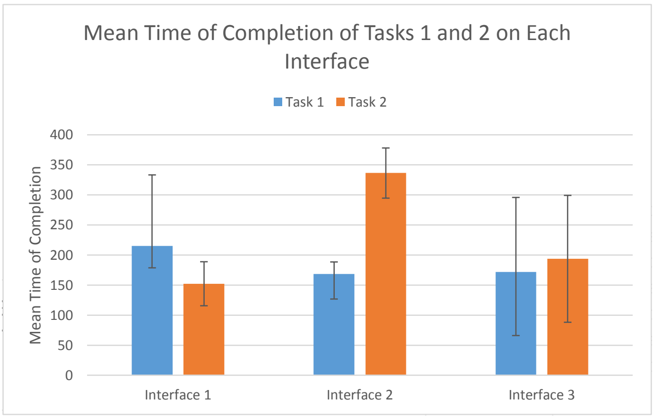
The first hypothesis suggested that there will be an interaction between the independent variables of task and interface on total completion time. The mean total time of completing tasks 1 and 2 on each of the three interfaces were obtained and are presented in Table 6.4 and illustrated in Figure 6.1a with error bars identifying Standard Deviation. The lowest mean value for Task 1 was the completion using Interface 2 (M=168.52, SD=19.95). The highest value on Task 1 was obtained using Interface 1 (M=215.19, SD=117.97). Accordingly, Task 2 was

[3]https://www.google.com/design/spec/material-design/introduction.html
[4]https://www.google.com/design/spec/components/cards.html

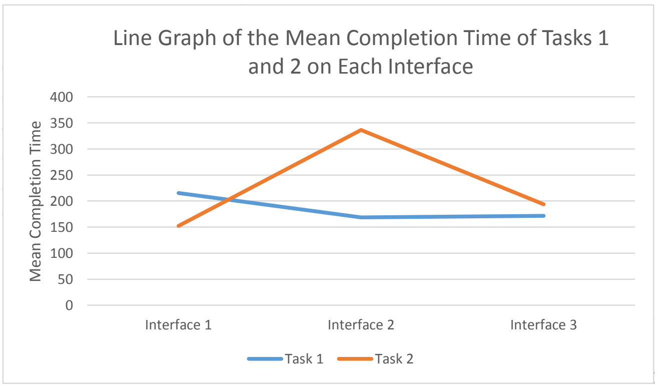
completed fastest on Interface 1 (M=152.19, SD=36.62) and slowest on Interface 2 (M=336,35, SD=41.73). The lowest total time of completing both tasks is on Interface 3 (365.28). When being plotted on a line graph, the obtained values suggest interaction (Figure 6.1b). Therefore, the descriptive statistics seem to support the first testing hypothesis.

Task\Interface Time(s)	Interface 1	Interface 2	Interface 3
Task 1	215.30 (117.97)	168.52 (19.95)	171.62 (124.19)
Task 2	152.19 (36.62)	336.35 (41.73)	193.66 (105.56)
Total Time(s)	367.49	504.87	365.28

Table 6.4: Mean Time for Task, performed on Interface



(a) Mean Time Bar Graph



(b) Mean Time Line Graph

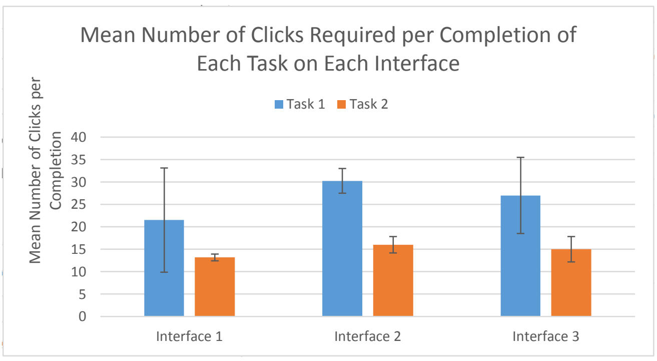
Figure 6.1: Results for Dependant Variable: Time

A 3x2 mixed factorial ANOVA was used to analyse the data. A significant interaction between Task and Interface was found (F(2,18)=5.058, p¡0.05, 2=0.360). However, there were no significant main effect of Interface (F(2,18)=1.866, p=0.185, 2=0.172) an of Task (F(1,18)=1.903, p=0.185, 2=0.096).

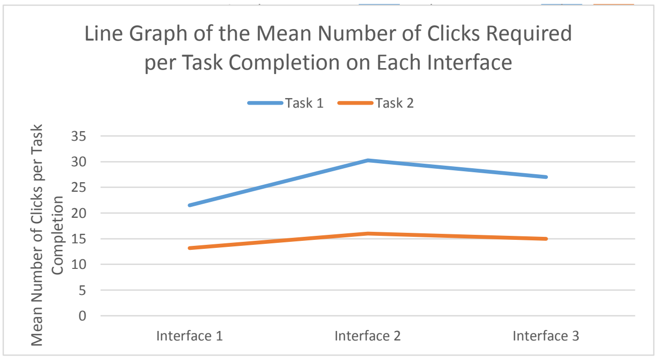
The second hypothesis suggested that there will be an interaction between Task and Interface on Total Number of Clicks. The mean number of clicks for completion of a task using a particular Interface were obtained and presented in Table 6.5. These were also illustrated along with errors bars for Standard Deviation on Figure 6.1a. As it is shown, Task 1 is on average completed with the least number of clicks on Interface 1 (M=21.50, SD=11.62), and largest number on Interface 2 (M=30.25, SD=2.754). The best mean value on Task 2 was also completed on Interface 1 (M=13.17, SD=0.753), and the highest mean value on Interface 2 (M=16, SD=1.826). The lowest total number of clicks required for completion of both tasks was obtained on Interface 1 (34.67). Furthermore, as it is illustrated in Figure 6.2b, when plotted on a line graph, there is no evidence of interaction.

Task\Interface Clicks(count)	Interface 1	Interface 2	Interface 3
Task 1	21.50 (11.62)	30.25 (2.754)	27.00 (8.49)
Task 2	13.17 (0.753)	16.00 (1.826)	15.00 (2.828)
Total Clicks(count)	34.67	46.25	42.00

Table 6.5: Mean Number of Clicks for Task, performed on Interface



(a) Mean Number of Clicks Bar Graph



(b) Mean Number of Clicks Line Graph

Figure 6.2: Results for Dependant Variable: Clicks

A second 3x2 mixed factorial ANOVA of Task (1*2) and Interface (1*2*3) was run to explore the number of clicks. No significant interaction between Task and Interface was found (F=(2,18), p=0.618, 2=0.052). Also there was no main effect of Interface on number of clicks per task completion (F=(2,18)=1.901, p=0.178, 2=0.174). However, a significant main effect of Task was found (F(1,18)=14.844, p¡0.05, 2=0.52).

Results from Questionnaire

All the participants had to complete the System Usability Scale questionnaire twice, once after each task and interface. For interface 1, representing the system with all the features, it was filled 12 times, for interface 2(the mobile version): 8 times and for interface 3(the version with the removed features): 4 times. Figure 6.3 shows the scores after calculating the results. The biggest mean score is for interface 1(89.38), followed by interface 2(87.50) and interface 3(84.25). The fact that only 4 people filled in the questionnaire for interface 3 did not affect the overall score in this case as when further analysed, all of them had disagreed on the same statement: I found the various functions in this system were well integrated. In comparison, 3 of the them agreed and 1 strongly agreed on the same statement when asked for interface 1. On the other hand, the difference between interface 1 and 2 came from the fact that 4 participants disagreed on the first statement(I think that I would like to use this system frequently) after performing a task on the mobile version but agreed after using the full version. Finally, when asked, 8 participants said that they might not use the system as they did not use Twitter.

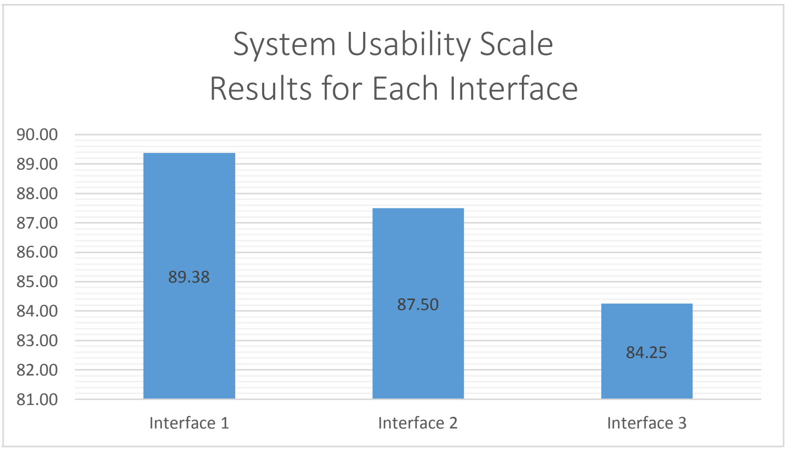


Figure 6.3: SUS Scale Mean Values per Interface

Results from Think Aloud

The Think Aloud was beneficial in two ways. First, it helped proving that the changes to the interface, performed after the initial phase of the evaluation, were successful as none of the participants had any comments on the new separation of components. Second, it helped identifying potential usability issues as well as comparing the different interfaces.

Also, while performing tasks on interface 1, 4 participants said that the date input was not clear. The reason was that there were arrows and clearly written Day, Month and Year on the buttons but nothing was leading them to the conclusion that this was the place where they had to input a date. 3 participants were confused by the tweets and their text as they were not used to Twitter and the notion of multiple hash-tags. Additionally, when performing task 2, 5 participants made an error while inputting the venue in the search input box and 2 of them did not even notice the error message, returned from the system. Also, 6 participants were misled and thought that entering the venue in the location input box would lead them to the required place and did not take into account any chance for a difference in coordinates.

Furthermore, while using interface 2 and performing task 2, 6 participants said that they had to scroll too much in order to reach the target and 2 of them suggested squeezing the timeline in case of too many tweets in a particular section and give the option to show more if required. 5 participants forgot which query corresponded to which timeline in the comparison screen due to the long scrolling and needed to go back to the top of the page. 3 people were confused from the Busy Venues events, which were displayed, and stated that there were no clues for showing the source of information. Furthermore, 4 participants felt it hard to find the place on the map as on mobile devices there is no hoover effect and only after pressing on the screen, they could have determine the venue that was selected. On the other hand, 6 participants liked the alignment of the timelines and the fact that they could be compared directly.

In addition, for interface 3, all of the participants directly noticed the lack of features and started looking for clues around the interface. They did not like the fact that there was no clear separation of the sections and that they had to follow the timestamp on each event in order to determine the timeslot. Finally, all did not notice the lack of a graph but when asked, they pointed out that it was not needed for the particular task.

Conclusion

The results from the two ANOVA tests revealed that there is no significant effect of the interface, on its own, on the performance of the participants. However, due to the small sample size(12 people), the outcome of the experiment can not be generalised. Contrastingly, the analysis of the Questionnaire and Think Aloud evaluations revealed that the participants tend to prefer Interface 1, which represents the full system, due to its extra features. Finally, these results achieved the goal of the evaluation and showed that the implemented features were of value to the users.

6.2 Testing

The functionality of the project was tested using white-box testing. This is a method that tests the internal components rather than the overall functionality. To design the tests, internal knowledge of the system was required. White-box testing can be used for both unit and integration tests.

Testing such application is a challenge. All the models talk to a service that is external to the system. However Laravel comes with built in Testing Component that allows for mocking objects and testing the views. Furthermore, for better coverage and reliability, both unit and integration tests were written to make sure that the application logic is consistent with the data, received from the services. Additionally, the interface was tested both manually and with unit tests.

6.2.1 Unit Tests

Testing Models

To prepare each test, mocked sample response for the tested model was extracted form an original service response. Some of the values were modified so the tests can target specific behaviour of the tested Model. For example, the BusyVenue Model was tested against a response, containing three time series but only one was indicating that the venue was busy. This test asserted that the returned list with events from this model would be of length 1 and contained the exact time series.

Testing Interaction

Additional to the manual interaction tests, unit tests were written during with the implementation to test both the navigation though the application and the validation of the input fields. This was time saving as there was no need for manually testing after a change had been done.

6.2.2 Integration Tests

Testing Models

The models were tested in the same way as in the unit tests but this time using the actual response from the services. This insured that there were no changes to the format of the responses. Additionally, this was very important as the services were still under development and provided a quick way to check if an error came from the services or from the application logic.

Testing Json APIs

Two RESTful APIs were implemented for this project but due to time constraints only one had been tested using integration tests. The Busy Venue API(reference to implementation) was tested against valid, wrong or missing GPS input. All the return messages were checked if they were appropriate depending on the given inputs.

Testing Summary

For the duration of this project, 26 tests with 84 assertions were written with overall test coverage of 32.5%.

6.2.3 Manual Testing

Both the initial and the final phases of the Evaluation process were used for performing manual testing. All the participants were given tasks to complete. As new users to the system, they were making mistakes. However, they did not manage to break the system but a few bugs were discovered. There were few cases when the timeline was not loading when there was an attempt to retrieve the venue's time series. Also, for a few participants, the two timelines in the comparison screen where not properly aligned but the problem was identified and fixed.

Chapter 7

Future work

Appendices

Bibliography

[1] Gregory D. Abowd Russell Beale Alan Dix, Janet Finlay. *HumanComputer Interaction, Third Edition*. 2004.

[2] Google Places API. `https://developers.google.com/places/javascript/`. Accessed: 2016-03-18.

[3] P. C. Cozby. *Methods in Behavioral Research: Tenth Edition.* 2009.

[4] Integrated Multimedia City Data. `http://ubdc.ac.uk/blog/2014/september/urban-life-captured-through-survey-sensors-and-multimedia/`. Accessed: 2016-03-05.

[5] Wayne W Eckerson. *Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications.*

[6] Cesare Pautasso Erik Wilde. *REST: From Research to Practice*. 2011.

[7] Stephen B. Hulley. *Designing Clinical Research.* 2007.

[8] JavaScript API Usage Limits. `https://developers.google.com/maps/documentation/javascript/usage`. Accessed: 2016-03-18.

[9] Ofcom. The communications market 2015.

[10] Barry Schwartz. Google maps timeline: User-friendly location history. `http://searchengineland.com/google-maps-timeline-user-friendly-location-history-225783`, July 2015. Accessed: 2016-03-06.

[11] Tim Storer and Jeremy Singer. *Lecture Notes on Software Engineering*. 2013.

[12] Ian Summerville. *Software Engineering, Ninth Edition*. Addison-Wesley, 2011.

[13] Google Maps Timeline. `https://www.google.co.uk/maps/timeline`. Accessed: 2016-03-02.