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

The rapid development of technology and growth of data centres have increased the amount of information about smart cities, which is being created and stored . As a result the number of opportunities for researching, analysing and tracking key events has also increased. We can find sources that give information about tweets, weather, trains and venues but not a single one, which is trying to combine all of them.

This project aims to build a tool that will be able to merge all the sources of timely data, which one smart city like Glasgow has to offer. This can be achieved by visualising them on a timeline, where they can be explored or compared. Furthermore, the final product has passed through a two stage evaluation process that tries to build a vision of how valuable such a system is to the users, as well as, how it can be improved in the future.

Overall, this is an innovative project that represents an idea, which we all as citizens can benefit from.

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 people's 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 Centre's 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 people's 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.

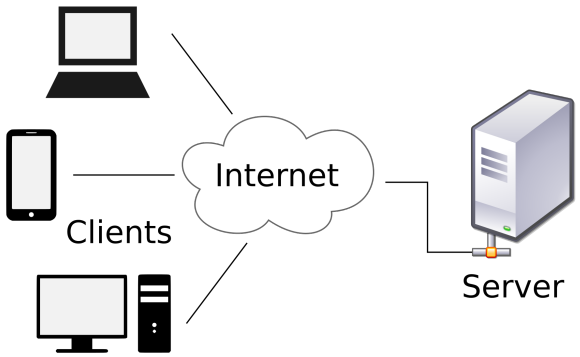


Figure 1.1: Client-Server Model

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 design 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 were 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.

Summary

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 hash-tag 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.

On the other hand, the next chapter describes how the project was planned and lists the identified use cases, requirements and challenges along with a high level diagram of the system.

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

Chapter 3

Project Planning

3.1 Software design approach

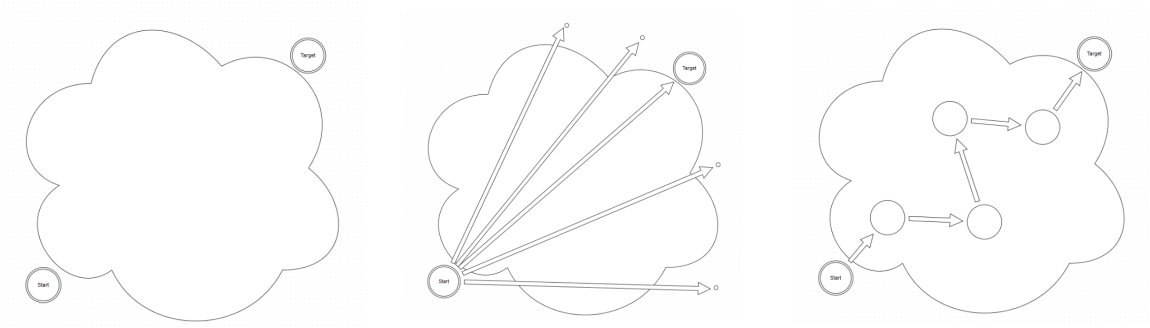
When a new product is developed, it is not clear how it will end up and if it will fulfil the user's requirements. The developers can see the first steps but there are plenty of problems or challenges that can not be predicted from the beginning (Figure 3.1a). That is true for any project, no matter 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 hash-tag are distributed over a period of time
- see popular hash-tags that are twitted together with a specific hash-tag
- 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 hash-tag
- 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.

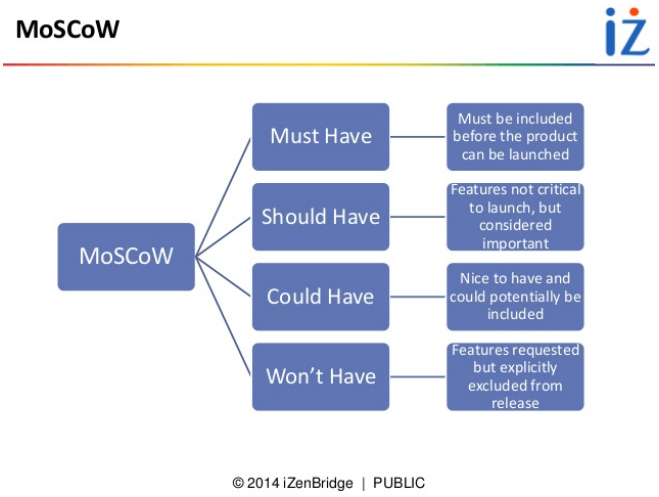


Figure 3.2: MoSCoW rules prioritisation scheme for requirements

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.

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

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

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].

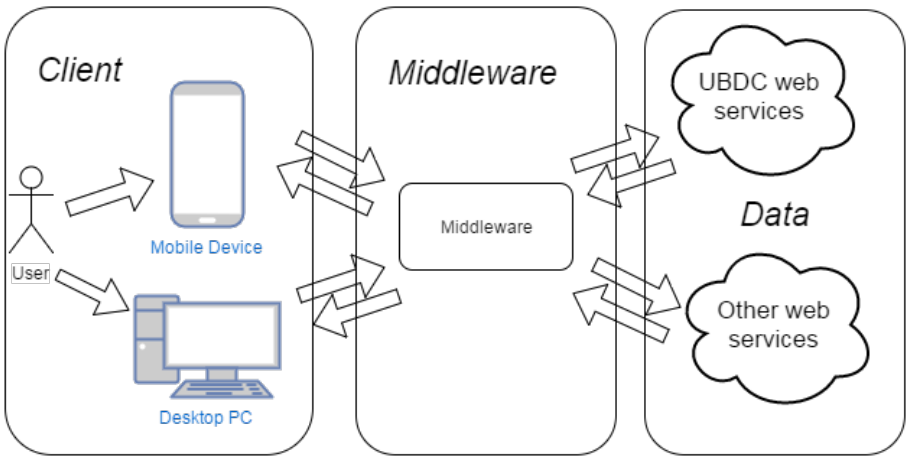


Figure 3.3: High level architecture diagram based on requirements and use cases.

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 hash-tags 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 user's 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.

3.3 Summary

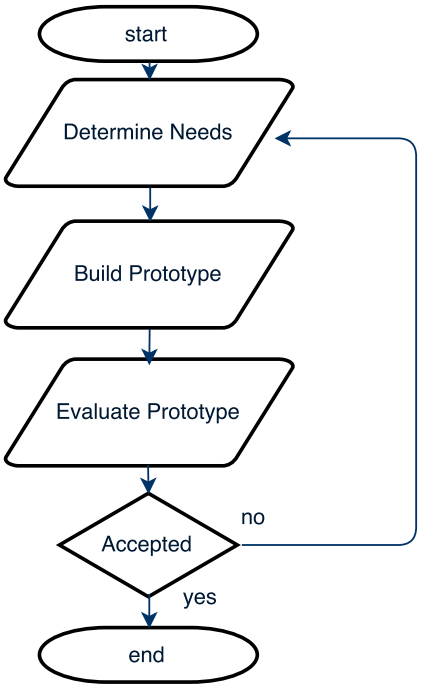
Finally, this chapter defines what design approach was used for the duration of the project. Furthermore, it lists the identified use cases and requirements, along with their representation as user stories. On the other hand, the next chapter goes though the design process and describes how the application was going to meet the requirements and outlines what was the chosen combination of technologies to solve the challenges, identified in this chapter.

Chapter 4

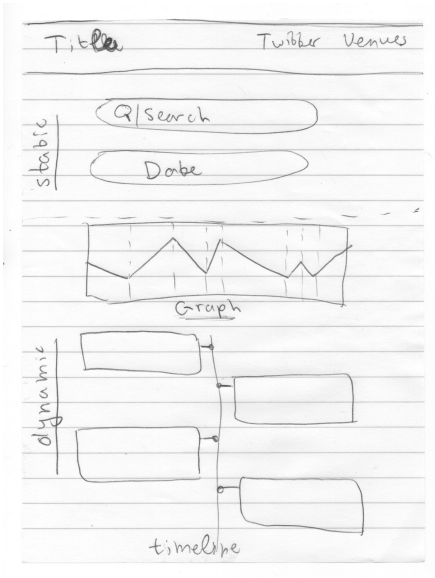
Design

4.1 Paper prototypes

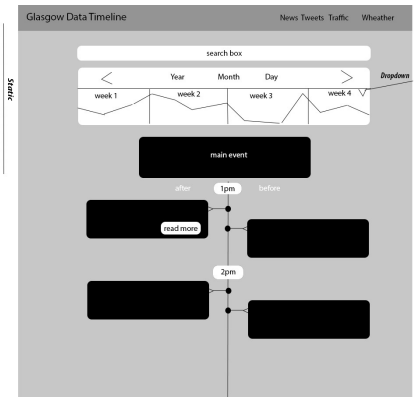
As shown in Figure 4.1a, designing an interface prototype is an iterative process that consists of four steps. The fist step is determining the needs of the users. This is already discussed in the previous chapter. The second step is building the actual prototype. It is good to start from sketches. Once 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.

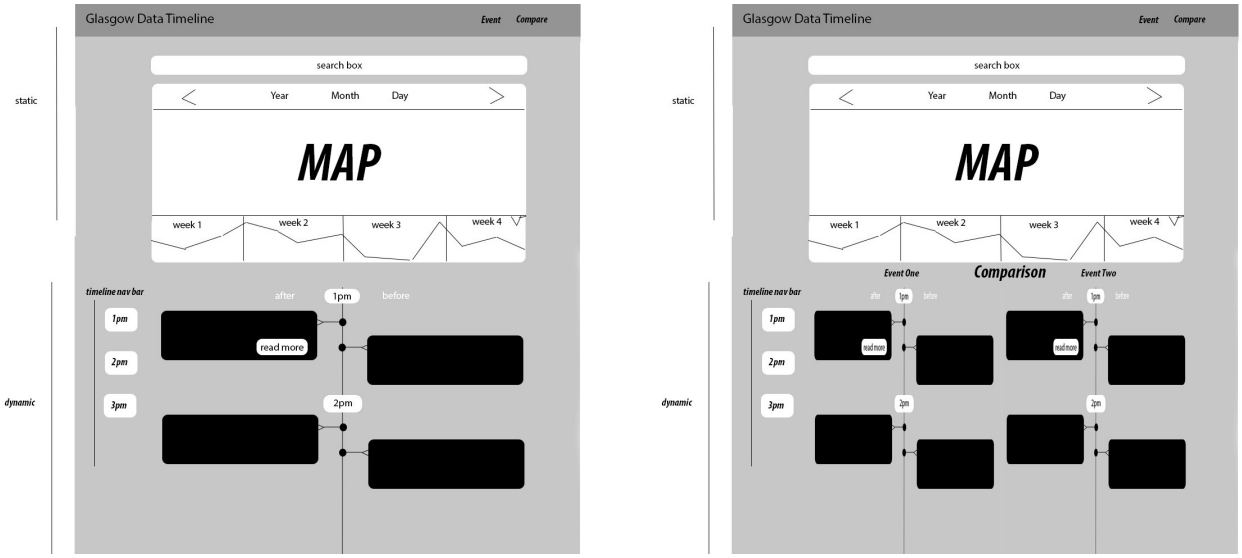


(c) Wireframe from the paper prototype

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 hash-tags. 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.

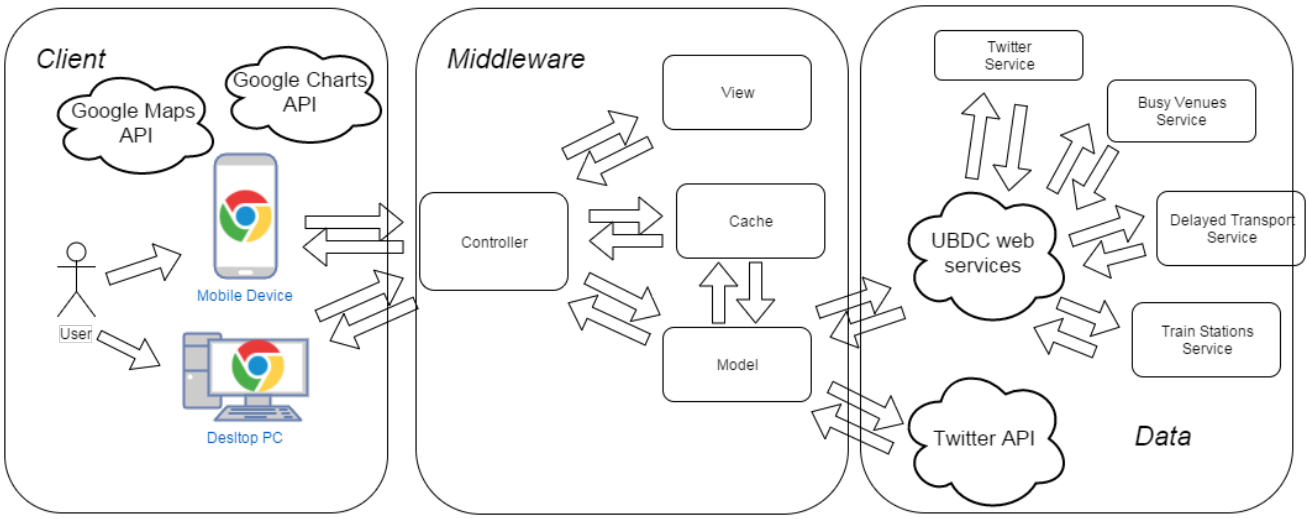


Figure 4.3: Component level architecture diagram.

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.

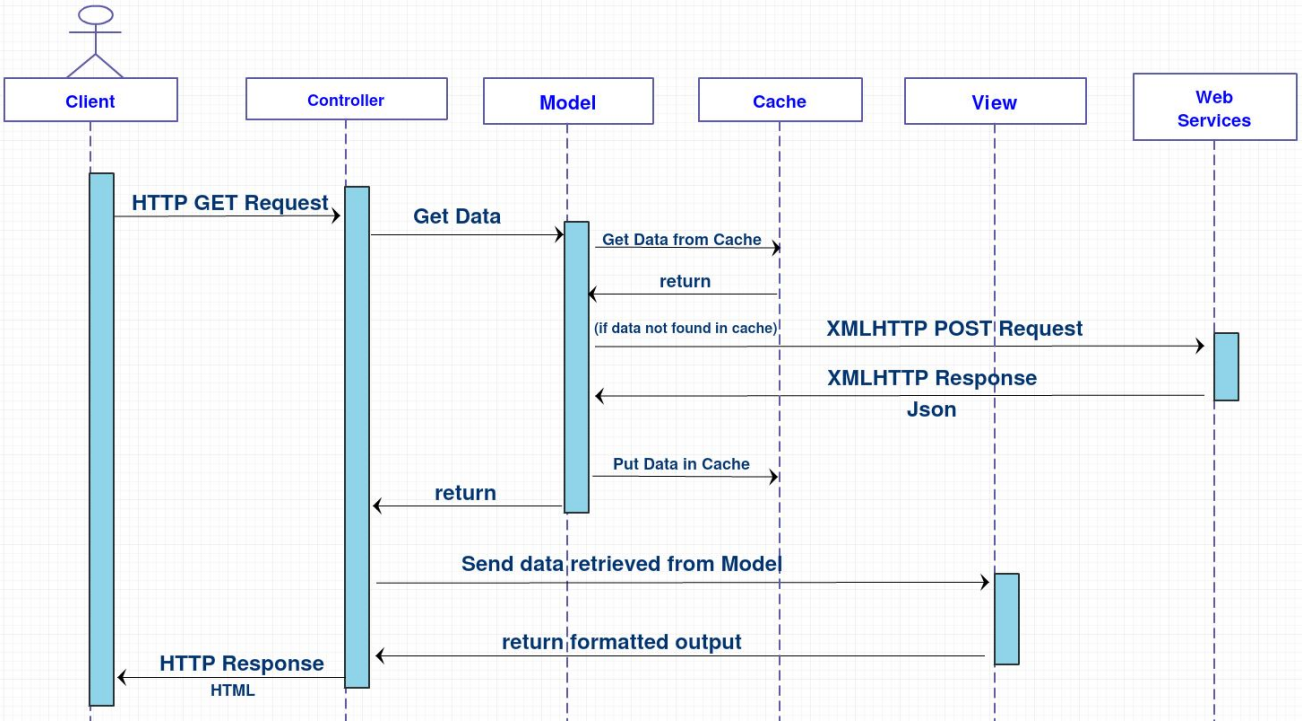


Figure 4.4: HTTP Request Sequence diagram

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.

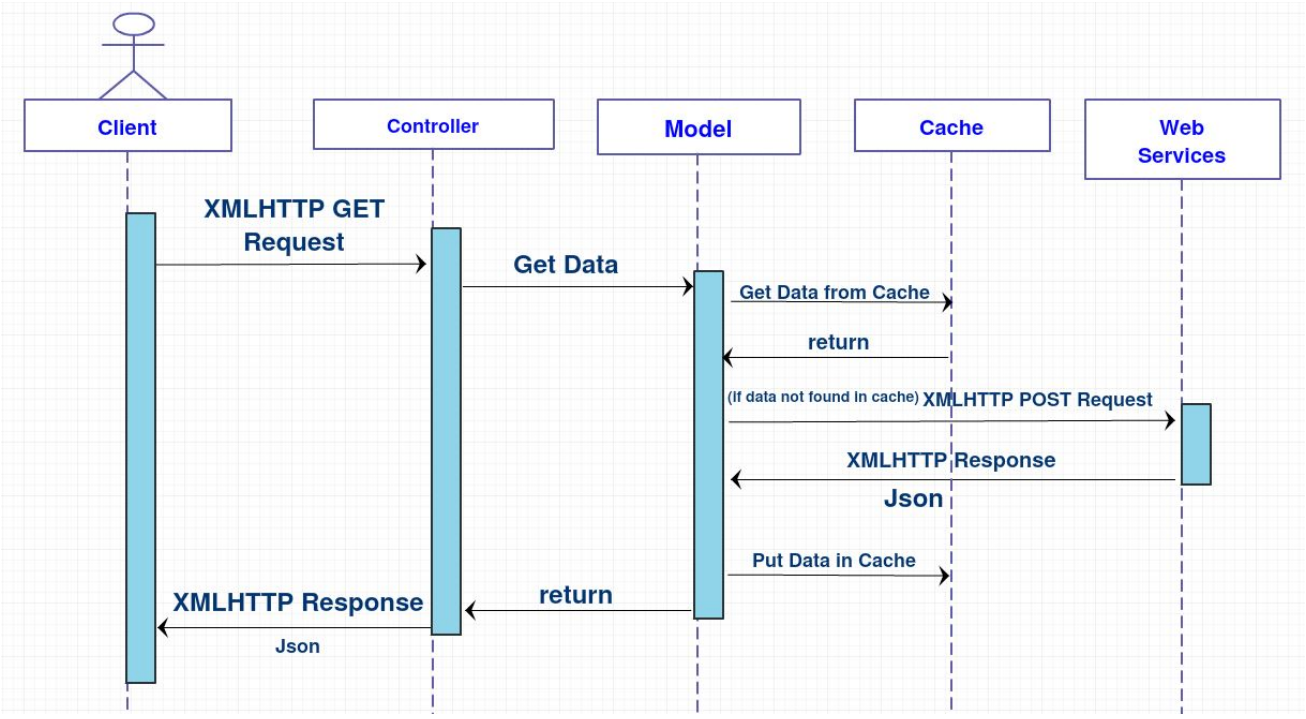


Figure 4.5: XMLHTTP Request Sequence diagram

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 in 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 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.
- easily make Ajax requests.
- make use of built in effects and animations.
- 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.
- perform Ajax requests.
- validate forms.
- 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.

4.5 Summary

Finally, this chapter outlines the details, needed before the start of the implementation. It shows how the system is supposed to look like, both in terms of the interface and the component structure. Furthermore, it lists the technologies that were used for the implementation. On the other hand, the next chapter highlights some of the main steps and challenges that were identified during the iterations, allocated for developing a prototype and transforming it to a final product.

[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 the required tweet hash-tag. 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.

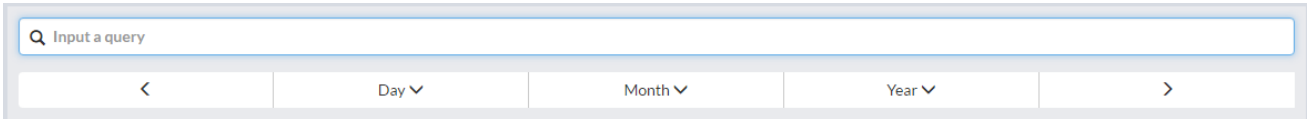


Figure 5.1: Search Box and Date picker

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(Section: 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.

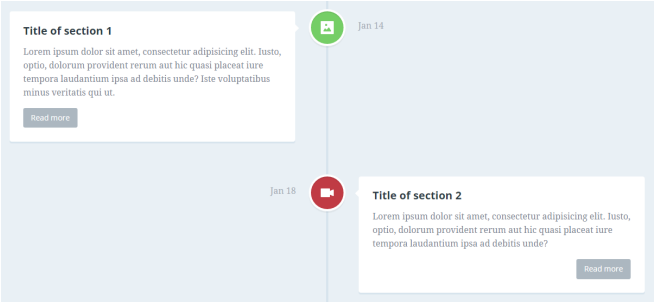


Figure 5.2: Initial timeline

5.2 Product Implementation

Accordingly, during the fifth iteration, the prototype was re-factored to make use of a Model-View-Controller framework, which was chosen in Section 4.4 so that communication with the services could be introduced.

5.2.1 Model

The models represented the data. They, as shown on the architecture diagram(Figure: 4.3), corresponded to an external service. At the start, in order to learn how the framework worked, each model was implemented separately. However, this led 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(Listing: 5.1).

Listing 5.1: Abstract Service class

```
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 method *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 was 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 the venue's 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 as shown on the architecture diagram(Figure: 4.3). 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 was able to be

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

integrated with Laravel and allowed for extracting a collection of the most recent Tweets, posted by specific user. It allowed for searching based on screen_name or user_id(Listing: 5.2). The only requirement for it to work was creating a Twitter account and enabling the API by generating the appropriate keys and tokens.

Listing 5.2: Getting tweets from twitter account

```
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 implementation. First of all, the Twitter service was not returning a time-stamp 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 implement 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. The Caching capabilities of Laravel were very flexible and allowed for directly inserting objects and specifying a time limit for how long each insertion stayed in the Cache(Listing: 5.3).

Listing 5.3: Caching the response from Twitter

```
    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;
    }
```

In addition, Caching was a part of the requirements(Table: 3.1). As it is shown on the architecture diagram(Figure: 4.3), it was included for all the Models as most of the users were expected to be researchers, who would not perform random queries but rather adjust some of the input parameters to extract the most out of the system. As a result, when a request was made, there would be no point in requesting data, needed for a previous search. Furthermore, this resulted in a reduced number of the requests to the external services.

5.2.2 View

The views represented an interface to present the data from the models as shown on the HTTP sequence diagram(Figure: 4.4). Initially, they were static pages. However, after migrating to the MVC pattern, using Laravel, the project benefited from the template engine, which the framework provided, called Blade[2].Two of the primary benefits of using Blade were template inheritance and sections. Since this project maintained the same general layout across all views, it was convenient to define it in a single Blade template, which was extended by all the pages(Listing: 5.4).

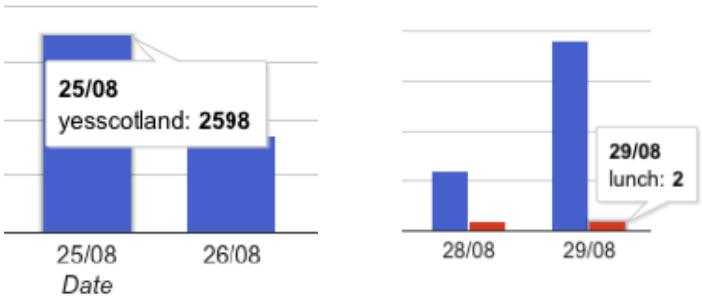
Listing 5.4: Master layout

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    @yield('title')
    @include('layout.style')
</head>
<body>
    @include('templates.partials.navigation')

    @yield('content')

    @include('layout.script')
</body>
</html>
```

This file contains typical HTML mark-up. However, it makes use of the @include and @yield directives. The @include directive defined a feature or component from the interface, while the @yield directive was used to display the contents of a given section, taken from a child page. This structure allowed for high flexibility as well as modularity.

One of the requirements was about visualising data on a graph(Table: 3.1). Accordingly, a chart view component was introduced. Initially, a bar chart was developed from scratch. However, due to time constraints, an already implemented solution was introduced, which offered a lot more extra features and ensured compatibility with different browsers. Google Charts was very flexible as it was adjusting the graph, based on the input that was given. For example, when the input was an array of arrays of length 2: *[['2014-08-21',10],['2014-08-22',20]..]*, a single bar for each data was presented(Figure: 5.3a). On the other hand, when the input was *[['2014-08-21',10,15],['2014-08-22',20,11]..]*, two bars for each date with different colours were shown (Figure: 5.3b). Furthermore, as the project was making use of data from different domains, each that offered some way to count events got its own chart representation. The one for the Twitter service was showing number of tweets for a given hash-tag. On the other hand, the Busy Venues service chart was presenting the number of busy venues in a specified area. The user could easily switch between charts by clicking on the buttons, labelled with the corresponding service name.

[2]https://laravel.com/docs/5.1/blade



(a) Single Chart

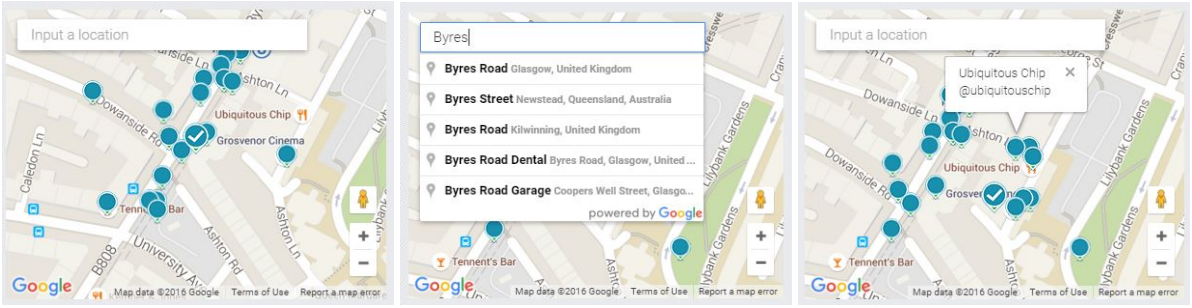
(b) Compare Chart

Figure 5.3: Charts

Additionally, taking location input was one of the requirements(Table: 3.1). In order to achieve that, a map component was introduced, which made use of Google Maps API(Figure: 5.4a). To allow the user to quickly find a place on the map, an input field was included that offered searching based on name and location(Figure: 5.4b). Furthermore, each marker contained information about the venue, which was representing(Figure: 5.4c). The map was populated with data, retrieved from the Controllers(Section: 5.2.3) with an Ajax request. On every change of the location, a new request was made to get the nearby venues(Listing: 5.5).

Listing 5.5: Getting nearby venues

```
google.maps.event.addListener(marker, 'dragend', function (evt) {
     map.setCenter(evt.latLng);
     lat.value = evt.latLng.lat();
     lng.value = evt.latLng.lng();
     getNearbyVenues();
});
```



(a) Map

(b) Map search

(c) Venue information

Figure 5.4: Map Component

With the extra functionality of the map, new possibilities for the user to browse different series of events were introduced. Google Maps API allowed for the markers on the map to be clicked. Part of the data, received from the Busy Venues service was about the venue's time-series. In addition, the Train Delays service was offering a time-series of train delays for a particular train station. Table 5.1 summarises the new alternatives, presented to the users based on their input. However, due to time constraints, train delays were not included in the views.

	Query	No Query
Location(click on the map)	Timeline of tweets with information about busy venues in radius of half a mile.	Timeline of busy venues around the area
Location(click on a venue)	Timeline of tweets for the query combined with tweets for this venue and hourly information about this venue	Venue time series including hourly information about the venue with tweets from the venue's twitter account
Location(click on train station)	Timeline of tweets for the query combined hourly information about train delays from this train station	Timeline of train delays

Table 5.1: Timeline content based on user input

In addition, one of the *Could have* requirements for the user interface(Table: 3.1) was to show a summary of the performed search, a box that pops up from the top and sticks there while scrolling was included as a component. The purpose of this box was to store key statistics and also to provide the user with other popular tweet hash-tags, based on the already provided input(Figure: 5.5).

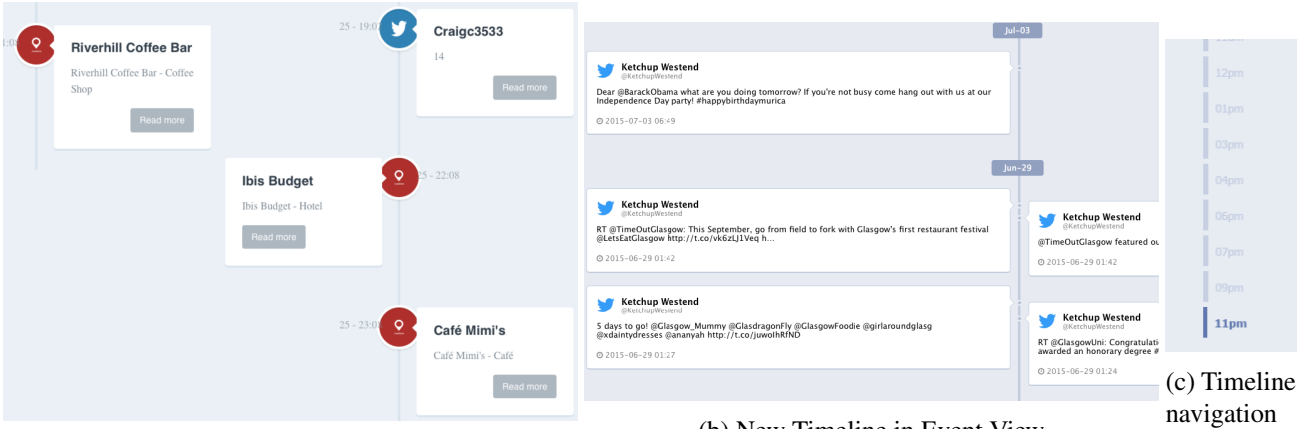
Hashtag	Date	Twitter Users	All Tweets for query	All Tweets for day	# Popular tags, tweeted together with #yesscotland
#yesscotland	2014-08-28	568	720	99925	#indyref[380] #voteyes[107] #bettertogether[93] #patronisingbtlady[74]

Figure 5.5: Information Box

Challenges

Originally, the design was made to accommodate only a single search. However, in order to meet one of the must have requirements for the user interface(Table: 3.1) and increase what the users could extract from the system, a compare screen was introduced so that two events could be compared side by side. At this point, all the components had their unique ids so just duplicating the logic for the single event search would break the system. To go around this, depending on how many search spaces were required, the controllers were implemented to generate an id for each one and passed it to the View so that each component could append it to its own unique id. For example a field with id *day* would change its id to *dayFirst*, *daySecond* and so on. The positives for such design were that it could be extended to support as many search spaces as wanted. However, the disadvantage was that they must have been predefined in the required Javascript files.

Additionally, when the changes were made and the timeline was visualised, it was clear that the open-source timeline, which was chosen earlier, did not scale properly(Figure: 5.6a). Furthermore, there was a lot of empty space so in a case of a really long sequence of events, the timeline would have been far too long. On way to solve this problem was to redesign it, but the code was very complicated and not easy to get on with. Finally, it was decided to implement a timeline from scratch so that it could be designed specificity for this project with all the desired features in mind(Figure: 5.6b). It displayed the events in sections, where each section represented a period of 1 hour. In addition, to reduce the amount of scrolling needed to reach a point of interest, a navigation bar for the timeline was included(Figure: 5.6c). It highlighted the timeslot, which the events on the screen corresponded to, as well as supported clicking that would automatically navigate to the desired section.



(a) Open Source Timeline in Comaparison View

(b) New Timeline in Event View

(c) Timeline navigation

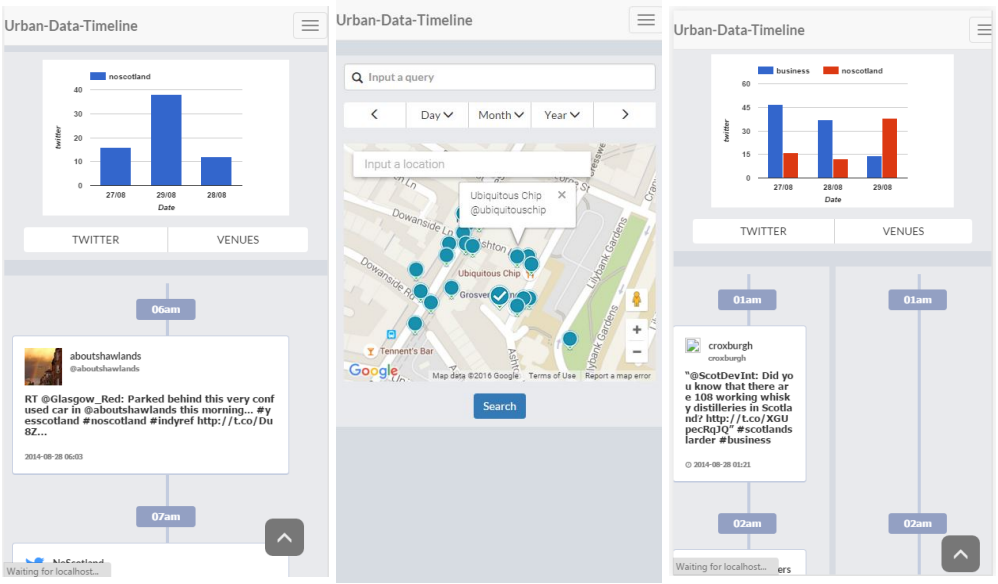
Figure 5.6: Timelines

Furthermore, support for mobile devices was a non-functional requirement(Table: 3.2) for this project. Initially, all the technologies (Section: 4.4), used were supporting mobile devices, meaning that they would scale properly even on small screen sizes. On the other hand, Bootstrap provided CSS classes that were implemented, using media queries[3], so all the components on the interface that made use of them will resize depending on the screen size of the device. However, the timeline, which was implemented from scratch, was not optimised and additional media queries needed to be developed. In order to emphasise the content of the events, the timeline was modified to show only one line of events(Figure: 5.6a), instead of two as shown in Figure 5.6b. Additionally, the navigation bar and the information box were made invisible for devices with width less than 900px(Listing: 5.6).

Listing 5.6: Media queries

```
@media only screen and (min-width: 300px) and (max-width: 900px) {
    .timeline ol.timeline_nav, #infoBox {
        display: none;
    }
}
```

[3] https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries



(a) Single event search (b) Inputs (c) Comparing two events

Figure 5.7: Mobile Version of the Application

To conclude, one of the most valuable feature, as well as, one of the biggest challenges when implementing the views was including infinite scrolling. A scenario when the views had to render a lot of events would increase the time, for which the user had to wait before starting to use the system. Infinite scrolling was the solution for this problem as it was made possible for a series of events to render on the page only after the user actually required them. The following code snippet shows a high level view of the implementation(Listing: 5.7).

Listing 5.7: Infinite scrolling

```
// on scroll
$(document).on('scroll', function () {
    $timeline_block.each(function (event) {
        // once the bottom of the screen is reached
        if ($(this).offset().top <= $(window).scrollTop() + $(window).height() * 0.9) {
            // get the id of the next section
            var sectionID = $(this).attr('id');
            // check if it is a new section, which has not been required yet
            if (prevSection !== sectionID && $.inArray(sectionID, listOfFilledSections) === -1) {
                listOfFilledSections.push(sectionID);
                // retrieve the events for the section
                ajaxCall(sectionID);
            }
            prevSection = sectionID;
        }
    });
});
```

5.2.3 Controller

The controllers served as operations that could be performed on the data. As it is shown on the architecture diagram(Figure: 4.3), they formed the connection between the client and the data. Every controller starts with validation. Depending on its purpose, every controller had different rules for validation. The rules for the Single Event Search Controller are show at Listing: 5.8:

Listing 5.8: Validation Rules

```
    'query' . $this->firstID => 'max:100|alpha_dash',
    'date' => 'required|date',
    'lat' => 'required|regex:/^(\-?\d+(\.\d+)?).\s*(\-?\d+(\.\d+)?)$/',
    'lng' => 'required|regex:/^(\-?\d+(\.\d+)?).\s*(\-?\d+(\.\d+)?)$/',
```

The date, latitude and longitude were required and must follow a specific format. If the validation passed, the controller got the data from the required models. Otherwise, a page was rendered by the View with error messages under the fields that were required or had wrong input.

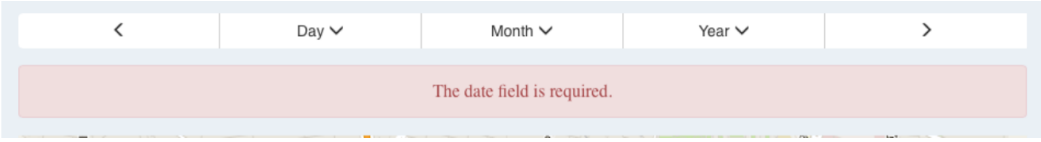


Figure 5.8: Error messages in the Views

There was one controller for each page. Initially every controller had the same mechanism for getting the data from the Model, which lead to a lot of duplicate code. This had to be refactored so that the implementation stuck to the DRY[4] principle and more controllers could be added more easily in the future. The common operations from the controllers were moved to a Helper class that took care of all data manipulations like sorting and section generating: splitting the events into separate sections, depending on their time-stamp.

In addition, controllers that were compatible with XMLHTTP requests(described in Section: 4.3.2) had to be developed to accommodate the needs of some of the View's components and meet the requirement(Table: 3.1) to be able to dynamically add data to the layout. They were implemented as RESTful APIs. Initially, the Chart component was requiring a list with the number of events for a specific time period. A CountController was implemented that made use of the getCount() method of each Model to retrieve data and return it in a Json format. For the duration of this project, the getCount() method was implemented only for the Twitter and Busy Venues models. The first one returned the number of tweets for a specific hash-tag and the second: the number of busy venues for a specific location. Table 5.2 shows the allowed input parameters for this API and Listing 5.9 demonstrates the response for a given url.

Listing 5.9: Count API Request and Response

```
Request URL
localhost:8001/rest/count?date=2014-08-25&query=business&lat=55.858&lng=-4.2590000000000146&range=3
Response:
{"business":
    {"twitter":[{"date":"2014-08-24","count":11},{"date":"2014-08-25","count":22},{"date↩
        ":"2014-08-26","count":40}],"venues":[{"date":"2014-08-24","count":14},{"date↩
        ":"2014-08-25","count":18},{"date":"2014-08-26","count":19}]}
}
```

[4]Do not repeat yourself

Field	Type	Optional	Default
date	string	no	
query	string	no	
lat	float	no	
lng	float	no	
range	int	yes	9

Table 5.2: Parameters available to the Count RESTful API

Additionally, the Map component was requiring the nearby venues for a given location. A NearbyVenues controller was developed that made use of the getVenuesNearBy() method of the BusyVenues model. This method was making a request to the services and keeping only relevant information about the venues from the response, including their location and twitter account. Table 5.3 shows the allowed parameters for this API and Listing 5.10 demonstrates how it works.

Listing 5.10: Busy Venues API Request and Response

```
Request URL
localhost:8001/rest/nearbyVenues?lat=55.874863914508&lng=-4.293143904860926&radius=0.01
Response:
{"status":"OK",
 "message":[{"name":"The Grosvenor Cafe","phone":"+44 845 166 6028","location":"31 Ashton Ln.","↩
     postalCode":"G12 8SJ","twitter":"the_grosvenor","lat":55.874805250556,"lng":-4.292950630188},{"↩
     name":"Ubiquitous Chip","phone":"+44 141 334 5007","location":"8-12 Ashton Ln.","postalCode":"↩
     G12 8SJ","twitter":"ubiquitouschip","lat":55.874863914508,"lng":-4.2931439048609},{"name":"↩
     Ashton Lane","phone":null,"location":"Ashton Ln","postalCode":null,"twitter":"","lat↩
     ":55.874865439609,"lng":-4.2929935455322},{"name":"The Wee Pub","phone":"+44 141 334 5007","↩
     location":"8-12 Ashton Ln","postalCode":"G12 8SJ","twitter":"","lat":55.874853401806,"lng↩
     ":-4.2929077148438}]
}
```

Field	Type	Optional	Default
lat	float	no	
lng	float	no	
radius	float	yes	0.05

Table 5.3: Parameters available to the BusyVenues RESTful API

Challenges

Initially, one of the challenges was with the ComparisonController and the Compare page. The problem was how two sequences of events could be compared side by side. Once the Helper class was implemented and the data was split into sections based on time, it became difficult to align the two arrays as they could differ. A simple algorithm was implemented that goes though each result and populates the other one with an empty section for the missing time slot so in the end they would both have the same number of sections and the corresponding functional requirement(Table: 3.1) could be met.

To conclude, one of the biggest challenges was implementing the backend support for the infinite scrolling. The logic behind it is simple. If the bottom of the page is reached, request the next N items from the database with id less than the last element's id in the View. However, this project, as shown in Figure 3.3, did not have a database. Furthermore, all of the services were not designed specifically for this project and it was not possible

to make a request for a specific data. On the other hand, if a new request was made every time, it would have increased the amount of traffic to the services and also the overall computation time required to process the data every time. The identified solution for this was once all the data was prepared, it was stored in a file system cache for a specific amount of time. It was still faster and more efficient but at the same time was not filling the server's memory, as well as, increasing the overhead of the project. In order to make this data available for the view, another RESTful API was designed. The logic behind it was to use the inputs and concatenate them together to produce an unique key, which was then used for extracting the data from the Cache(Listing: 5.11).

Listing 5.11: Using Cache for infinite scrolling

```
//Request URL: http://localhost:8002/infinite/single?queryFirst=noscotland&
//date=2014-08-29&lat=55.858&lng=-4.2590000000000146&_token=o0121trrqhGeBEv4F1fVqAcYLKinSJabgQ9GOjHQ↩
    &sectionID=03am
$requestParameters = array_values($request->all());
$id = array_pop($requestParameters);
//remove the crsf token
array_pop($requestParameters);
// form the key
foreach ($requestParameters as $value) {
    $cacheKey .= $value;
}
$fullResponse = Cache::get($cacheKey);
//return the part of the response, corresponding to the required id
```

5.3 Summary

Finally, this chapter shows how each of the components from the architecture diagram(Figure: 4.3) was developed. Section 5.2.1 goes thought the process of getting the data from the services. Section 5.2.2 explains how the data was rendered and section 5.2.3 describes the communication between the client and the middleware. However, the next chapter goes through the process of evaluating and testing the already developed system by outlining the taken steps.

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

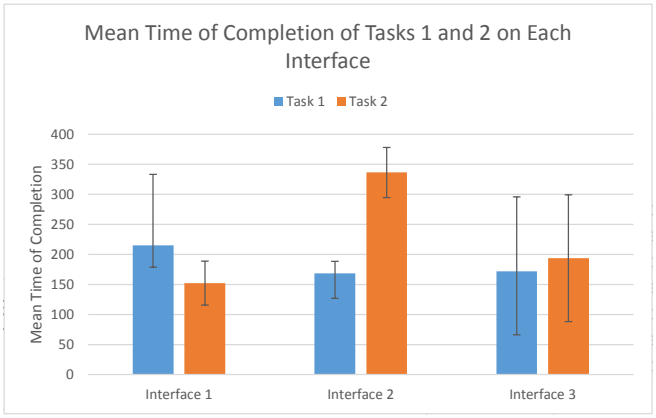
The first hypothesis suggested that there will be an interaction between the independent variables of task and interface on total completion time. The mean total time of completing tasks 1 and 2 on each of the three interfaces were obtained and are presented in Table 6.4 and illustrated in Figure 6.1a 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

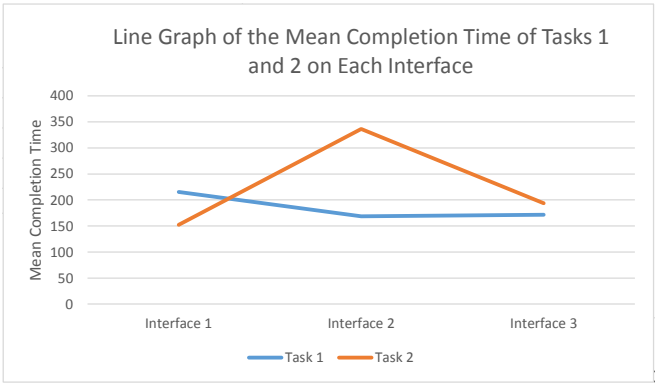
completed fastest on Interface 1 (M=152.19, SD=36.62) and slowest on Interface 2 (M=336,35, SD=41.73). The lowest total time of completing both tasks is on Interface 3 (365.28). When being plotted on a line graph, the obtained values suggest interaction (Figure 6.1b). Therefore, the descriptive statistics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

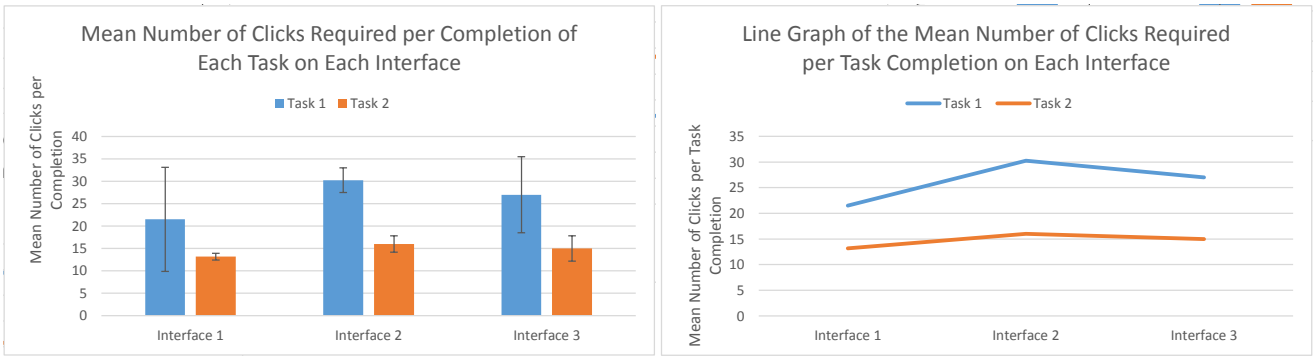
Figure 6.1: Results for Dependant Variable: Time

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 in 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.

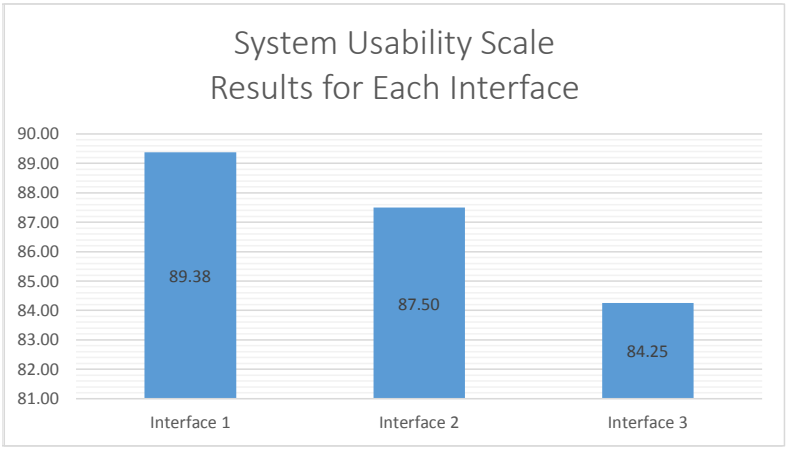


Figure 6.3: SUS Scale Mean Values per Interface

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 the 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 time-stamp 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 code 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.

6.3 Sonar

SonarQube[5] is an open platform that helps managing code quality. As such, it covers architecture & design, comments, duplications, unit tests, complexity, potential bugs, coding rules, comments and sources. It is compatible with PHP as well. The required configuration was made so the project could be analysed and a report can be extracted. Figure 6.4 demonstrates how such a report looked like. It clearly shows the test coverage, percentage of duplications and how many lines of code had been written. Furthermore, if clicked on any of the rows, a more detailed report was displayed.

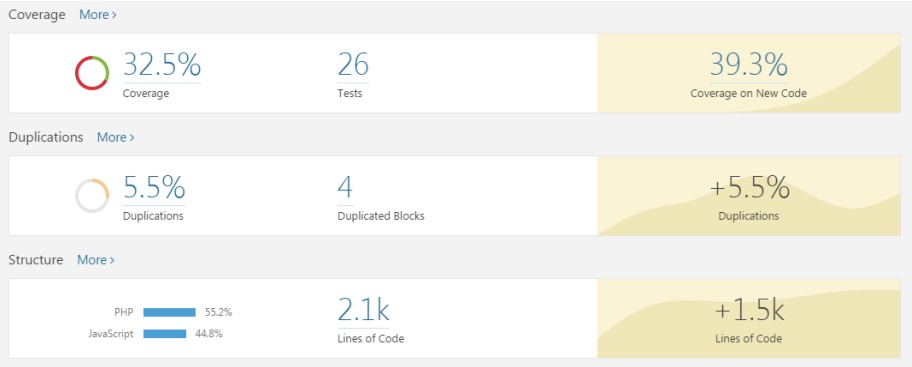


Figure 6.4: Sonar Report

6.4 Summary

To conclude, this chapter lists and explains the techniques, used for evaluating and testing the final product. Furthermore, it outlines possibilities for future improvements as well as identified problems. However, the next chapter expands on these findings and suggests possible solutions together with a summary of the whole project.

[5]http://www.sonarqube.org/

Chapter 7

Conclusion

7.1 Future Work

All the participants of the evaluation actively participated during the Think Aloud. As a result they played a significant role in identifying five possible improvements that can be used to improve the overall user experience. These are:

- **Scale**: The biggest issue, identified using the questionnaire, was that users might not use the system as they did not use Twitter. To try and increase the interest of this system, the best option is to add data from other social media portals like Facebook. Furthermore, the addition of data sources from different parts of the world might be beneficial as some of the events in Glasgow might be influenced, compared or contrasts by similar activities or trends worldwide
- **Comparison variables**: At this stage, the application allowed only for comparing between tweets, which correspond to two hash-tags. For the future, users like researchers may benefit from a feature, that allows for comparing different variables or data streams between different dates, hash-tags and locations.
- **Graphs**: During the final evaluation, some of the participants, who were studying sociology or business, suggested that they could benefit from the inclusion of different graph representations, such as Line graph or Scatter plots. These could provide a visual representation of the data, which will facilitate the analysis.
- **Suggestions**: Presenting suggestions, based on the previous searches of the users, could lead to easier understanding of the capabilities of the system and what data it has to offer.
- **Interface improvements**: The limitations of the user interface, identified in Section 6.1.3, should be addressed and most importantly an option, which hides most of the events and allows for expanding a section by the user, must be introduced so that the amount of scrolling can be drastically reduced.

7.2 Lessons Learnt

The biggest lesson for the author was how to drive alone such a relatively big project from the initial planning up to the final evaluation. Furthermore, PHP had to be learnt in parallel with the progress on the project. Additionally, the authors knowledge about the model-view-controller as well as how the frameworks made use of this design pattern had increased significantly.

7.3 Summary

This project aimed to build a tool for the fusion and visualisation of timely data. It went thought 23 iterations and 64 commits to the development branch in GitHub[1]. The final product was evaluated using 16 participants in total, who managed to prove that the application was usable by giving it a very high usability score of 89.38 out of 100(Figure: 6.3). Furthermore, the received verbal feedback confirmed that the implemented features were valuable to the users.

7.4 Acknowledgements

I would like to thank Dr. Iadh Ounis and Dr. Craig Macdonald for supervising this project and providing their very valuable feedback when required. Furthermore, many thanks to the Urban Big Data Centre for making this project possible by providing part of the data, collected for their Integrated Multimedia City Data project, and to Dr. Sean Moran, who was developing and supporting the services that were giving me access to the data. Finally, I would like to thank all of the participants, who took part in the evaluation, as their performance and feedback managed to prove that the final product was successful.

[1] https://github.com/skDn/Urban-Data-Timeline/tree/dev

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.

Appendices

Appendix A

Progress Reports

Progress Report 1

Here is what I've been working on

- Requirements
- User stories
- Design
- Provided services – checking what data is available
- Reading papers/researching
- Researching possible technologies to be used

Here is what I've achieved

Requirements

User Interface

- **Must** be able to display a fusion of different kinds of timely data (tweets, weather, traffic, train delays etc.).
- **Must** be extensible so that different things can be rendered on it.
- **Must** be accessible from desktops, laptops, tablets and smartphone devices.
- **Must** display events in a timeline manner.
- **Must** show the impact of an event on other events.
- **Must** show user navigation though events
- **Should** dynamically add data to the layout (using AJAX)

Server

- **Must** use algorithm for post processing the information it receives from various services.
- **Should** have fast data access – Caching.

User Stories

- As a Citizen, I want to know about car crashes, so that I know the effect that they have on the traffic in my area.
- As a Researcher, I want to know about the weather conditions, so that I can see if the weather conditions are really bad, will this affect the traffic jams and their duration. (Maybe people are scared to drive in bad weather conditions so that they take public transport)

Possible Technologies

UI: HTML5, jQuery, CSS3 (use it also in animations for better performance), Ajax

Server: Play Framework or Spring MVC (probably too big of an overhead) or PHP

Here are, I think, the next steps

- Research/try Terrier – it may be added as a separate service, as it is designed to use tweets natively.
- Finish my technology research and come up with pros and cons for each of them.
- Finish my research on various techniques of filtering an important event using the resources provided.

Here is what I want to talk about tomorrow

- Who is the user?
- Are the user stories accurate?
- What should we display to the user when he/she starts the application? Login?
    - Are we planning to use Real Time Data?
    - Is the data only about Glasgow?
    - Ideas: maybe get basic information about the user like location and show events based on that location and add search option to users that want to look at specific events.

Progress Report 2

Requirements:

Service

- **Must** extract key words from tweets combined with timestamp and use them in querying the other services
- **Should** support understanding of the query (For example: if user type how did the traffic suffer after a football match, the NOUNS – traffic and football must be extracted and used for searching)
- **Should** use multithreads for processing the data for faster responses.
- **Should** store pre-render events.

User interface

- **Must** be **universal** to support all kinds of users(citizen, scientist, local authorities)
- **Must** be able to display a fusion of different kinds of timely data (tweets, weather, traffic, train delays etc.).
- **Must** be extensible so that different things can be rendered on it.
- **Must** be accessible from desktops, laptops, tablets and smartphone devices.
- **Must** display events in a timeline manner.
- **Must** show the impact of an event on other events.
- **Must** show user navigation though events
- **Should** dynamically add data to the layout (using AJAX)

Milestones:

- Start with the most generic thing – what happened in Glasgow the past few months from twitter prospective.

Problems:

- In the Social media exploration documentation there is no documentation for “Exploring detected events in Twitter”.
- Also cannot run the visualisation examples

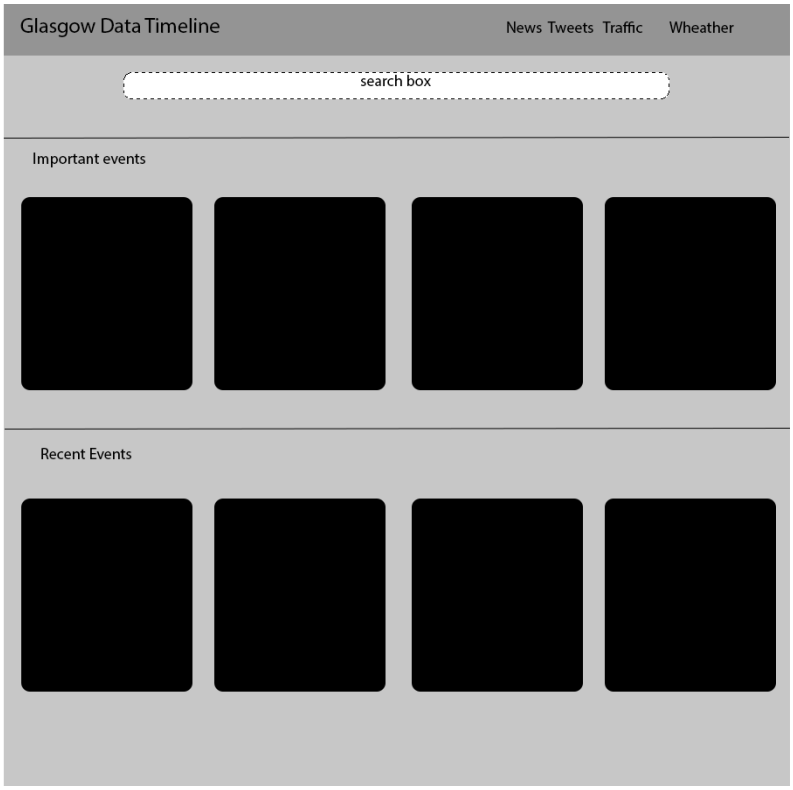
Proposal for a new service:

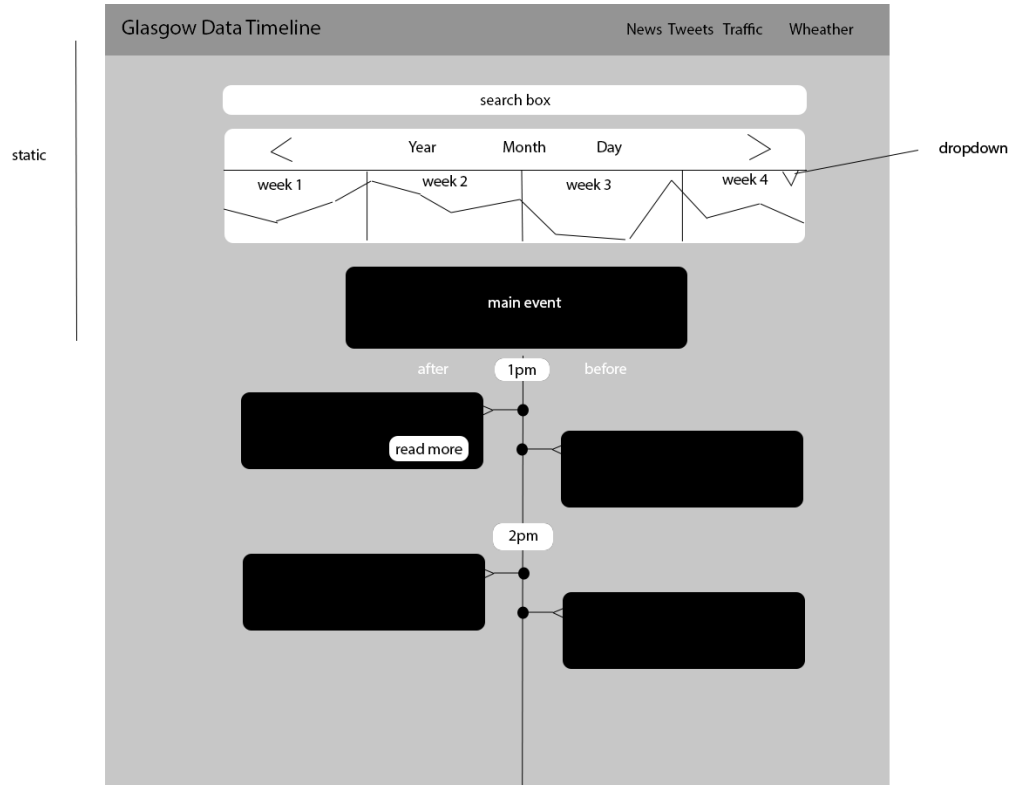
What about a service that is constantly looking for recent events and extract important or recent events and store them with a timeline of events connected to them? This can be used for pre-rendered events on the home page of the application. (BBC website style)

Here is what I want to talk about tomorrow

- What are the milestones?
- Did we manage to gather more use cases from the UBDC people?
- Talk about the design.
- Finish with the requirements

Design: based on other systems like Google Timeline





Progress Report 3

Here is what I've been working on

- Finalising the Requirements
- Choices of technologies
- Provided services – checking which service works by writing a simple php app that extracts data from every service

Here is what I've achieved

Functional Requirements

User Interface

- **Must** be **universal** to support all kinds of users(citizen, scientist, local authorities)
- **Must** be able to display a fusion of different kinds of timely data (tweets, weather, traffic, train delays etc.).
- **Must** display events in a timeline manner.
- **Must** show the impact of an event on other events.
- **Should** dynamically add data to the layout (using AJAX)

Server

- **Must** extract key properties of tweets(like Venue names, GPS, Timestamp) and use them in querying the other services
- **Should** support understanding of the query (For example: if user type how did the traffic suffer after a football match, the NOUNS – traffic and football must be extracted and used for searching)
- **Should** store pre-render events.

Non-Functional Requirements

- **Must** be extensible so that different things can be rendered on the views.
- **Must** be able to quickly process the data from the services.
- **Could** be accessible from desktops, laptops, tablets and smartphone devices.

Technologies

UI: HTML5, jQuery, CSS3 (use it also in animations for better performance), Ajax

Server: PHP, Laravel, Artisan, Composer

Pros:

- PHP has a very good built in JSON support
- Don't need MVC pattern which every framework enforces on
- PHP has C like socket support for low level communication through sockets
- Uses **weak types** – strict types are not really good for communication between services as require lots of extra work.
- Composer is a really good **dependency manager** that makes it easy to add, update or remove dependencies from the project.

- Artisan is a very powerful command line tool that helps you work with Laravel, it automates creation of controllers etc. It also provides a local dev environment.
- **Queue management** to abstract the unnecessary tasks behind and queue them behind so that the user can get faster response.

Cons:

- No multithreading support
- PHP applications tend to run slower

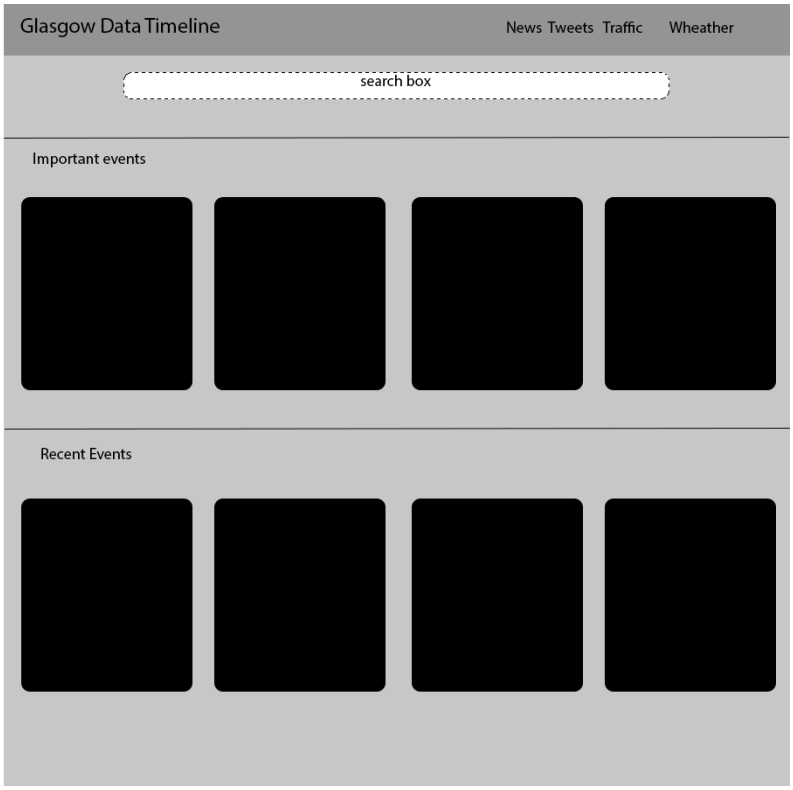
Here are, I think, the next steps

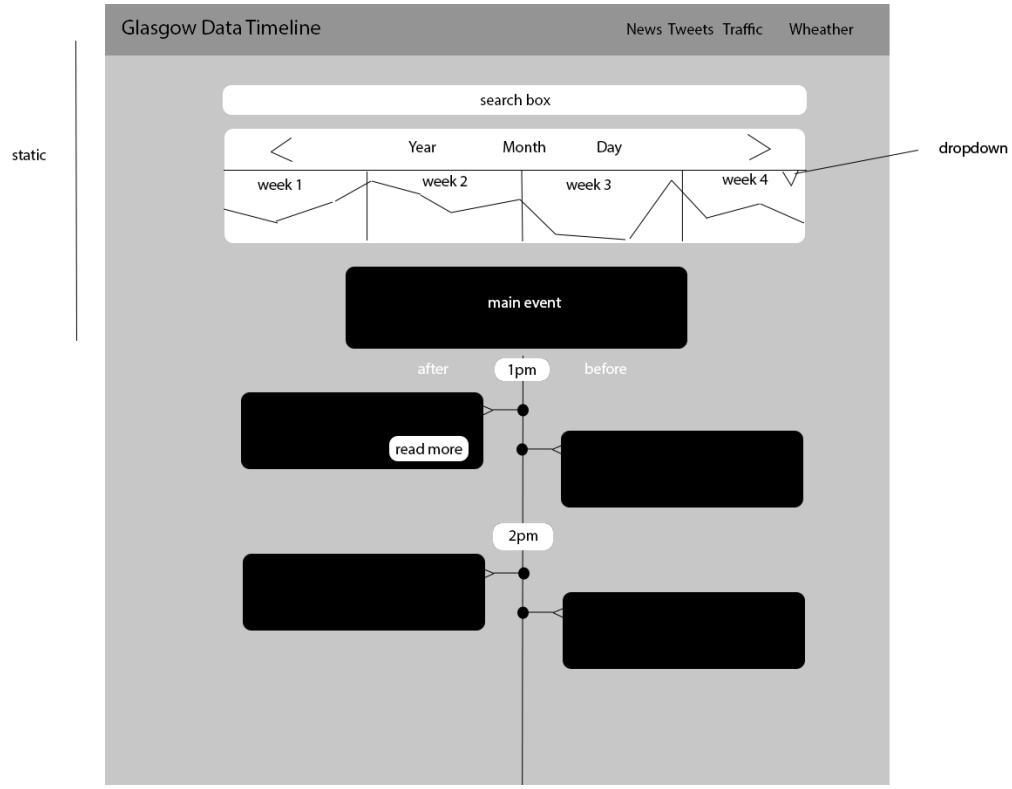
- Come up with **Milestones** for the project.
- Familiarise myself with the technology and code up a sample prototype

Here is what I want to talk about tomorrow

- Milestones for the project?!
- **Compare Java(Play Framework) to PHP**
- I would like to know more about the ***Event Detection as a Storm Topology,*** as I don't really understand it.
- Should we **validate** the service responses? (if yes we should add it to the requirements)
- Did we manage to gather more use cases from the UBDC people?
- Talk about the design.
- Talk about what methods we are going to use for extracting key events.
- Finalise the requirements

Design: based on other systems like Google Timeline





Progress Report 4

Here is what I've been working on

- Design
- PHP skills
- Research about Infinite Scrolling

Here is what I've achieved

Design

- Started working on the search page
- Designed a search input box using bootstrap
- Added an open-source timeline that will be modified or changed later if required

PHP Skills

- Started reading a PHP book
- Learned about some useful features in PHP
    - Generators – "helpful feature introduced in PHP 5.5.0. Unlike the iterator, it doesn't require you to implement the Iterator interface in a heavyweight class. Instead, generators compute and yield iteration values on-demand" Page 22. This can be useful when working with big Json objects as it will save a lot of **memory**.
    - Built-in HTTP server – perfect tool for local development. It is not good for production purposes though.
    - Components – "a component is a bundle of code that helps solve a specific problem in PHP app" page 52.
    - Standards - Components combined with Composer can be really useful as due to these standards all components for validation for example need to have the same interface so that components can be switched easily later. Good for a modular application.

Here are, I think, the next steps

- Researching for a good VALIDATION component for PHP
- Implement the work that I have done so fare using a MVC PHP framework like Laravel

Here is what I want to talk about tomorrow

- Are we going for lightweight interface and busy server or heavy interface and light server?
    - Different sorting approach (by Date/Time):
        - Sort on client
        - Sort on server
    - Different content loading approach(concept of Infinite Scrolling)
        - Load 100 on client
        - Load 50 on client and then request for another 50
- Is there any progress with the services?
    - "text\":\"java.nio.HeapByteBuffer[pos=0 lim=142 cap=154]\" on Twitter service
    - Event Detection Service?

Progress Report 5

Here is what I've been working on

- Integrating the prototype from last week into PHP MVC framework
- Designed the "Date Picker"
- Started working on the "Week Preview" of events

Here is what I've achieved

PHP MVC

- Implemented Controller and View
- Separated the View in components so that it can be easily extendable

Date Picker

- Decided to design my own date picker so that I can keep up with the clean design so far.
- Managed to implement the main functionality – changing date using directed arrows

Week Preview

- 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:
    - Showing value of the bar while hovering over

Here are, I think, the next steps

- Finish the Date Picker and integrate it to work with the searching.
- Finish the Week Preview and integrate it to display the number of results for each day.
- Implement separate Models for each service as now the controller is returning mock data.
- Maybe – spend more time on the separate components and implement them as plugins. So instead of just writing the whole code, I can just map a plugin to a and pass some configuration parameters.

Here is what I want to talk about tomorrow

- I would like to Demo the work I've done so far.
- Do you think I should implement every component of the design or should I use some already created components?

Progress Report 6

Here is what I've been working on

- Visualizing data using Google Charts
- Adding Compare screen
- Adding scroll-To-Top button

Here is what I've achieved

Visualization

- 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

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:

- 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.
- 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 doesn't scale properly when used in the comparison screen so I may have to look for alternative component.

Here are, I think, the next steps

- Finish my migration from support for one search space to two (or more) search spaces.
- Look for alternatives for the timeline visualization

Here is what I want to talk about tomorrow

- I would like to Demo the work I've done so far.

Progress Report 7

Here is what I've been working on

- Implementing services as Models
- Changing back to single date selector for comparison view
- Including chart representation for each service

Here is what I've achieved

Models

- I have separated the common features, that will help extracting data from each service
- Created an Abstract class that will contain all these features and each Model will extend it
- Fully implemented Busy Venues service
- Implemented random time generator to coupe with timestamp problems

Chart representation

- As the app is working with data from different domains, I introduced a new feature – each service gets its own chart representation.
- User can easily switch between charts by clicking on buttons with the service name, listed below the chart.

Overall improvements

- Using the abstract class mentioned above will make including of another service really easy for the developer.
- 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.
    - 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.
```


Problem

- Found out that tweets doesn't have timestamp
- Found out that venues, despite the fact that some venues are with score higher than 0, they don't seem to have any entries (all entries are set to 0). On the other hand, some venues with score 0 have entries greater than 0.

Here are, I think, the next steps

- I would like to introduce another feature – getting user location using two methods – first ask the user to allow the app to use his location and if permission denied, then use coordinates of service provider's closest router.

Here is what I want to talk about tomorrow

- Are we going to introduce any observable features like watching how many people access the website or from what locations and include a dashboard where these data can be visualized?

Progress Report 8

Here is what I've been working on

- Validation
- Adding location input

Here is what I've achieved

Validation

Research

- Javascript validation
    - Advantage – live update if a field is valid or not
    - 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
    - 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.
    - 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.

Location

- Integrated Google Maps API into the project.
- Placed a map under the date input.
- The user is asked for allowing the app to use his/her location on start of using the system. If permission is not given, the default location will be Glasgow default coordinates. The users is given an input box that accept inputs from post codes to name of venue or street. This input makes use of the Google Maps Places class so the search performed will be identical as searching the original Google Maps page.
- If the user searches and finds the desired place, he/she is redirected to that place and venues in a mile radius are displayed with suitable marker for each venue and its name that is shown when mouse over the corresponding marker.
- There is a CUSTOM marker that shows the current location input and can be easily distinguished between all other markers that appear on the map.
- User can also drag or click on the map to fine tuning the position of the marker.

Here are, I think, the next steps

- I would like to introduce work on trying to synchronise both timelines on the comparison screen. For example when one timeline has 20 entries and the other has 4, the one with 4 has to

be split accordingly as the user can compare for example at this time were there any events for both queries.

Here is what I want to talk about tomorrow

- I would like to demonstrate the work I have done this week and talk about the report that I have to submit on the 16th.

Progress Report 9

Here is what I've been working on

- Think about what user can be able to extract from the system by using the interface
- Working on a template for a new timeline
- Added info bar about the search results

Here is what I've achieved

Querying / extracting data

	Query	No query
Location (click on the map)	Timeline of tweets with information about busy venues in radius of 1 mile.	Error **OR** timeline of busy venues around the area
Location (click on venue)	Timeline of tweets for the query combined with tweets for this venue and hourly information about this venue	Venue time series – including hourly information about the venue
Train Station	---------- same ----------	Timeline of train delays
Delayed Services?		

Template for a new timeline

- Due to the new ideas/requirements, that was discussed last week, I decided to go and experiment with a new really basic timeline so that I can learn its infrastructure in a reasonable time so that I can build on top new features like including key time periods like start of working day or end of working day.

Info bar

- I decided that when the user scrolls down to check the timeline, it will be nice to have some relative information that summarise the conditions on that day.

Here are, I think, the next steps

- I would like to come up with a draft for the report by the middle of next week so you can take a look at it and discuss it on the next meeting.
- I would like to talk about any other features that we can possibly introduce to the system as I worked on a "weekly sprint" basis and I would like to spend time polishing the interface as I think this will take me a lot of time due to CSS overlapping that occurs on a few components.

Here is what I want to talk about tomorrow

- I would like to demonstrate the work I have done this week and discuss the proposed features that I'm describing above.

Progress Report 10

Here is what I've been working on

- Worked on the draft report
- Continued the development of the timeline that I showed last week
- Researched how to extracted tweets from a twitter account

Here is what I've achieved

Timeline

- Included feature events that take the whole width of the timeline
- Increased the size of the event icon and added suitable colours
- Included timeline bar that shows on which section of the timeline is the user currently

Tweets

- Researched for a suitable component that will do the communication with twitter
- `thujohn/twitter` is a php component that can be integrated with Laravel and allows to search for tweets by username.
- For the component to work, Twitter account was needed.

Integration

- Created a test View and Controller.
- The controller uses the PHP twitter component to fetch tweets for specific venue and extracts information about it with all the tweets.
- The view iterates through the data sent from the controller and populates both the timeline bar and the timeline.

Challenges

- For some reason I cannot ssh and connect to the services, but I used the documentation and found that Twitter account is one of the data parameters that are returned from the BusyVenues service.
- As I cannot test if the service will return only one venue if queried for 0 radius around the location, I wrote the test View and Controller, using one of the Twitter accounts listed in the documentation.

Here are, I think, the next steps

- I would like to try and integrate the timeline to work with all the services. As I cannot connect to the services, I will try and use the json responses included in the documentation.
- With the new timeline, changes to the styling of the whole application will be required.
- I would like to think about how I'm going to distinguish between different search (only venues + venue related tweets, venues + query specific tweets) in the backend.

Here is what I want to talk about tomorrow

- I would like to demonstrate the work I have done this week and discuss the draft progress report that I submitted.

Progress Report 11

Here is what I've been working on

- Finishing the draft report
- Improving the timeline
- Connecting twitter timeline with venue timeline
- Including Caching

Here is what I've achieved

Timeline

- Included option for displaying venues
- Improved the navigation bar to be clickable and included a scrolling effect. When a navigation button is clicked, the page is scrolled to the event corresponding to that button
- Improved how the navigation bar shows current event. Now when scrolling down, the first event from top to bottom is selected to be current event, not the first one coming from the bottom
- Created a switch that will choose between the old and the current timeline interface.

Tweets with venue timeline

- Changed how the controller sends the data to the view. As the view was changed to support sections (1pm, 2pm etc.), the data is sorted on the backend and represented as an array where each element has an ID and events that are part of the corresponding timeframe.
- Implemented an additional model for the communication with twitter.

Cache

- Twitter restricts the usage to 180 requests in 15 minutes and advises the developers to cache the responses locally. I decided to include caching now, as it will also increase the speed of the responses, especially when testing new features.

Challenges

- The busy venue service is returning data for 48 hours that increases the load on the view. Possible solution will be implementing "infinite scrolling" and limiting the data from the service for only 24 hours. Also change to the sorting of the data might be required as currently the timeline navigation has shortcuts for each hour.

Here are, I think, the next steps

- Moving each hardcoded values from the backend to a configuration file.
- Continue improving the styling.
- Also a bug was found - google maps coordinates differ from services coordinates - change to the markers is required as currently I use the locations provided by google.

Here is what I want to talk about tomorrow

- I would like to demonstrate the work I have done this week and discuss the draft progress report that I submitted.

Progress Report 12

Here is what I've been working on

- Front and Back end support for venue search
- Improving the venue search response

Here is what I've achieved

Front and Back end support for venue search

Front end

- Fixed the bug, mentioned in the previous report, that google maps coordinates of the venues doesn't match the actual coordinates provided by the services. The fix was introducing a REST API that will take lng and lat and radius (optional), query the services and return a JSON with info about nearby venues, containing their coordinates that are taken from the Google Maps API and highlighted as markers on the map.
- Introduced tooltips, modals and popups. When hovering over a field either a tooltip will appear, explaining what this field is supposed to do or a popup including useful hints for specific feature. A modal is like a popup but it is used to confirm an action e.g. if user clicks on venue, confirmation is needed so that we can avoid accidental clicks that will lead to redirects.

Back end

- After submitting a venue search, an extra token is added to the request that if present, will be acknowledged by the Controller and the required search will be performed.

Improving the venue search response

- Due to the service returning data for 48 hours that includes day before and day after the query date, I introduced a filter that removes any information that is not from the query date.
- Included checking of the returned parameters from the BusyVenue service as I found out that the service returns less data for some venues e.g. some doesn't have twitter account, other doesn't have contact information.

Challenges

- It was found out that the Busy Venue service is limited to 20 venues in a response. Limiting the query range was required as if the range is too big lots of venues are missed. Also a big range leads to invalid response, as Venues didn't match to their coordinates when trying to extract the venue's time series.

Here are, I think, the next steps

- The development was introduced only to the "Single Event Page" so one of the next steps will be adding the functionality to the "Comparison Page".
- Trying out an idea that I have, using Cache, to implement infinite scrolling if time allows.
- Working on the "info bar" feature by adding real data from the services.
- REFACTORING/CLEANING OUT THE CODE.

Here is what I want to talk about on the next meeting

- I would like to demonstrate the work I have done during the winter holidays and discuss how it might be improved.

Progress Report 13

Here is what I've been working on

- Navigation bar and information box
- Google maps, regarding the venue search

Here is what I've achieved

Navigation bar and information box

- Both of these features are fixed to the top so that they stay on the page the whole time and the user can easily navigate through the app or see overview of the search that was made.
- Information box shows only when the user has scrolled down past all the input boxes.
- Change to the CSS was required so that the timeline navigation lines up.

Google maps

- While doing a venue search, the venue service didn't return any values for the particular location. This was due to different number of digits in the lat and lng parameters for each venue. The problem was restricting the number of digits after the decimal point to 6 while some venues have 8 or 10. The venue service matches exact location so the restriction had to be removed in order for the venue search to work. However the fix was not tested due to venue service not working.

Here are, I think, the next steps

- Continue working on the "Comparison Page".
- Continue working on refactoring/cleaning out the code.
- Implementing infinite scrolling.

Here is what I want to talk about tomorrow

- I would like to demonstrate the work I have done.

Progress Report 14

Here is what I've been working on

- Comparison Screen (Back end and Front end work)
- Polishing the CSS
- Refactoring
- Included a simple home page

Here is what I've achieved

Comparison Screen (Back end and Front end work)

- The format of the data that is being send to the view for this screen needed to be changed as now our timeline is split into sections. The code for the single event search was used but had to be modified to work for two separate searches.
- Same for the Front end part: using the view for displaying data for single event but modifying it to support multiple events being rendered.
- **Challenge –** due to the fact that results for different events may differ in terms of time frames, I had to come up with an algorithm that goes though each result and populate the other result with empty time frame in order to have the same time sections(1pm, 2pm etc.) in both results. Otherwise the view will look out of order and nobody can make the comparison.

Polishing the CSS/styling

- Due to having two timelines right next to each other, visual space for rendering the data is limited. Long paragraphs were not displaying correctly and, via CSS, word splitter was introduced that when a line is too long it splits it so the whole text fits.
- Font and text properties were changed on common components like the navigation bar to match on all pages.
- In comparison view, via jQuery, all the sections align for easy comparison.

Refactoring

- In the comparison screen progress, stated above, it is clear that leaving the logic like this will lead to lots of code duplication. In order to avoid this anti-pattern, a helper class was introduced for the Back end that will get the data required from the services and will leave the Controllers only to format it.
- For the View, a partial view that represents a section was introduced (section – e.g. 1pm – these tweets, these venues, different section will be 2pm etc.) that is used both in the comparison and single event view. Now if support for additional service is introduced, only this section partial template will need to be changed.

Simple home page

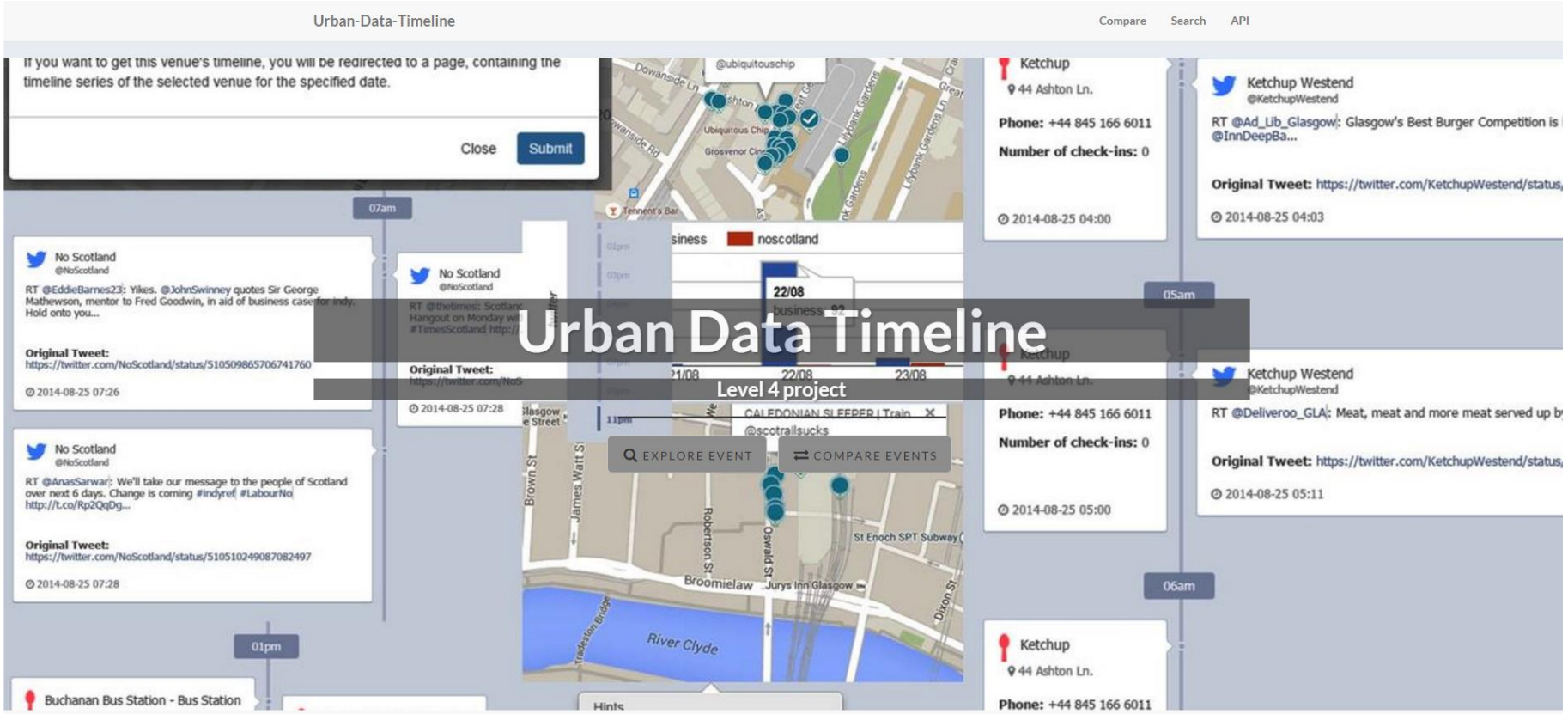
- Due to the fact that we want to run some user stories and for the actual evaluation, our app wouldn't be full without a home page. I spent a few hours to integrate it and I will add screenshots in the end of the report.

Here are, I think, the next steps

- I can spend a few hours more on the CSS and make it compatible with mobile devices and maybe evaluate if there is any need for mobile support.
- Continue improving the product and maybe finally introducing the infinite scrolling feature that was discussed.

Here is what I want to talk about tomorrow

- I'm still very sick and cannot speak so I won't be able to attend the meeting. Finally tomorrow I will be able to go and get my antibiotics and hopefully get better. If we want to run real user stories, it will be good to improve the services as well. Currently the tweet service doesn't return timestamp for a tweet and I'm using a random time generator on every tweet so that I can attach a timestamp to it that is not real but without it, it will be meaningless to display it on the timeline. Furthermore the tweet text is still not fixed - "text":"java.nio.HeapByteBuffer[pos=0 lim=142 cap=154]".



Urban Big Data Center

A special thanks to Urban Big Data Center for sharing their stored data for this project.

Urban Big Data Centre

Figure 1 Home Page

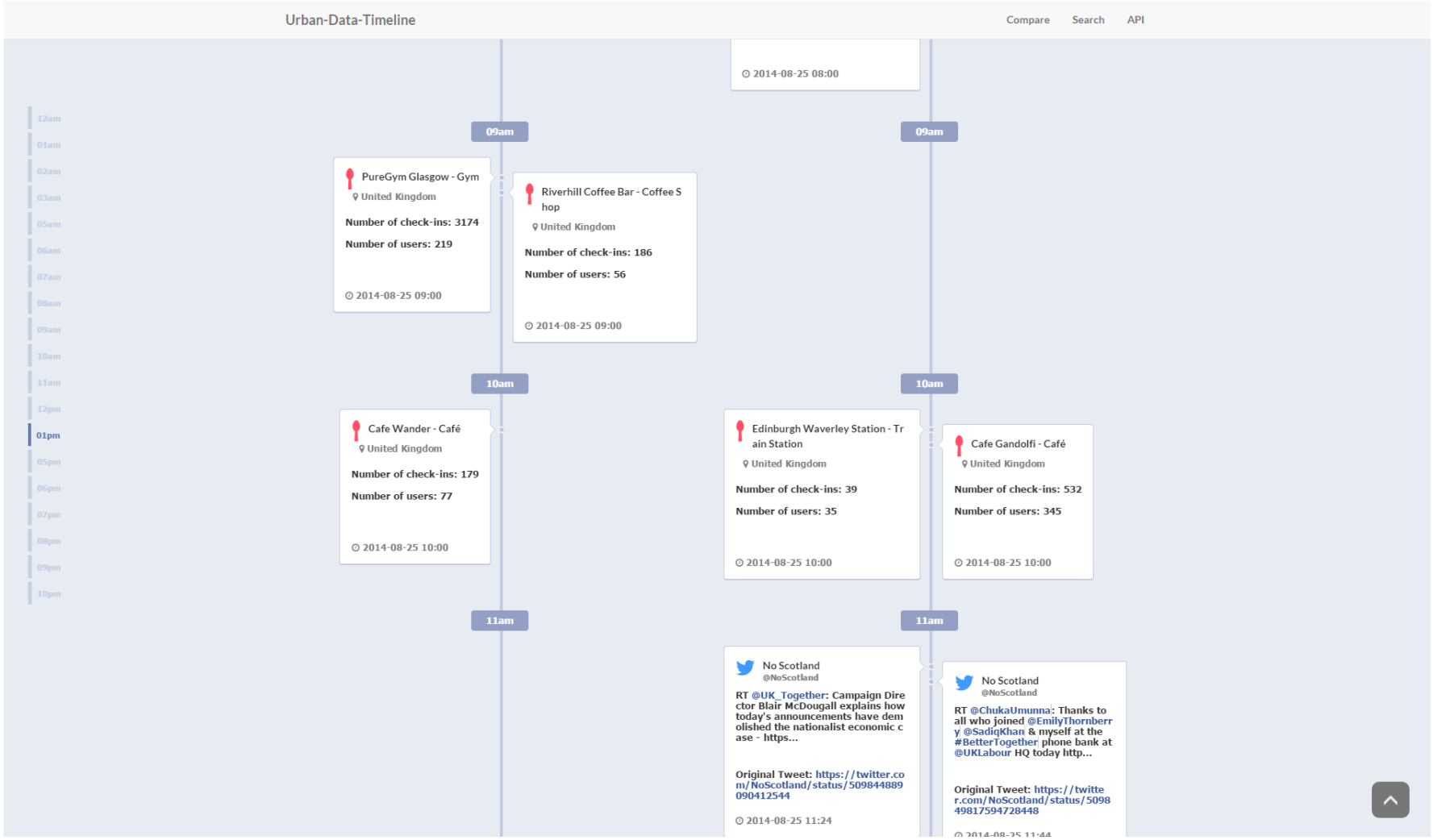


Figure 2 Comparison Page

Progress Report 15

Here is what I've been working on

- Support for mobile devices
- Improving the date picker
- Including additional services

Here is what I've achieved

Support for mobile devices

- My initial idea was to include support for mobile devices and maybe during the evaluation, include a test on mobile device to get a feel if it is reasonable to support such functionality.
- Changes to the CSS of the whole project was required. Bootstrap provides the functionality needed for making the improvement. Customising the CSS for screens with small resolution was required as there is no space to display the whole content and only key features should be available.

Date picker

- Date picker was displaying only a few dates that were hardcoded. jQuery was used to include all the possible dates and CSS class was introduced. This CSS class recuses the number of dates that the user can view but still gives the ability to select the desired date by scrolling up or down. This was designed with Mobile Device support in mind as the dropdown won't appear to be larger than most available screen sizes.

Additional Services

- I thought about the use cases and I came with one that required including another service.
    - Use case – As a user, I would like to know if there are more train delays, the number of tweets will also grow. One reason for a delay might be weather conditions and this might be a reason for a big number of tweets about bad weather.
    - Idea – as we take location as one of our key inputs, the back end will check if there are any train stations in half a mile radius. If there are, for each train station it will check for delays.
    - Challenge – the documentation of the delayed services doesn't reflect the actual response. There is no way of finding out about delayed services. The service just lists a pile of train service data.

Here are, I think, the next steps

- THINK ABOUT THE USE CASES.
- Think about what other functionality we should add to support more use cases.

Here is what I want to talk about tomorrow

- I would like to show you the progress over the past week and discuss different use cases.

Progress Report 16

Here is what I've been working on

- Caching
- Modifying the homepage
- InfoBox
- Use cases

Here is what I've achieved

Caching

- Caching the responses from the services so that users will get faster response in a case of slight modification of their query.
- Finishing this will allow implementing infinite scrolling in the next iteration.

Modifying the home page

- Darker background was added to the home page key message so that it can be easily read and available options can be picked.

Info Box

- Back end modification was required. Additional field to the data send to the view was introduced that includes key information, gathered from the services.
- In the front end, the view displays a table that adjusts its size depending to the number of elements displayed so it makes it flexible for adding additional fields.

Use cases

- 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 user's 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 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.

Here are, I think, the next steps

- Modifying the CSS so that popping elements are animated properly.
- Implementing infinite scrolling if time permits.
- Plan the evaluation.

Here is what I want to talk about tomorrow

- I would like to demonstrate the work I'd done and talk about the evaluation.

Services Feedback

- The weather service doesn't work or the mapping to it is wrong.
- The event service throws an exception when queried.

Progress Report 17

Here is what I've been working on

- Styling
- Sonar
- Use cases
- Evaluation

Here is what I've achieved

Styling

- I spend some time to improve the styling of how different features appear to the user.
- When a search is performed, the page automatically scrolls down to the point where the results are shown.
- When the results are shown, appropriate tools are made visible using a smooth animation.

Sonar

- SonarQube is an open platform to manage code quality. As such, it covers the 7 axes of code quality: architecture & design, comments, duplications, unit tests, complexity, potential bugs, coding rules, comments and sources. It works on PHP as well. I made the required configurations to analyze our project so a report can be extracted.

Use cases

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

Evaluation

- I think that evaluation should be quick (max 10 min) and should not take much time of the participants, as it will be really hard to gather participants for 20 min or more.
- My plan for the evaluation includes two questionnaires, tasks and maybe thinks aloud. The first questionnaire will gather data for the background of the user. Then the user is given a list with 3-5 tasks. After completing the tasks, another questionnaire (System Usability Scale). SUS has become an industry standard, with references in over 1300 articles and publications. The noted benefits are that it can be used on small sample sizes with reliable results and it can effectively differentiate between usable and unusable systems
- In the end depending on how the user executed the tasks, a think aloud evaluation can be performed.

Here are, I think, the next steps

- Plan the evaluation.
- Come up with a plan for the dissertation.
- Talk about what else should be done.

Here is what I want to talk about tomorrow

- Come up with a plan the evaluation.
- Talk about the dissertation.
- Talk about what else should be done.

Progress Report 18

Here is what I've been working on

- Testing
- Infinite Scrolling
- Evaluation – Initial evaluation
- Dissertation – table of content

Here is what I've achieved

Testing

- First I came up with a strategy to test each model and then how they interact with each other and communicate with the views.
- **Challenges:** PHP provides mocking of methods and fields. The problem is initially each method was designed to use queryService(parameters) method internally. When trying to mock the queryService method, it returns the expected result when tested. However, when a test is written for another method that use queryService internally, it doesn't work as expected.
- **Solution:** PHP provides a way to mock fields to specific values and ignoring constructors. Associating a separate local variable to the response from the service will reduce the number of calls that are made to the services and will allow testing all methods.
- **Conclusion:** one model was refactored and fully tested.

Infinite Scrolling

- Started implementing infinite scrolling. Javascript code that communicates with the backend is finished. Html structure for each result is still to be done.

Evaluation – initial stage

- As planned, the initial stage of the evaluation consisted of practical tasks, SUS questionnaire and think aloud. Evaluation was performed on 3 users (studying Sociology, Music and Computer Science)
- Tasks where completed correctly and the system go a SUS score of 75 out of 100. During the think aloud useful information about the application as a whole and possible use cases was received. The average time for completing the evaluation was 9 minutes and 27 seconds.
- **Challenge:** as testing, infinite scrolling and refactoring are taking place, the initial stage of the evaluation could not be performed.
- **Solution:** a new branch was created and changes had been reverted to the latest stable version and it was used for evaluation.

Dissertation – table of content

- Introduction
- Similar Products
- Architecture/Solution?
    - Choice of technology
    - Architecture/Design how components will communicate
- Components
    - Design of each component
    - Issues/challenges faced
- Evaluation
    - Initial phase
    - Actual phase
- Future work

Here are, I think, the next steps

- Consent form and Plain Language Statement (info sheet) need to be created before the actual evaluation.
- Perform the actual evaluation, continue writing tests, research more about other similar applications

Here is what I want to talk about tomorrow

I would like to discuss what I have done this week and see if you agree with my assumptions and decisions.

Tasks

- 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.
- Find how many tweets with the query word – noscotland are tweeted on the 27th of August 2014.
- Compare the number of tweets with query words – noscotland and yesscotland on the 29th of August 2014 at 2am.

Feedback

- Average score for the first question – I think that I would like to use this system frequently. – was 2 out of 5. When asked after the Evaluation, the participants said that they wouldn't use the app before they don't use twitter or they are not doing a research, as this app will be great for researchers. Also participants thought that the app is quite unique as they were not using anything similar in that time.
- **Bad:** buttons under the graph were not clear. Infobox and the timeline navigation were not noticed at first. Participants reasoned that as they were concentrated on completing the tasks and didn't look around.
- **Good:** Participants really liked the clearness of the design and the graphics. Also further use cases were suggested. One participant suggested that the app could be used in the tv/media shows to show peoples opinion about specific topic. Two participants told that during a political event, they could use the app to check how other people think which completely overlaps with one of our use cases.

Progress Report 19

Here is what I've been working on

- Testing
- Refactoring and Cleaning parts of the code
- Thinking about evaluation / refactored tasks
- Infinite Scrolling

Here is what I've achieved

Testing

- Tests for 2 more models were added to the test suite. All the models were refactored and can be added to the test suite in a future iteration.

Refactoring and cleaning the code

- The models were refactored to work with the new design.
- Comments were added to the abstract classes.
- Twitter model now returns text instead of user count. **Challenge:** there is still no timestamp or user name so a random time generator is used to allocate the tweets to specific section of the interface.

Evaluation

- Tasks were refactored as discussed in the previous meeting but due to the 3th task not being supported currently, I though about instead of running between subject experiment, to run a within subject one. Participants will do tasks 1 and 2 on both desktop and mobile screen sizes.
- I found example for consent form and information sheet so a future step will be to write one for our project. I'm planning to go to the Cyber Security Exercise in the weekend and perform the evaluation on some of the other participants.

Infinite Scrolling

- Infinite scrolling was implemented for the single event search. Also support was added for the venue search. It resulted in a faster page loads but the downside is that the animations are not supported.
- Infinite scrolling was implemented partially for the comparison event search. Initially it was designed to save the two requests that are made separately but it resulted in a massive amount of requests made to the server. After that the implementation was refactored and the two results were saved to the cache together that cut in half the number of requests to the server. Furthermore, the current overall implementation is not performing as fast as expected and there is still implementation to be done, as sections are not aligned as they were before.
- **Challenge –** javascript doesn't recognise elements being loaded dynamically. Tried a few workarounds and different ways of loading the data but I couldn't manage to include the animations.

Here are, I think, the next steps

- Coming up with consent and information forms.
- Performing the evaluation.
- Start writing dissertation. I would like to submit weekly drafts as well.

Here is what I want to talk about tomorrow

- Demonstrating the work I have done on the infinite scrolling.
- Finalize the evaluation experiment that I will perform.
- Discuss what are the next steps.

Progress Report 20

Here is what I've been working on

- Prepping the systems for evaluation
- Writing the dissertation

Here is what I've achieved

Prepping the systems for evaluation

- Comparison view in the full system was switched back so that it does not use infinite scrolling.
- The following features were removed from the third system:
    - Timeline navigation bar
    - Sections headings that show to what time the following events correspond to (1pm, 2pm etc.)
    - Graph
    - Infinite scrolling – the user will need to wait more for the pages to load.
    - Information box – the popup on the top of the screen, showing you a summary of your request.
- I implemented a piece of Javascript code that will run on the browser and keep track of user's clicks, time between each click and total time from the point where the interaction had begun. (*possibly add id of element clicked*)

Dissertation

- I downloaded the template from the Moodle page and changed the structure of the document corresponding to what we had decided on the previous meeting.
- I started writing the "Related Work" section. So far two related products were reviewed and compared – Google Maps Timeline and Social Media Timelines (Facebook and Twitter).

Here are, I think, the next steps

- Submit dissertation draft by the middle of next week. I would like to do this the following week as well. I will do my best to make a good progress with it over the weekend and in the beginning of next week.
- Perform the evaluation.
- Finish with testing.

Here is what I want to talk about tomorrow

- Do I need to submit consent forms, signed by all the participants?
- Will I receive any feedback on the report that I had submitted in December?
- What should be included in the video?

Progress Report 21

Here is what I've been working on

- Dissertation
- Evaluation

Here is what I've achieved

Dissertation

- Initial Introduction, Related Work and Project Planning chapters were written.

Evaluation

- Evaluation was performed with 6 people and it is arranged to be finished by Tuesday, next week.

Here are, I think, the next steps

- Finishing Design and Implementation chapters from the dissertation and submitting a draft by the middle of next week.
- Finishing the Evaluation and preparing the results for the next meeting.

Here is what I want to talk about tomorrow

- I would like to discuss the first draft that I had submitted on Wednesday.
- How should I send the video?
- What should it be in the presentation?

Progress Report 22

Here is what I've been working on

- Dissertation
- Presentation
- Video
- Evaluation

Here is what I've achieved

Dissertation

- Design chapter is almost finished and will be submitted for reviewing later on Friday or early Monday together with the Implementation chapter.

Presentation

- Draft of the presentation was prepared. Included screenshots of both mobile and desktop versions of the application.

Video

- Video was done as an advertisement. This was a time consuming and challenging process as syncing video and audio was hard. The final version shows all of the key functionalities of the app.
- All the third party videos and pictures are free to use.

Evaluation

- Evaluation was performed. The results are still to be analysed.

Here are, I think, the next steps

- Analyse the evaluation results
- Submit draft early next week.

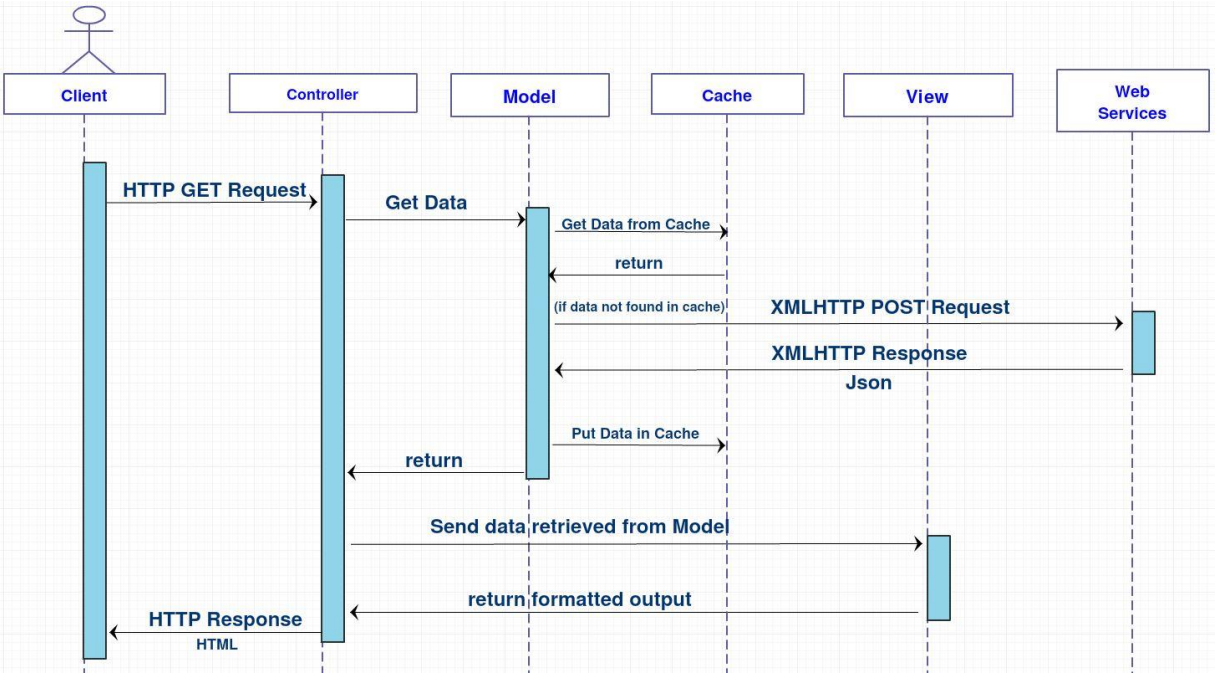
Here is what I want to talk about tomorrow

Presentation

- Should I tell how I solved the challenges in the presentation?
- Data sources – not sure how to name this slide?
- What should I say in conclusion?

Dissertation

- Should I say that I choose PHP as I didn't know it and it was additional challenge?
- Should I say in the sequence diagram that I use validator before the Model is called?


Client
Controller
Model
Cache
View
Web Services
HTTP GET Request
Get Data
Get Data from Cache
return
(if data not found in cache)
XMLHTTP POST Request
XMLHTTP Response
Json
Put Data in Cache
return
Send data retrieved from Model
return formatted output
HTTP Response
HTML

Appendix B

Semester 1 status report

Urban Data Timeline

Progress Report

by Yordan Yordanov

Project Description

The rapid development of technology and growth of data centres have increased the amount of information being created and stored. As a result the number of opportunities for researching, analysing and tracking key events has also increased. Following this trend this project aims to build a tool that will fuse and visualise timely data collected from the Glasgow Urban Big Data Centre (http://ubdc.ac.uk/). Past data from various urban big data streams, such as social media posts, news, blogs, traffic information, environmental sensors and many more has been provided. The tool should query all this data and come up with a timeline representation of observations that might be of interest to the user.

Progress

This project follows the Agile methodology for software development. It is an interactive approach that splits the work into a number of iterations (sprints). 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 project supervisors was received, based on the extent to which their requirements were met. Moreover, these meetings allow us to discuss issues that were experienced and compare the available workarounds. At the end of the meeting we agree upon exactly what work will be done during the next sprint. Following this scenario, at least one new feature was introduced after each sprint. By the time of writing, we have performed 9 iterations.

To start with, the first three iterations, during which, the main focus was on requirements, user stories and research of technologies to be used. Based on the user stories, list of functional requirements for the server and the user interface and non-functional requirements for the system as a whole was introduced. It was agreed that the user interface must be universal so that it supports all kinds of users (citizen, scientist and local authorities). Furthermore, it must display events in a timeline manner and must be able to visualize a fusion of different timely data (tweets, weather, busy venues traffic and train delays). The server must extract key properties from the provided services, pass them to the user interface and store pre-rendered events. The system, as a whole, must be extensible so that different data streams can be rendered on the view. Also it must quickly process the data from the services. Finalizing the requirements made it easier to design the infrastructure of the tool and choose appropriate technology for the implementation. HTML5, jQuery, CSS and Ajax are the technologies that we chose of the user interface. For the server side, Java and PHP were compared. In the end PHP was chosen due to it's built in JSON support and that it uses weak types (strict types are not a good choice for communication between services as they require lots of extra work). Finally, a paper prototype was introduced (See Figure 1).

Additionally, during the fourth iteration, the main focus was on implementing the paper prototype into a demo html page, learning PHP and researching one of the features that can be introduced at later stages - infinite scrolling. CSS framework - Bootstrap - was used for the implementation of the prototype. It provides styles and custom components that help to easily arrange and style a web page. Simple input fields (See Figure 2) combined with an open-source timeline (See Figure 5) were presented. The timeline was filled with mock data to simulate a real use scenario. The built-in HTTP server, which comes with PHP, was used to run and test the page. It is a perfect tool for local development. Along with that, the concepts of Components and Standards were introduced. A PHP component is a bundle of code that helps solve a specific

problem in a PHP application. Components can be really useful as all of them follow the PHP standards to have the same interface so that they can be switched easily.

Accordingly, during the fifth iteration, it was decided to move the prototype to a MVC framework so that communication with the services can be introduced. The communication could have been included with the prototype but it was decided to implement it during this iteration, as it will be easier to split the logic. Initial Controller and View were written using the Laravel MVC framework. They were divided into separate components so that they can be easily extended throughout the project. Two of the view components are the date picker (See Figure 2) and the week preview (See Figure 7). The date picker is used for inputting date, while the week preview is used for showing numeric results from the services (like number of tweets or busy venues) in a period of 9 days before and after the input date.

Furthermore, the sixth iteration three new features were added to the system. Initially the "week preview" was implemented from scratch but due to time constraints it wouldn't be possible to satisfy the requirements. D3js and Google Charts are few alternatives that were researched. Both provide enough options but Google Charts is more suitable for the data that we work with. The second feature was including a compare screen where two separate events can be presented at the same time so that they can be compared side by side (See Figures 3, 6, 8). Both queries share the same date and the same chart, where the common attributes can be compared as well. Scrolling down and viewing the timeline can lead to a long process of coming back to the search fields if the user wants to make a new query. In order to try and improve the user experience, scroll-to-top button was introduced that when pressed, it returns the user to the top of the page.

Following this pattern of development of the project, during the seventh iteration, each service was implemented as a Model. Common features from the Models were separated into an Abstract Class. Each model extends it and the abstract methods are implemented specifically. As a result, a new model can be easily integrated and accessed by a Controller. Creating separate Models means that more data will be sent to the View. The "week preview" was changed so that each service gets separate chart representation (See Figure 7).

Subsequently, during the eighth iteration, input validation was introduced. It was chosen out of two options – JavaScript and PHP validation. JavaScript validation has the advantage of providing live feedback to the user, however, it limits the error messages styling and positioning. I chose PHP validation as it returns the error messages which can be styled and displayed anywhere on the page (See Figure 10). The only drawback is that it requires a page reload. 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. We try to trigger the user's recognition with displaying the error messages fields using Bootstrap CSS in a red colour, so that the user can assume there is an error. Furthermore, during this sprint, another feature was also integrated - location input using Google Maps. A few of the services require location as one of its query parameters. User is given the opportunity to select location by pressing on a map, dragging a cursor or directly inputting a location (See Figure 4).

Finally, by the time of the ninth iteration, most of the features were already introduced, so we focused on what the user should be able to extract from the system by using the interface. We decided that the user might be interested in analysing timely data about separate services. Example for that is when there is no query word selected but the user has already pressed on a venue on the map. This shows interest in the venue only and the interface should display hourly information about the venue (See Table 1). Displaying more information requires changes into the current timeline that is complex and hard to learn. A new really basic timeline was implemented with perspective to be extended in the next iteration with features like indicating key time periods like start or end of working day (See Figure 9).

Plan

There are three major milestones left. Firstly, is finishing of the coding part, which includes integrating all the services to work with the current timeline, improving the styling and swapping the old timeline with the new one that is currently being developed. This should be done by the end of January. Secondly, comes the evaluation of the product. This includes planning, piloting the evaluation experiment and fixing bugs that may appear, recruiting participants and performing the actual evaluation. All of which should be done by the end of February. Thirdly, the final part is the dissertation, which should be done by the end of March. This comprises of a draft, which should be sent to the project supervisors at least two weeks ahead of the deadline.

Throughout the development of the project, there were a few issues that were faced. To start with, our first choice of a timeline didn't scale properly in the comparison screen (icons stay the same size and overlap the content that is displayed (See Figure 6)). We solved this by implementing a new timeline from a scratch and both timelines can to be used and compared during the evaluation of the product. Also, Tweets, provided by the services, didn't have timestamp. That is why we introduced a time generator, so that they can be displayed on the timeline. Moreover, the project itself appeared challenging. The compare screen was introduced in iteration six. However, by this time the view was implemented only for single query behaviour in mind. We had to come up with an approach that will separate both queries that are going to be compared and also a standard way of passing data from the Controllers to the View. As a future problem, styling all the components will be difficult and time consuming, due to the overlapping styles. Furthermore, finding a representative sample for the evaluation is problematic as well. Finally synchronising the two timelines in compare view will be a difficult task as post processing on the View will be required (See Figure 6).

Appendix

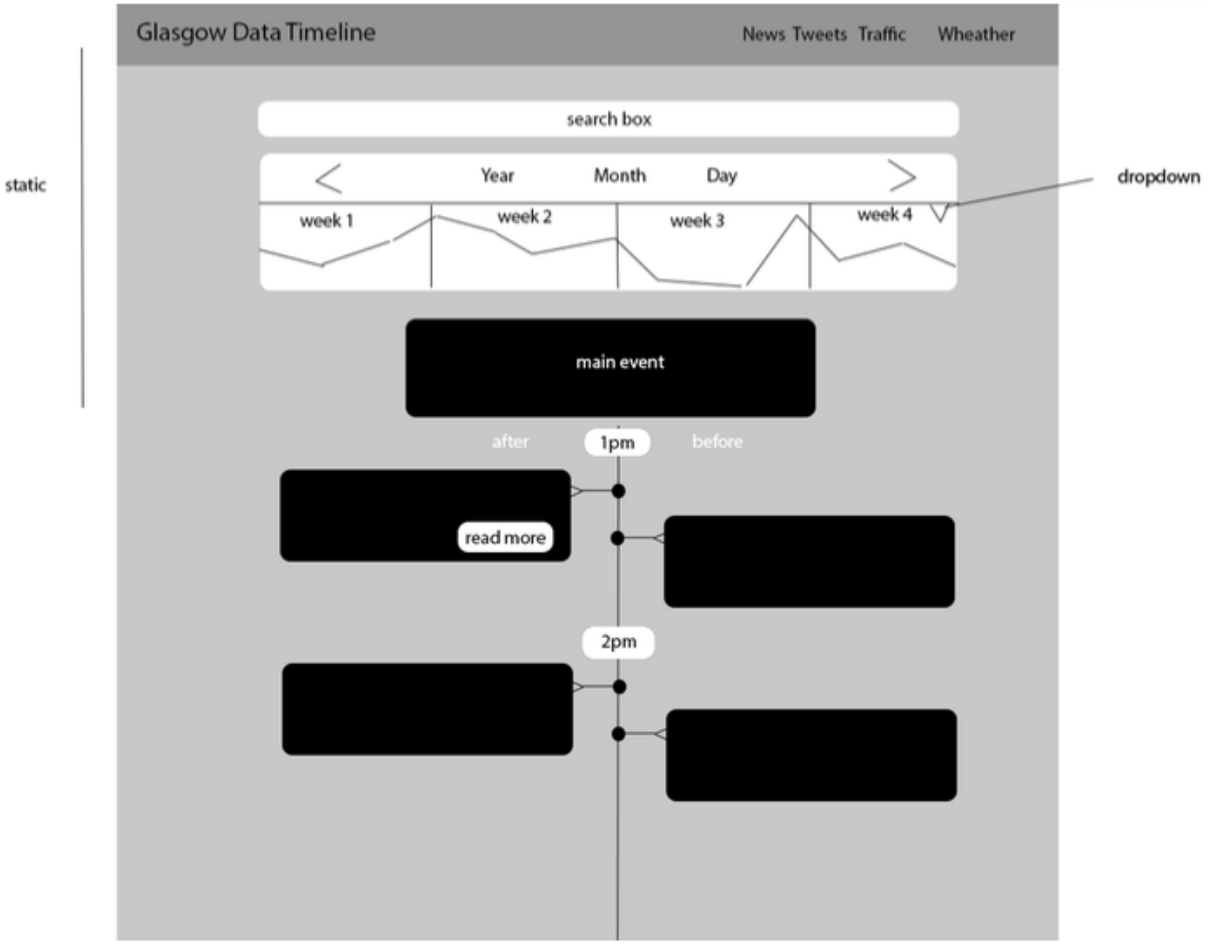


Figure 1: Paper prototype

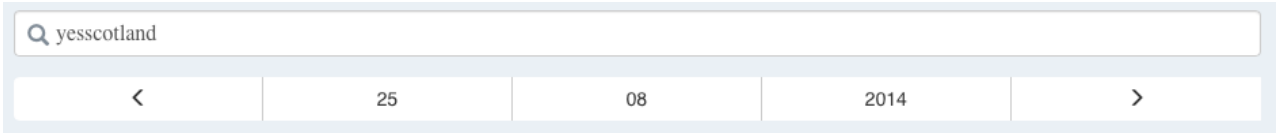


Figure 2: Input fields for querying a single event

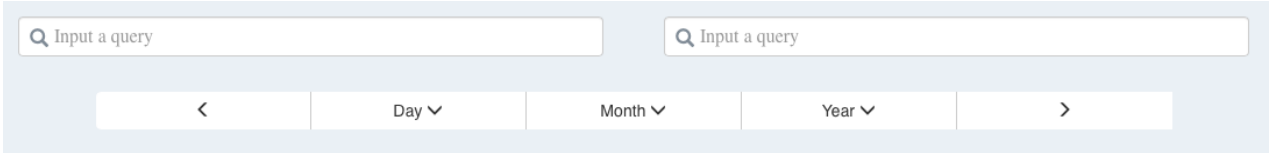


Figure 3: Input fields for comparing two events

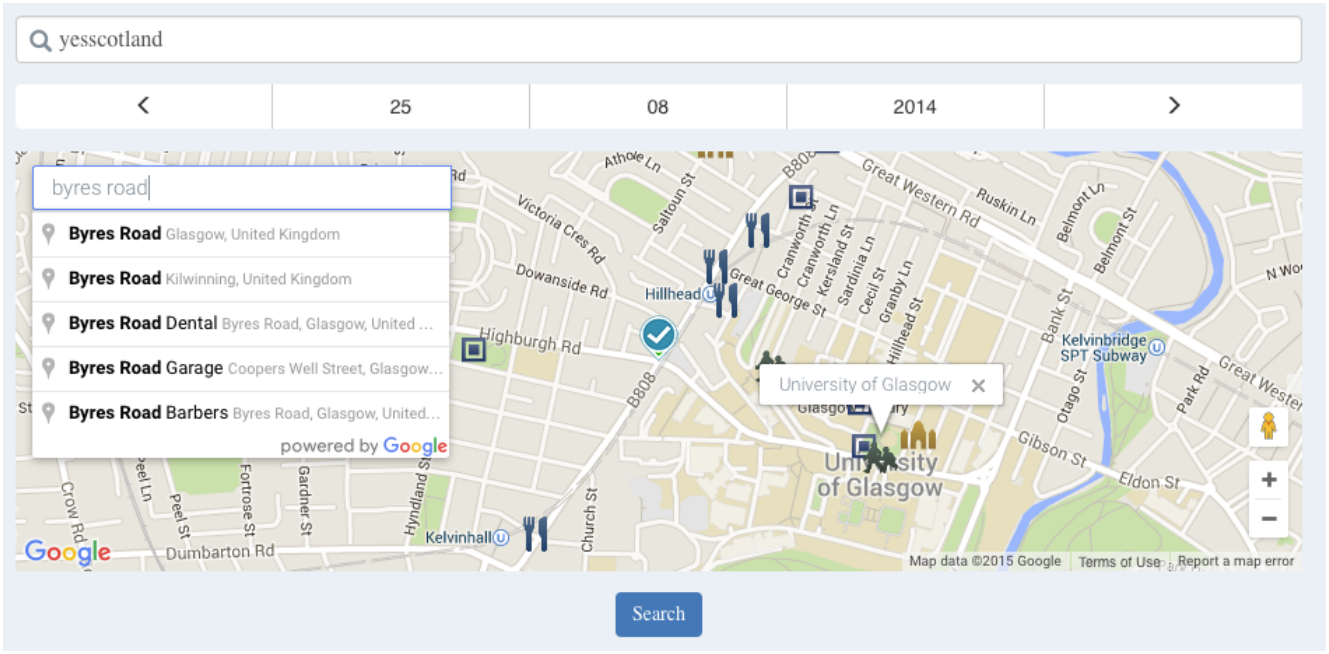


Figure 4: Location input

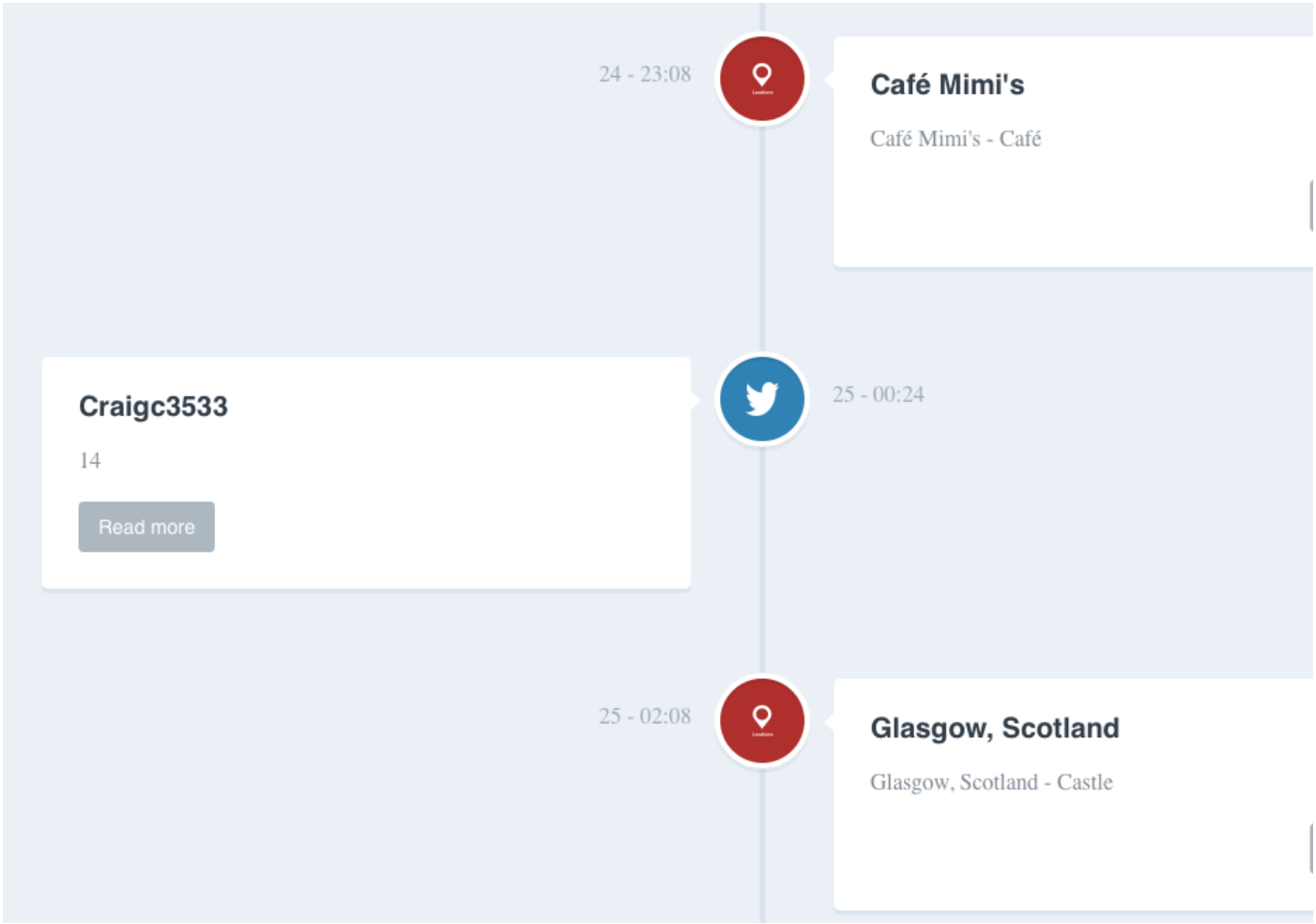


Figure 5: Timeline version 1

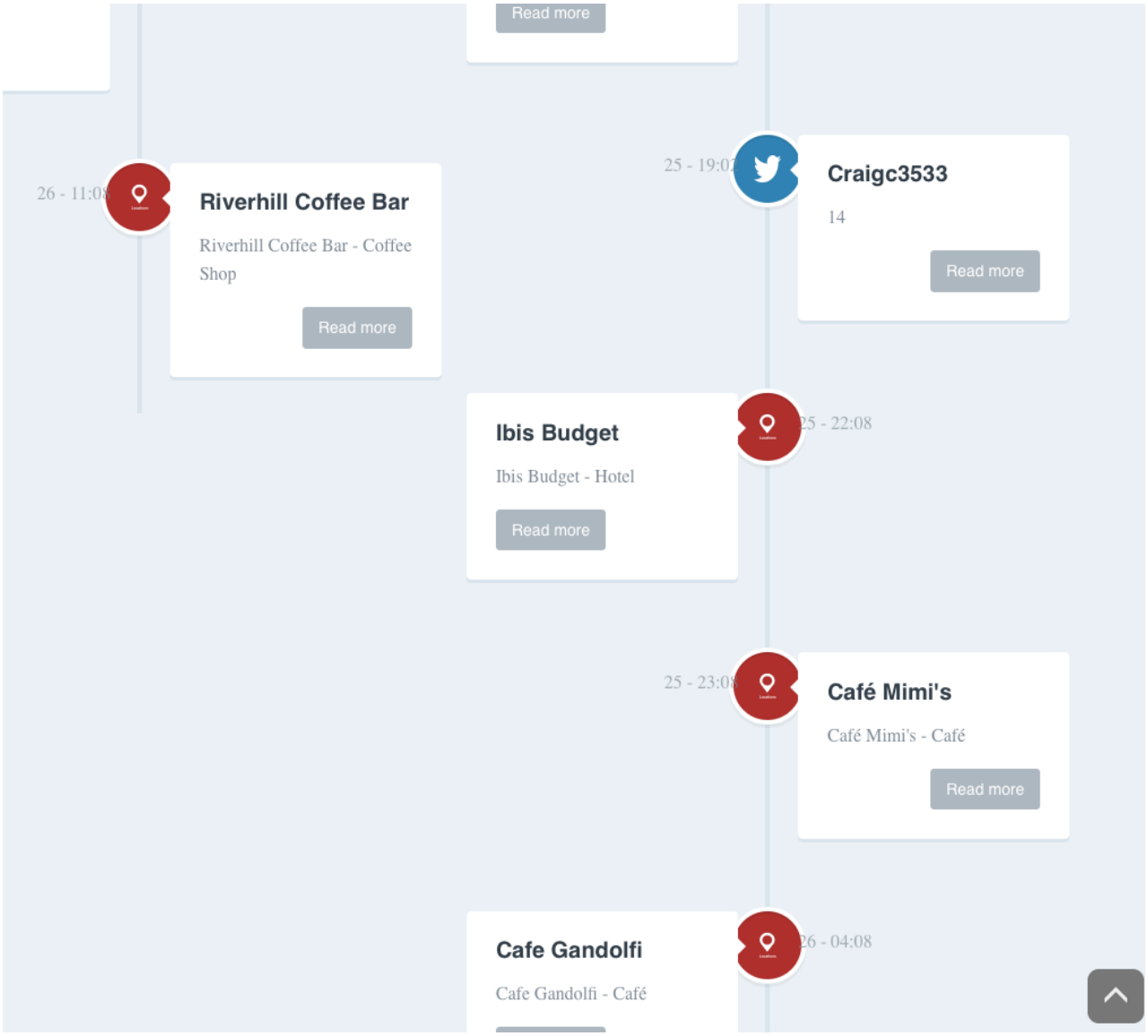


Figure 6: Timeline version 1 in comparing view.

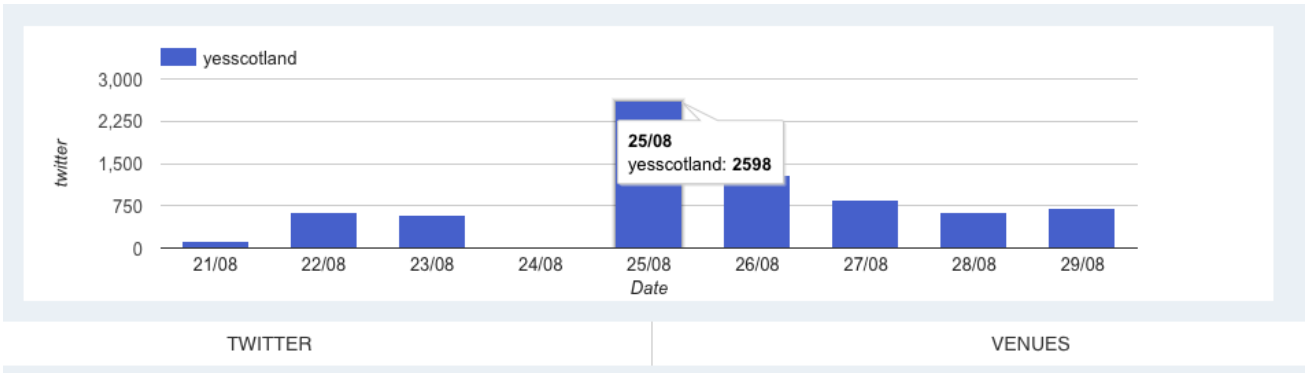


Figure 7: Week preview with Google Charts showing number of tweets on each day.

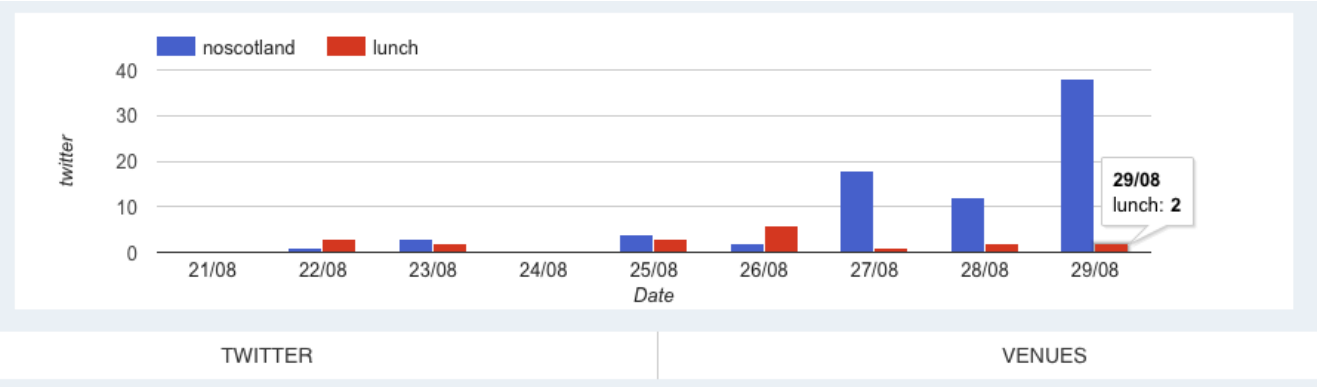


Figure 8: Week preview used when comparing two events.

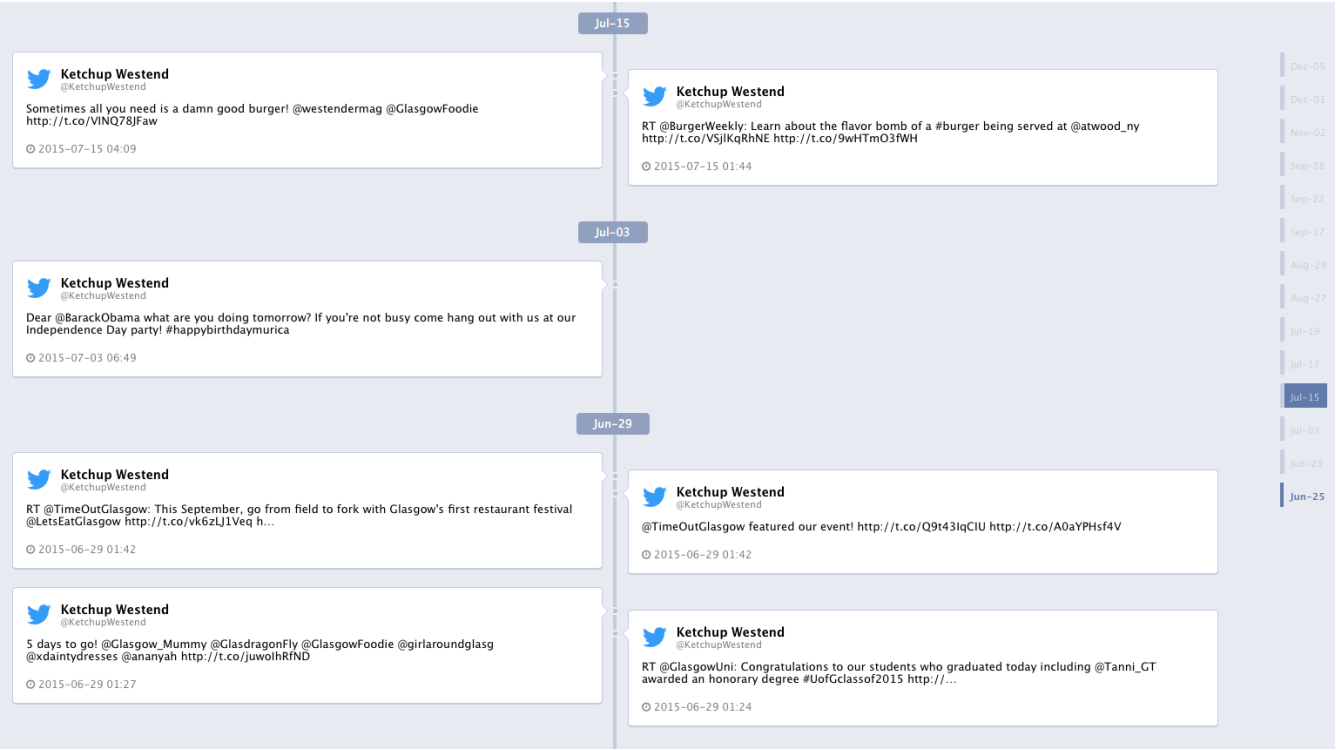


Figure 9: Timeline version 2

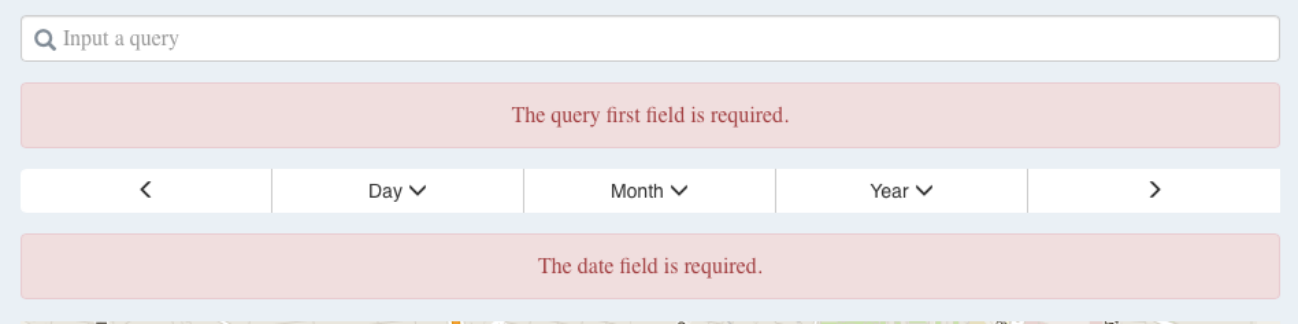


Figure 10: Validation error messages

	Query	**No query**
Location (click on the map)	Timeline of tweets with information about busy venues in radius of 1 mile.	Timeline of busy venues around the area
Location (click on venue)	Timeline of tweets for the query combined with tweets for this venue and hourly information about this venue	Venue time series – including hourly information about the venue with tweets from the venue's twitter account
Train Station	--- Same but for train stations ---	Timeline of train delays

Table 1: Querying / extracting data

Appendix C

Evaluation Raw results

C.1 Time

Descriptive Statistics

Dependent Variable:Time

Interface	Task	Mean	Std. Deviation	N
1	1	215.2965	117.97028	6
	2	152.1867	36.62344	6
	Total	183.7416	89.56454	12
2	1	168.5225	19.95123	4
	2	336.3513	41.73111	4
	Total	252.4369	94.68111	8
3	1	171.6220	124.19341	2
	2	215.6900	126.75596	2
	Total	193.6560	105.56683	4
Total	1	192.4261	91.69742	12
	2	224.1588	99.80429	12
	Total	208.2924	95.12103	24

Dependent Variable:Time

		task					
		1			2		
		Time			Time		
		Minimum	Mean	Maximum	Minimum	Mean	Maximum
interface	1	101.87	215.30	430.00	91.00	152.19	196.21
	2	145.88	168.52	194.00	285.74	336.35	385.21
	3	83.80	171.62	259.44	126.06	215.69	305.32

Dests of Between-Subjects Effects

Dependent Variable:Time

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power[b]
Corrected Model	93903.206[a]	5	18780.641	2.960	.040	.451	14.801	.726
Intercept	865508.607	1	865508.607	136.419	.000	.883	136.419	1.000
interface	23679.685	2	11839.843	1.866	.183	.172	3.732	.337
task	12075.025	1	12075.025	1.903	.185	.096	1.903	.257
interface * task	64181.748	2	32090.874	5.058	.018	.360	10.116	.747
Error	114201.010	18	6344.501					
Total	1249361.757	24						
Corrected Total	208104.216	23						

a. R Squared = .451 (Adjusted R Squared = .299)

b. Computed using alpha = .05

C.2 Clicks

Dependent Variable:Clicks

		task					
		1			2		
		Clicks			Clicks		
		Minimum	Mean	Maximum	Minimum	Mean	Maximum
interface	1	12	22	44	12	13	14
	2	27	30	33	14	16	18
	3	21	27	33	13	15	17

Descriptive Statistics

Dependent Variable:Clicks

interface	task	Mean	Std. Deviation	N
1	1	21.50	11.623	6
	2	13.17	.753	6
	Total	17.33	8.978	12
2	1	30.25	2.754	4
	2	16.00	1.826	4
	Total	23.13	7.918	8
3	1	27.00	8.485	2
	2	15.00	2.828	2
	Total	21.00	8.641	4
Total	1	25.33	9.345	12
	2	14.42	1.929	12
	Total	19.88	8.639	24

Tests of Between-Subjects Effects

Dependent Variable:Clicks

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power[b]
Corrected Model	925.542[a]	5	185.108	4.212	.010	.539	21.059	.881
Intercept	8241.004	1	8241.004	187.513	.000	.912	187.513	1.000
interface	167.083	2	83.542	1.901	.178	.174	3.802	.342
task	652.367	1	652.367	14.844	.001	.452	14.844	.954
interface * task	43.417	2	21.708	.494	.618	.052	.988	.118
Error	791.083	18	43.949					
Total	11197.000	24						
Corrected Total	1716.625	23						

a. R Squared = .539 (Adjusted R Squared = .411)

b. Computed using alpha = .05

Appendix D

Evaluation Documents

Participant Consent Form
Urban Data Timeline

D.1 Consent

The main aim of the experiment is to test how valuable the implemented features are for the users as well as how usable the system is.

The experiment will take about 10 minutes to complete.

At the start of the experiment, the home page of the application will be presented as well as a list of the tasks you will need to perform. After finishing one of the tasks, you will be given a questionnaire. Meanwhile, another interface will be presented and you will have to perform another task from the list and fill the questionnaire again.

At the end of the experiment, you will be encouraged to share what you liked or disliked about the system.

All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university.

Please note that it is the web interface, not you, that are being evaluated. You may withdraw from the experiment at anytime without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:

Yordan Yordanov
School of Computing Science
Lilybank Gardens
2038011y@student.gla.ac.uk

I have read this information sheet, and agree to voluntarily take part in this experiment:

Name: _____ Email:_____

Signature: _____ Date: _____

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

D.2 Introduction and Debrief scripts

Introduction Script

Urban Data Timeline

The main aim of the experiment is to test how valuable the implemented features are for the users as well as how usable the system is. It will involve completing a list of tasks on a web interface and then filling a questionnaire. In the end, I encourage you to think about what did you like or dislike about the system and share your thoughts with me. Please remember that it is the system, not you, that is being evaluated.

The tool that you are going to use visualise timely data originating from various urban big data streams such as social media posts, news, blog, environmental sensors and many more.

Do you agree to take part in this evaluation?

You are welcome to withdraw from this evaluation session at any time.

Do you have any questions before we start?

Debrief Script

Urban Data Timeline

The main aim of this experiment was to test how valuable the implemented features were by following if you used them or not. Furthermore, I was looking if you were confused about some part of the system.

Do you have any comments or questions about the experiment?

Please take a note of my email as I encourage you to use it to contact me in case you have any further questions or comments.

Email: 2038011y@student.gla.ac.uk

Thank you for your help and participation in this experiment!

D.3 Task sheet

Task Sheet

Urban Data Timeline

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.

Task 2:

Compare tweets with hashtags – noscotland 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.

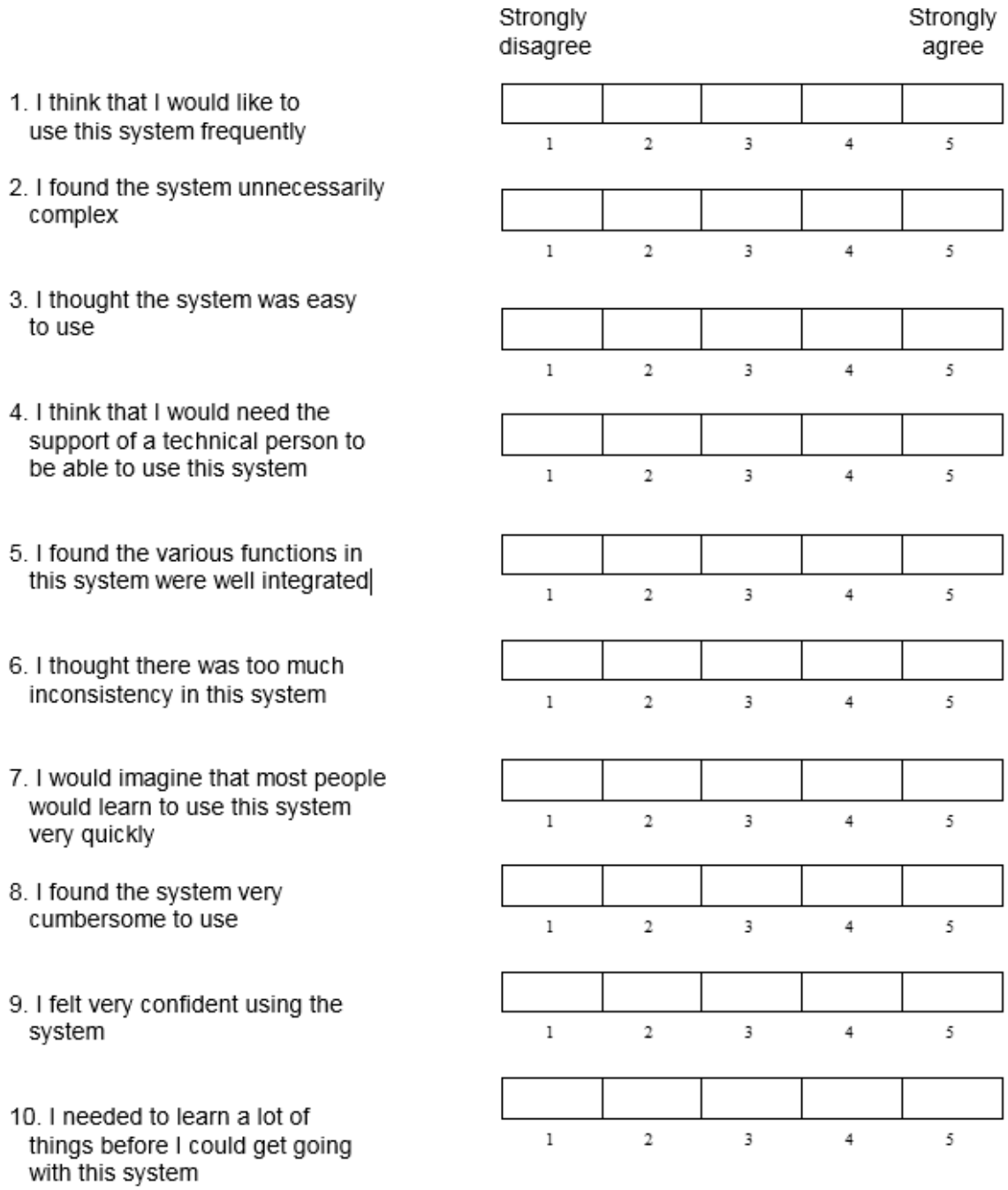
Task 3:

Find out if there were any delays in Glasgow Central on the 5th of September 2014 and if there were, write down a sentence summarizing people's opinions about them if there were any.

D.4 Questionnaire

Questionnaire

Urban Data Timeline



Appendix E

Setup

Urban Data Timeline

Starting the server

Setup guide

Typically, a web server such as Apache or Nginx can be used to serve this application. If there is PHP 5.4+ installed that comes with PHP's built-in development server, the serve Artisan command can be used to start the application but only if it is executed into the main directory of the project:

$ php artisan serve

By default the HTTP-server will listen to port 8000. However if that port is already in use, the port can be specified. Just add the --port argument:

$ php artisan serve --port=8080

Services configuration

Furthermore, the “Urban-Data-Timeline\config\services.php” file has to be modified with the urls to each of the services. This change is not required if a ssh tunnel to the services is opened, using the following command:

$ ssh -L 8080:localhost:9057 -L 8081:localhost:9058 GUID@sibu.dcs.gla.ac.uk "ssh -L 9057:130.209.67.145:8080 -L 9058:130.209.67.145:80 roma"

Twitter API

Finally, if the communication with the genuine Twitter API is required in order to get tweets from any venue twitter account, the “Urban-Data-Timeline\config\ttwitter.php” file has to be modified with the corresponding CONSUMER_KEY, CONSUMER_SECRET, ACCESS_TOKEN and ACCESS_TOKEN_SECRET.