University of Glasgow | School of Computing Science

Urban Data Timeline

Yordan Yordanov

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

Level 4 Project — March 28, 2016

Abstract

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

Education Use Consent

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

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

1.1 Motivations

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

1.2 Aims

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

1.3 Background

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

However, the biggest asset of a smart city is not the number of cameras or sensors but the people that occupy it. A third of the internet users[5] represent their smartphone 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 analyzing 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[2] (iMCD) is one of the Urban Big Data Centres inaugural projects, funded by the Economic and Social Research Council. It is designed to provide the UBDC with innovative primary data sources. The project consists of four strands: representative household survey, tracking of real-time urban sensors, internet based visual media collection and internet based textual media collection.

The core research strand is the representative household survey that aims to gather data about peoples attitude and behaviour when it comes to information and communication technologies, traveling 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

Tell a bit about Terrier IR

1.3.3 Web Services

Describe what a web service is

1.3.4 Web Application

Describe what a web application is

1.4 Outline

The rest of the report is structured as follows:

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

Chapter 2

Related Work

There are applications that have been developed to accomplish similar if not the same goals. This chapter is presenting a short description of related products, along with key differences and correlations in comparison with the goals of 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 [9] 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 [6].

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

Social Media Timeline

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

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

Tech City Map

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

Conclusion

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

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

Chapter 3

Project Planning

3.1 Software design approach

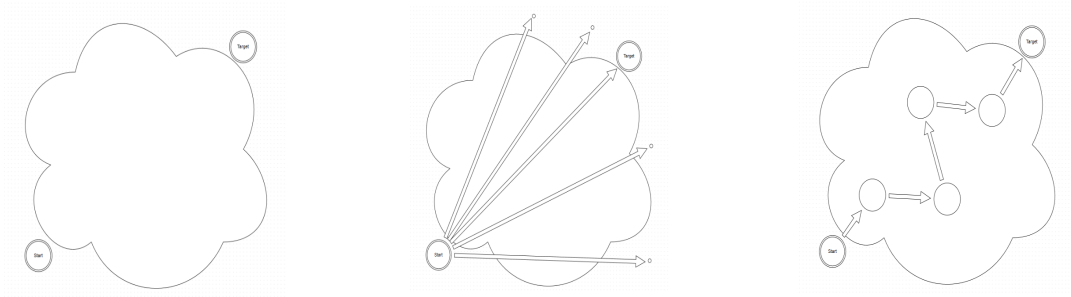
When a new product is developed, it is not clear how it will end up and if it will fulfil the user's requirements. The developers can see the first steps but there are plenty of problems or challenges that can not be predicted from the beginning (Figure 3.1a). That is true for any project, no matter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

3.1.1 Waterfall model

The Waterfall model is an example of a plan-driven process - in principle you must plan and schedule all of the process activities before starting work on them [8]. 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 [8]. Each of these sprints last one week. It starts with a meeting and demonstration of the work done in the previous sprint. As an outcome of these meetings feedback from the supervisors is received, based on the extent to which their requirements are met. Moreover, these meetings allow discussions of the issues that are experienced and compare the available workarounds. At the end of a meeting, it is agreed upon exactly what work will be done during the next sprint. Following this scenario, at least one new feature is introduced after each sprint and changes to previous features are performed as early as possible. By the end of this project, 23 iterations are to be performed.



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

Figure 3.1: Software development methodologies

3.2 Requirements Engineering

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

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

3.2.1 Use cases

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

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

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

3.2.2 Functional requirements

Functional requirements represent services the system should provide, how it should react to particular inputs and how the system should behave in some situations. More specific requirements can describe the system functions, its inputs and output or exceptions in detail [8].

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.

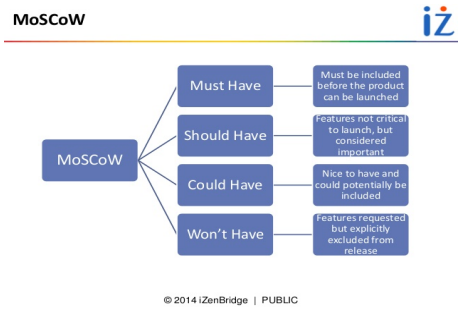


Figure 3.2: MoSCoW rules prioritisation scheme for requirements

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.

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

3.2.3 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 [8]. 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

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[3].

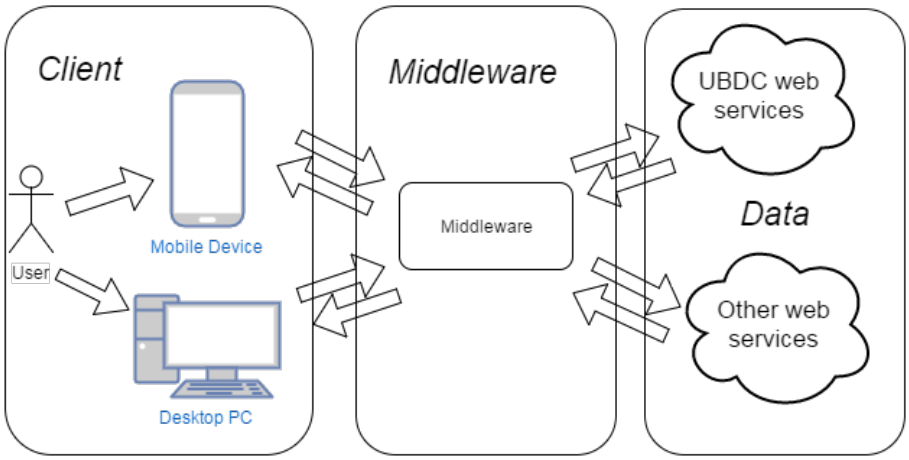


Figure 3.3: High level architecture diagram based on requirements and use cases.

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***. [7]

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 [7]. The following list represents the user stories, captured for this project based on the identified target users(TODO: should I talk about target users):

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

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

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

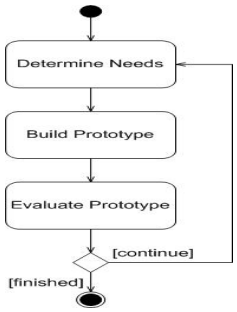
Chapter 4

Design

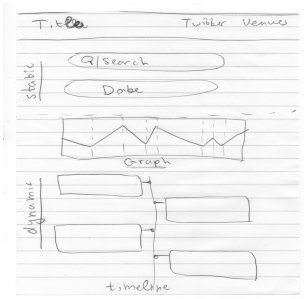
4.1 Paper prototypes

As shown on the 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. One the sketch (Figure: 4.1b) was finished, 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.

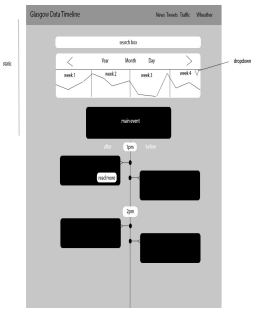
During the implementation process, there had been a few changes to the design. In iteration 6, 8(reference progress reports) and 9 new requirements were identified and modifications to the whole design were required (Figures: 4.1d and 4.1e).



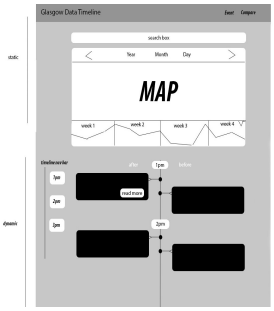
(a) Prototyping process



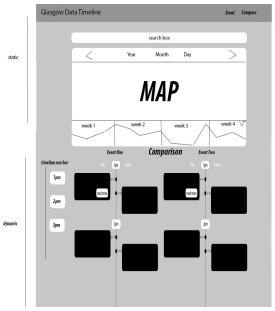
(b) Initial Paper Prototype.



(c) Wireframe from the paper prototype



(d) Wireframe for event exploring



(e) Wireframe for comparing events

Figure 4.1: Paper Prototypes and 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, provided in the referenced section, a more in-depth diagram was designed to outline some of the key components of the system (Figure: 4.2).

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: 4.1d and 4.1e). 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. As for now, only the first four services are used. The second on is the Twitter API. It is used to compensate for some of the limitations of the other Twitter service, provided by the UBDC, and to broaden the possibilities of tweets querying.

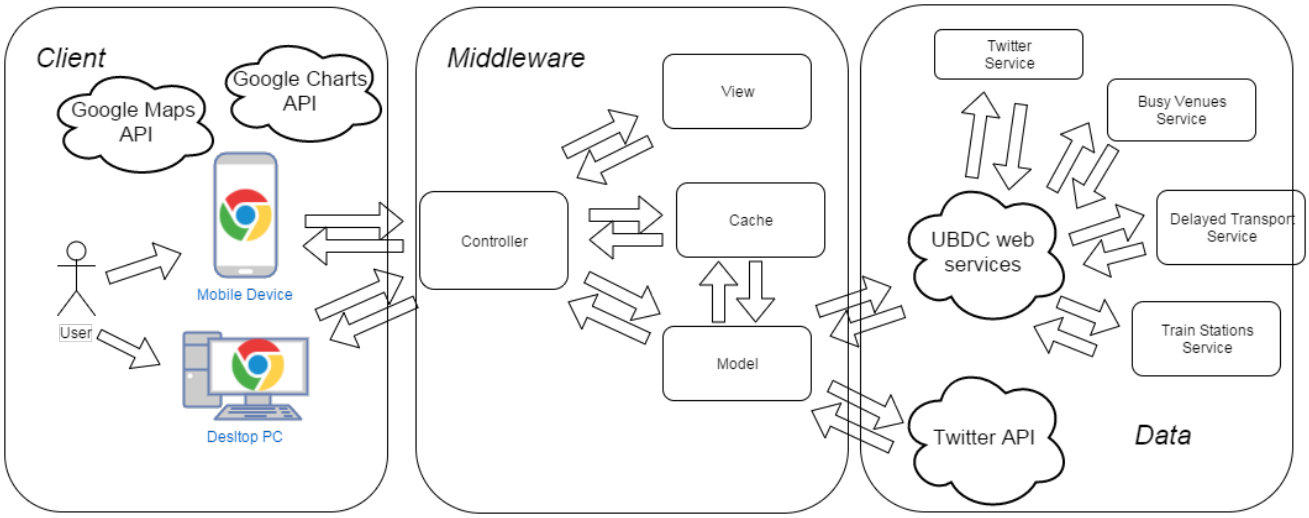


Figure 4.2: Component level architecture diagram.

4.3 Components communication

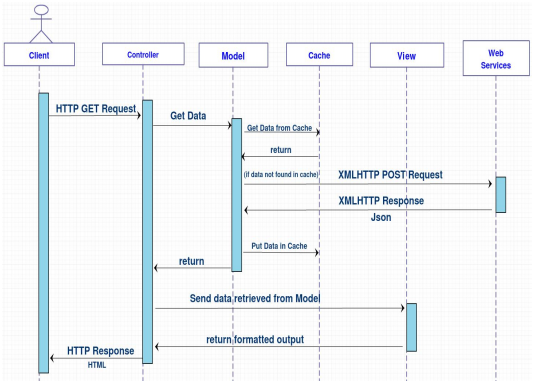
Following from the previous section, this section describes how actually the components are communicating between each other. There are two scenarios, based on what request is made by the browser.

4.3.1 HTTP Request

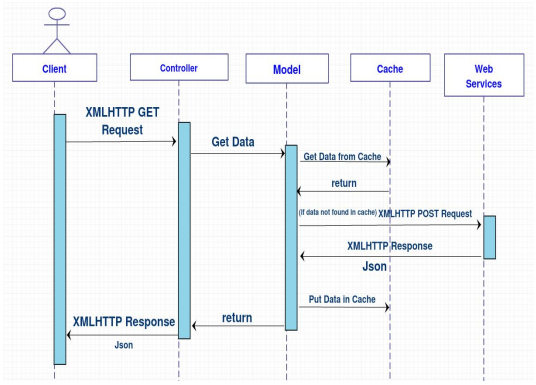
The first one(Figure: 4.3a) follows the traditional pattern when the user wants to access a page, which visualises data, using the application. To begin with, the browser makes a HTTP GET request to the controller, that maps to a specific URL. Then the request is validated against a set of rules and 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 specific Web Service and a XMLHTTP Response is received in the form of a Json object. Depending on the initial request by the user, 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 request by changing only one or two variables at a time. Once the model is 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 back to the Controller. Finally the Controller returns HTTP Response with the generated HTML.

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) cannot be rendered by the View. In addition, these elements use Ajax that performs XMLHTTP Requests in stead of HTTP Request. In order to handle such requests, the system must provide REST endpoints. If the browser makes a XMLHTTP request to one of the endpoints, the Controller, same with 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 user.



(a) HTTP Request Sequence diagram



(b) XMLHTTP Request Sequence diagram

Figure 4.3: Components communication

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. 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 try and find 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 external service. 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.

4.4.2 Web Application Framework

Play Framework

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

- It dramatically improves the productivity of a developer compared to other Java based web frameworks like Jersey or Spring MVC. The server not always has to be restarted 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[1] and available libraries.

Cons:

[1] Integrated Development Environment

- Play 2 is relatively new framework so the community is not as big compared to other Java frameworks.
- The framework is immature. Not all bugs have been fixed and good practises are still no clearly defined.
- Java does not provide features like closures to keep asynchronous code clean. Play is build 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 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 little piece of code. In addition, this automatically leads to less hours spent in developing and lower chance of introducing bugs.
- 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 design 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 introduced.
- 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.
- If the application is using a database, long queries that make use of multiple JOINs or UNIONs can be quite hard to implement.

Laravel

Laravel 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.
- Community is great with many active people on the Laravel's forums.
- Error messages are easy to understand and points directly to the error.
- PHP: built in JSON support.

Cons:

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

4.4.3 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 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 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.

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 this problems but Bootstrap was the only one considered for this project as first of all is one of the most widely used. It is very simple to get started with: just reference the framework into the header of the page. Bootstrap offers 12 column grid system as well as many layouts and components. One of the most important features is providing responsive utility classes

that can automatically rearrange a page based on the screen size of the device. Bootstrap has great documentation with a lot of examples and 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:

- 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:

- 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.

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[4] was the only one considered as it is free and offers 25,000 map loads per 24 hours for 90 days. It also provides wide variety of customisable options like colours, shapes, markers and many more. Furthermore, Google Places[1], 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.

Chapter 5

Implementation

5.1 Model

5.2 View

5.3 Controller

5.4 Challenges

Chapter 6

Evaluation

6.1 Product evaluation

6.1.1 Initial phase

6.1.2 Final phase

6.2 Testing

Chapter 7

Future work

Appendices

Appendix A

Running the Programs

An example of running from the command line is as follows:

```
> java MaxClique BBMC1 brock200_1.clq 14400
```

This will apply $BBMC$ with $style = 1$ to the first brock200 DIMACS instance allowing 14400 seconds of cpu time.

Appendix B

Generating Random Graphs

We generate Erdós-Rënyi random graphs $G(n, p)$ where n is the number of vertices and each edge is included in the graph with probability p independent from every other edge. It produces a random graph in DIMACS format with vertices numbered 1 to n inclusive. It can be run from the command line as follows to produce a clq file

```
> java RandomGraph 100 0.9 > 100-90-00.clq
```

Bibliography

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

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

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

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

[5] Ofcom. The communications market 2015.

[6] 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.

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

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

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