CS5003 - Masters Programming Projects

Project 3: Data Mashups

Student number: 130018883 Words: 691

Introduction

The purpose of this practical was to develop an application that visualised data from real world APIs, as well as learning how to work with modular code, and in a large team.

Our application KivaStats takes information from the Kiva and World Bank APIs, and uses d3 and the Google Maps API to generate tables, charts and maps using this information.

Division of Labour

The following is a rough outline of the divisions of labour, as decided towards the start of the development process. As well as the sections described below, we supported each other by discussing each section in team meetings, as well as online, and reviewing each other's code

Katherine Brooke	Client side - d3, HTML, CSS
Thomas Brunner	Client side - d3, HTML, CSS
David Burgess	Server side - Kiva API, World Bank API
Niall Colfer	Server side - Kiva API, World Bank API
Stefan Gladkov	Server side - World Bank API, Kiva API
Jairus Houdek	Client side - Maps
Scott Schorr	Client side - Maps
Xiaochen Sui	Client side - Maps
Stefan Vassilev	Client side - Maps

My Contribution

At the beginning of the project, my teammate had built a simple server file that got information from the Kiva API and sent it to the client. We saw that this took some time to process, as the server made calls to the Kiva API on each click. To improve this we decided to cache data in MongoDB, and began looking at the best way of doing this. We decided that our application would serve new data if the cached data was > 48 hours old, and designed a process that would achieve this, which my teammate then encoded into the server file.

When another teammate had written functions that retrieved information from the World Bank API, I then wrote a function that performed this database check and caching for the World Bank API, and served data from the database to the client. After this was functional, I realised that having two functions for this (i.e. one for Kiva and one for World Bank) was overkill, and so consolidated the functionality into the `'accessAPI()'` function, and used it for both. I also refactored and cleaned the server code, as it had gotten somewhat convoluted during development.

While this was ongoing, the Maps team were building the client side using pre-built data. We then combined this with the existing server file so that they were getting data served from the database. This lead to some teething problems, as the data was formatted slightly differently. I worked with a member of the Maps team to resolve this by manipulating the data on the server side before sending it to the client.

After this was coded by hand, I again saw that manually parsing the data in the same way for each call was something that could be exported to a function, so wrote the `'parseMapData()'` function that took the information from the database and reformatted it as the client side wanted it, and called this function on the data before returning it in the response object.

Finally, we had to make some changes to the way in which cached data was checked before a new API call was made. We had been using the `.getDate()` method on the JavaScript date object, but since this just returned the date (i.e. without the month or year), we acknowledged a limitation whereby old data would be served if an API call happened to be made on the same date but in a later month or year.

I did some research online, and decided that the best way of storing a timestamp in the database would be in the format 'YYYYMMDD' as an integer. This could then be compared to the current date, in the same format, to see if new data had to be called. To encode this I wrote the `'parseDate()'` function, which took in the date object and returned an integer of the aforementioned form. I had to convert to string first, to add a leading 0 should the month or date contain only one digit, but once this was complete it functioned as desired.

Once we had all completed the code for our respective sections, I assisted the team by reviewing all of the code, and suggesting some ways it could be refactored and cleaned. I then tested the entire system by removing my entire local configuration on a lab machine, and running the application from the perspective of a new user, which worked as expected, as well as contributing to the README file