CS1013 - Programming Project I Documentation - Team 4

Team Members

- Anton Yamkovoy - Student Number 17331565
- Jack Gilbride - Student Number 17340868
- Luke Hackett - Student Number 17324340
- Owen Johnston - Student Number 17330339

Outline of Design

Initial Project: During the initial stages of the project we planned how we wanted it to be presented, on paper through diagrams and illustrations. We also set up a memory based version of some of our features, using the sample dataset, which was limited and consisted of 1500 reviews for businesses. This quick draft that we completed as our first week demonstration showed us the limitations of the sample dataset, and pushed us on to the idea of setting up a database and query system, which is the basis of how our final project finds information.

Backend: Luke and Anton were responsible for the backend. We decided that we wanted to use the big dataset so Luke uploaded the tables to a MySQL server hosted on his machine at home, as well as a backup server hosted on Amazon Web Service relational database service for redundancy. Speed was a concern so we added Full text indexing to most of the tables. We also wrote a class full of functions that queried the server for the desired data that were optimised for speed. A principal we used when writing the query functions was to write many small functions with accurately defined roles, so that efficiency was maximised. To optimize further Luke split most queries to run on different threads, so results could be returned in parallel and the program doesn't hang. This involved making multiple crawlers for images, graph data, reviews, and maps.

Organisation

Throughout the six weeks of the project, we organised a group meeting during the week when it was suited to everyone's schedule, and organised a team programming time at 10-12 on a Wednesday, This was our allocated time outside of the labs for working on the project together, aside from the work done at home.

For communicating between team members, we used a Facebook / Discord chat to update everyone on changes made and to discuss new features or ideas about the project. This worked out well, as we could discuss before starting to implement code, and mix-ups with who is completing what sections of the project didn't happen often. Also SVN logs were helpful, in managing minor changes which weren't mentioned in the group-chat.

In our SVN folder, we tried to organise the project based on IDE used and production / testing code. *programmingProjectIntelliJ* was used as our main folder and *TEST FOLDER* was for introducing new libraries and dependencies, and alternative methods or functions to be tested.

Individual Breakdown of Work

- **Jack :** I worked on the Scrollbar (Week 1&2), testing the Widget class (Week 2), testing and trying to implement UnfoldingMaps (Weeks 3&4) and the Business Hours Chart (Weeks 5&6).
- **Anton :** I worked, on setting up the db_Access, later merged with queries class, which contained query function setup, connection and the query functions themselves (Week 1&2), Setting up the classes and methods used for the graph classes, CheckinBarChart and StarLineChart (Week 3&4), and working on the business screen, implementing the graphs into our business page, for the final weeks of the project.

- **Owen:** I worked on formatting of the reviews, and made it in a way that only an arrayList of Reviews would have to be passed and then all their reviews would be formatted to a specific screen width and all would be separated out properly (Week 1&2), added the functionality to be able to easily switch screens using the controlP5 controllers. I created the category button to the home screen, when pressed would change screen to the searchResult screen after running one of the queries that Luke and Anton made, returns all the businesses of that category. On that business screen page, it was my job to display all the businesses as a separate button on the searchResultScreen, using the controlp5 library. (Week 3&4) Created the animation class, which takes in a filename prefix and the number of frames to be added and creates an array of images and draws them at a defined frame rate. I added the number of stars each reviewer gave onto each of their reviews. I then added the filter dropdown for the reviews, allowing the user to filter reviews by the number of stars each review had. I also created the attributes icons and added them to the project which query the server which returns a businesses attributes and then checks to see if parking, wifi, and wheelchair access are true (Week 5) From then the majority of the functionality of our program was there and was only a matter of making it look better with the UI elements. This was mainly my job to create the elements in Photoshop and then add them all to our existing elements in the project and repositioning them to their final place. (Week 6)
- **Luke:** In the first week I setup the mySQL server that was hosted on my machine at home. I also wrote a companion java class(queries.java) that connected to the server and had a long list of queries that were extended over the coming weeks. I also added indexing to the server and constantly tried to improve the efficiency of the queries. I was also responsible for the image search that user Bing cognitive image search api to get pictures of the businesses dynamically. These images were fetched using an ImageCrawler object for each business which all ran on seperate threads, meaning that the program would not hang and the images would all load simultaneously. I implemented this multi threaded approach in almost all parts of our project so instead of the program freezing we could have loading animations that owen implemented. I was also responsible for the static map api on the main business screen that got the business co ordinates and address and loaded an interactive map that the user can zoom in and out of. I wrote the GraphScreen object to make it easier for all the graphs written to be contained in one place and easily select which one to display. Finally, I refactored the code when it started getting too messy, made Drawable and UI classes and moved code from main to those classes. Halved the number of lines in main.

Features Implemented

- **Scrollbar:** This was an extension of the Widget class from previous lectures. As we used ControlP5 for all other buttons, Widget was made into an abstract class which was extended by Scrollbar. Scrollbar was passed the total height of what was to be scrolled through, and worked out its own height from that. Everything that was drawn on the screen had its initial Y value subtracted from the distance from the scrollbar to the top of the page multiplied by the ratio of the scrollbar size to SCREEN_Y. There are two Scrollbars in the program, one to scroll through businesses and another to scroll through reviews.
- **Star-Rating Change Over Time Graph:** This graph represented the change in star-rating over time for a certain business, the query was based on the business_id field in the database for each business, the query from a “name” string, found the business_id using a query function. Using this business_id, we found an ArrayList of data using a getStarsList() function. This was passed into the class that represented instances of the StarLineChart graph.
  The average stars-rating over each month segment was calculated this represented one datapoint in the line chart that we used to represent this graphic. Originally a bar chart was used, but for variety we changed this to a line chart. This chart had a limit of twelve months, ie representing the average rating change over the last year of the business. A problem we encountered was that not all businesses had enough of this data to make the chart presentable. This graph was implemented

into the graph using the GraphCrawler class. The graph classes themselves were implemented using the giCentre line-chart library.

- **Check-ins Bar Chart:** This graph represented the amount of people that checked-in to a business over it's lifetime or presence in the dataset. The chart sorted the results into Monday-Sunday categories which were represented by the bars in the chart. We used similar queries to find the business_id and pass it into a query function that found the ArrayList containing the input for the graph that whose instances were represented by the checkinBarChart class objects. Similarly the GraphCrawler class was used to implement the graph into the business screen. Similarly to the star-chart, it was implemented using the giCentre barchart library.
- **Business Hours Chart:** This took the opening hours from a business using the *getBusinessHours()* query and displayed them. It consisted of a 2D Array of booleans, one boolean for every half hour of the week. The business would either be open (true) of not open (false) during each half hour. Rounding was used for businesses that didn't open on the half-hour exactly. It was displayed as a 2D grid and stored in GraphCrawler like the other graphs/charts, although was not implemented using a library.
- **Animation:** Owen added an animation class that allows us to add an array of images which draws them at a certain framerate.
- **Attributes:** Using the big dataset allowed us to be able to show off some of the main attributes of a business, the main three being if parking, wifi, and wheelchair access was available. These three attributes are shown off from the search results screen.
- **Review Filter:** Provided the ability to the user to filter the reviews by the number of stars given.
- **Business Images:** Program dynamically fetches images and has a redundancy google image search ini the case that both bing searches fail
- **Business Map:** The program shows a map of the business and surrounding area that an be zoomed in and out of.
- **Search Bar:** It is possible to search the dataset of over 100,000 businesses for specific categories, business names or city and have the results returned almost instantly using a mySQL server.
- **Database/Server:** The program does not store any data for the businesses on its local machine, instead it can dynamically search a SQL server with Full Text indexing optimised for the fastest search times.

Problems Encountered

- **Using Widget Class:** During the second week of the program, Jack worked on a test program (*TEST FOLDER > Homescreen_program*) to see how the Widget class would fit into our program using buttons, a scrollbar and a searchbar. We decided that it would be too awkward to use these, with each widget having an event that was checked through a case statement, when we realised the sheer amount of buttons that we would have on the screen. To solve this problem, Luke and Owen used the ControlP5 library for the buttons and searchbar, but we kept the code for the scrollbar as it was quite easy to use.
- **Total Height for the Scrollbar:** At first it was difficult to figure out how to get the scrollbar to scale in different cases. We figured out that it would be a ratio of the height of what we could see on the screen compared to the "total" height of the screen, i.e. including what could not be seen. Owen worked on functions to get this total height, which was passed into the scrollbar and easy to work with from there.
- **UnfoldingMaps:** Originally we were using a library called UnfoldingMaps to display where a business was, using *getLatitude* and *getLongitude* from the Business class. There was no available version online for Processing 3 so Jack emailed the developer to get the most recent version. This worked fine in all test programs (see *Unfolding0_9_9_Testing* in *TEST FOLDER*) but we could not get it working properly with our program. Unlike in tests, we could not scroll/zoom within it. It was also quite slow and wouldn't move in relation to the Scrollbar. We decided that it was not worth it to use this library so we decided to use the Google Static Map API, which Luke worked on.

- **Formatting of Reviews:** Using the big dataset meant that none of the reviews were previously formatted, this meant that we had to format the reviews for a certain screen width by adding in our own new line characters to ensure the reviews don't run off the screen. In order to separate each review out, the total height of the review needed to be found so the next review could be drawn underneath that. Owen solved this by counting the number of new line characters and multiplying the number of lines by the height of the current text to get the full text height of a review.
- **Font Inconsistency** The font was being changed from all different parts of the program to the point where it was completely unmaintainable, we fixed this by clearly defining the different fonts that will be used in Main and removing all textSize() implementations.