Programming Project Report - Group 17

April 12th, 2019

Introduction

“Notice the small things. The rewards are inversely proportional” - Liz Vassey. Being able to view and evaluate a single users information, contribution and statistics in the midst of all the other users in such a large dataset is essential for this website to be practical and useful. It is a principal part of UI design to have be able to display information in a way that is not overwhelming and easy to digest and that was what we aimed to achieve.

The spotlight we have on each user aims to convey a clear picture of the users activity, patterns, contributions etc. Essentially the user page by nature demands functionality i.e different methods to interpret the users information and ways to compare the users information with other users to provide context and also build a more practical interface. To achieve this the user page utilises some useful widgets to help with user evaluation.

Teamwork

When we met up in the first week, we assigned roles to ourselves. This way, none of our work would overlap with one another. The roles we settled on were: Koh and Nixon on the front-end, Stefan and Adam on the back-end, and Stefan would also act as the project leader/coordinator.

Each week we met up for an hour to discuss our goals for the coming week. We got a chance to resolve any issues we had with our code and to stay on track with the assigned milestones. We found that these meetings were very productive in terms of team and project management. All of the work that we’d done individually over the last week came together. This allowed us to look back on the project as a whole and continuously assess our progress. We discussed any new ideas we had and appreciated others’ ideas and opinions to settle on a solution that we would all enjoy working with.

User Page, Analysis Page & Widgets

Implementation

Although the user page utilises different widgets it is in fact a widget in itself (this would later prove useful in the implementation of widget animations). The user page is extended from the page superclass which handles the functional aspects of the page. For speed and ease the user page utilises SQL to load user specific information from the database such as posts, comments, stories etc.

The user page chooses what user will be analysed by taking in the information from the search bar or is called when an author/username is clicked. Our code is designed this way for simplicity and to maintain the focus of being user friendly and intuitive. The user page loads a list of story widgets similar to the one that is displayed at the start of the programme however the userpage uses an SQL function to ensure that the stories loaded are the ones written by that specific user.

The analysis page is where the more useful information about the user and other users by extension are displayed, the analysis page includes graphical information such as showing the users activity on a day to day basis.

- This information was displayed via a barchart class.
- The data for most active days was taken from the database via a SQL function.

The analysis page also displays a users most popular stories by creating an ordered list of popular stories. The popular stories are loaded from the database via an SQL function.

At a quick glance the clock widget also provides light but useful information, through subtle increasing of clock hands and changing of colour the time that a user is generally activity is visible and clear.

This pie chart was to be another widget that would’ve displayed more visual information on the user, such as a comparison between the users stories, comments and score. The pie chart gets information from a specific user by implementing the user to make use of its SQL functions.

Lists, Auto-complete, and Dictionaries

List

The List stores its widgets as an ArrayList, it can be seen as a moving plane, to which the widgets are attached. Every time the list moves up or down, all the widgets attached to it are moved by the same amount, so that it can be scrolled through. The top of the list is moved up by some percentage of the total height of the widgets stored in it, which allows widgets that are around that percentage of the lists height to be displayed on screen. The percentage by which its moved is taken from the ScrollBar and it's just the ratio of ScrollBar slider's position and the total length of ScrollBar. New widgets can be added at the bottom, using addNew function, which returns the percentage to which the ScrollBar should be set to avoid any jumps of the widgets around the screen.

Dictionary

Dictionary class is used to store words using a Trie data structure. It allows to store words in a compressed manner, as a tree where each node correspond to some character, and each path starting in the root corresponds to some String, created by adding together all of the characters from nodes on that path. Each node also stores the count of how many words end in it, which makes it possible to see how often each word was inserted. Nodes also store an array of 5 most popular words that have this node as a prefix, which helps with autocomplete. The array uses wordFrequency class which just stores a word and an int corresponding to its number of appearances. It also implements comparable, so it can be sorted by the amount of appearances.

Autocomplete

The autocomplete tries to complete the passed in word by looking for it in the dictionary and trying to find a word that the user might want to use. It has 2 parts to it. One just takes the list of most popular words that have the passed in word as a prefix, this array is stored in a dictionary node corresponding to the passed in word. The second part tries to use recursion to go through all possible paths down from that node and add up to 4 characters and then returning the list of all words that were found and their number of uses. This is combined with the most popular words from that node and the whole array is sorted descending by the number of appearances of each word before being returned. Unfortunately, this took a lot of RAM and was a bit slow for the bigger data sets, so we didn't end up using this.

SQL

All of the data used by this program is done through MySQL. The connections and queries are handled by the SQLInterface class, and queries are made to this server using the Query class.

When the program loads, the SQLInterface's connect method is called on a separate thread. This allows the rest of the program to function without waiting for this long task to complete. If the SQLInterface can't connect to a MySQL server, it will start its own one using the MariaDB4J library and get a connection.

If the user has a MySQL server already running, then no new data needs to be loaded and the UI is immediately usable. The SQLInterface will only query the server then the user requests more data.

However, if the program needs to start its own MySQL server, then it will also read the data from the dataset. It reads the file line by line, parsing each line as its own JSON object and inserting it into the appropriate table (stories or comments) in the SQL server. As it reads through these stories and comments, it also creates data about each user (which stories and comments they wrote) and inserts that data into the user table. After the data is inserted, indices are created on each of these tables to improve the efficiency of the queries made.

This loading process can take a very long time which is why it is only done when necessary. If the user wishes to reload a data set manually, this can be done by clicking the "Reload data" button on the analysis page. This loads in the data on a separate thread and also shows a notification from the bottom of the screen indicating the loading progress and the estimated time to completion.

This approach lets the user continue using the program normally, but since the data is loaded asynchronously, the programs queries can access this new data as it's being loaded in real time.

Animations

Animations are handled by the Animator class along with the AnimationPage.

When a different page needs to be displayed, the Animator's `switchPages` method is called. This creates a new AnimationPage telling it the two pages involved in the transition along with the direction of the animation. The AnimationPage is set as the current page and handles the drawing of the UI until the animation is complete. The current page of the program is set to the second page and the program resumes normally.

The Animator class is responsible for returning the progress of an animation. The profile used for moving widgets in transitions is a modified version of the cosine wave. This appears smoother than just a linear profile since the start and end of the animations would be too abrupt.

Widgets and Event Handling

All of the special widgets we used in this project are extended from the Widget superclass. With inheritance, we were able to override the methods of the base class so that implementation of the base class method can be designed in the derived class. And this makes our works easier, especially on how we are handling the events of the widgets. While not using the Java 8's built-in event system, we create an Event class to handle all of the events (which is a crucial part in our project). It react with the widget is being pressed by the user and getting different result with the events.(Ex. Initialise the display widgets, switching the page with animation, etc.)

This approach made our page drawing method trivial. The pages stored an ArrayList of widgets and called each one's `draw` method individually.

Conclusion

Overall this project was a great experience for all of us. We got an insight as to how projects are created and managed in the real world, including version control, project deadlines, and communication. We had the chance to work on a “real” project that has a modern user interface and deals with large real-world datasets.