INFS3202
Individual Proposal

Maxwell Bo

March 19, 2017

Chapter 1

Requirements Engineering

1.1 Rationale

TODO Opening esigned to solve a very specific problem borne of my own experience at university.

I noticed that immediately after leaving classes, I was sending similiar message to many group chats across Slack and Facebook, asking if any people wanted to get lunch with me.

Thus, I required a service that would:

- Be aware of when I was leaving classes
- Be aware of my friends timetables, and when they were likely on breaks between classes
- Inform me, immediately on leaving a class, who was free and how long they are free for

My friends also expressed frustration that it was difficult to find times where all members of a certain group had breaks between classes, to coordinate meetups.

They required a service that would:

- Allow them to create groups of people of whom they wished to see shared break timetables
- Be aware of the group timetables, and when they were likely on breaks between classes
- Create a list of free times that the group can meet together

I realized that both of these problems could be addressed by very similiar services.

1.2 Business Function

A client wishing to use the service (henceforth referred to as *SyncUQ* or *'the app'*) would face the following workflow.

- If the client is using a
  - desktop, upon navigating to `syncuq.com.au`, the client is presented the UQ Single Sign-On (henceforth referred to as *UQ SSO*).
  - mobile, using the *SyncUQ* app, the client is asked whether they would like to permit the app to send them push notifications. The client is then presented the *UQ SSO*.
- After signing on with the UQ credentials, a dropdown of timetables, allows the client to choose which timetable is to be imported from `timetableplanner.app.uq.edu.au` (henceforth referred to as *UQ Timetable Planner*). A single `Import` button, allows the client to confirm their selection.

- The client is presented a choice of different tabs, corresponding to different features. Each tab, presents a different workflow.
  - `Friends`
    - The client is presented with a *'friend token'*, which they may send to a present to a friend that they wish to *'follow'*.
    - The client is presented with a field that allows them to enter a *friend token*,
    - The client is presented with a list of friends that they have *followed*. These entries have two forms:
      - *Pending follow*, where the the friend has not approved their *follow request*
      - *Pending follow request*, where a friend has requested to *follow* the client, but the client has yet to approve the *follow request*. Two buttons, a $\surd$ button, and a $\times$ button, allow the client to confirm *follow request*s.
      - *Confirmed*, with a date and time, indicating the instant that both the client and that friend share a break, a *Time until*, indicating the duration in time and minutes until that instant begins, and a *Duration*, indicating the duration of the shared break. Clicking the entry presents a timetable of *breaks* that are shared.
  - `Import`
    - The client is presented with the same screen as *Step 2*, so that they may refresh their chosen timetable, in the event that that they may have modified their timetable on *UQ Timetable Planner*, or a new semester has begun.
  - `Settings`

Whenever a client's scheduled classes are ending, and if the client had permitted their mobile app to send them push notifications, the client will be notified of which of their friends are currently are on, or are starting, breaks. *'Opening'* this notification will present the client the `Friends` tab of the app.

Chapter 2

Architecture

2.1 Development Language and Environment

2.1.1 Backend

Six requirements on the backend language and environment were identified.

1. Could not be Perl, PHP, or Java
2. Should support a mature, stable and frequently used microframework
3. Should support an industry-grade date and time library
4. Should support a stable *iCalendar* (`.ics`) file parser
5. Should be syntactically and semantically familiar to *all* members of the group
6. Should have first-class support or documentation for deployment to Amazon EC2 or Heroku cloud services

A number of considered languages and environments failed to meet these requirements.

- Haskell, using Scotty failed requirements 3, 5, and 6
- Rust, using Rocket, failed requirements 2 (on the grounds that Rocket was immature), 3, 4, 5 and 6
- Clojure, using Compojure, failed requirement 5

Python and Scala met all requirements.
If Python were to be used, it would use

- The Flask microframework
- The Flask-SQLAlchemy database abstraction layer
- The icalendar *iCalendar* parser
- The mypy static type checker
- Heroku, with the Python buildpack

If Scala were to be used, it would use

- Play framework, or http4s interface with circe JSON library

- The doobie database abstraction layer
- The iCal4j *iCalendar* parser
- The cats library for useful functional programing abstractions
- Heroku, with the Scala buildpack

With further investigation of the Python and Scala stacks, both were deemed sufficient for use on the project. Ultimately, three, rather insignificant issues, broke the tie.

1. While all members of the group are familiar with the basics of Python, I have intimate knowledge of Scala and thestack described above. Ultimately, the group decided to ensure that all group members would use a language that they were at least moderately familiar with, rather than a sole individual having a large amount of experience, and the rest having none.
2. Scala requires tooling support to be ergonomic. The group decided that they would rather not burden their laptops limited CPU, memory and battery life with heavyweight IDEs.
3. The group decided that they would prefer to prioritize development speed by using a dynamic, interpreted language, with optional mypy static type checking, rather than deal with Scala's cripplingly slow compilation and typechecking processes.

Thus, the Python stack was chosen.

2.1.2 Frontend

Four requirements on the frontend language and environment were identified

1. Should support a framework that uses a virtual DOM
2. Should be easy to learn for group memebers who have no former frontend development experience
3. Should support an ergonomic Foreign Function Interface (*FFI*) to Javascript
4. Should support the ergonomic use of React components and libraries

A number of considered languages and environments failed to meet these requirements.

- Elm failed requirements 3 and 4
- TypeScript with React failed requirement 2.

Thus, PureScript with Pux was chosen.
Futhermore, the Sass framework to be used is Bulma, chosen on aesthetic and usability grounds.

2.1.3 Deployment

The app is to be auto-deployed to a Heroku staging environment, triggered by a GitHub push hook. Manual deployment, environment configuration, and debugging facilities can be accessed with the Heroku CLI.

The use of Git Submodules, with the Heroku Submodules Buildpack allows the frontend and backend repositories to be developed in seperate *Git* repositories, and pulled together during the deployment process.

2.2 Integration

Ajax with JSON will mediate client-server communications, with server responses adhearing to the Google JSON Style Guide.

Endpoints that may need to be implemented

TODO Something about postgres

TODO Rest API, no server side HTML generation. Better decoupling between front-end and back-end. Permits implementation of dynamic SPAs with front-end virtual DOMS. Permits the creation of native mobile apps with react native or some shit later on (although we aren't doing that, you kinda need a REST API for that)

Chapter 3

Design

3.1 UI / UX

3.2 Endpoints