INFS3202
Individual Proposal

Maxwell Bo

March 19, 2017

Chapter 1

Theory

1.1 Rationale

Designed 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 requests*.
      - *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.

1.3 Development Language and Environment

1.3.1 Backend

I identfied 6 of requirements on the backend language and environment.

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 it was immature), 3, 4, 5 and 6
- Clojure, using Compojure, failed requirement 5

Python and Scala met all requirements. Further environments were investigated.

Python 3.6

Priorities - Mature microframework - Mature datetime libarries - Decent support for icalendar file format parsing

- All members of the group know it - Optional typing system with the typing https://docs.python.org/3/libra and mypy TODO: href (as I have experience with typed functional languages) - Good support for microframeworks (not Ruby)

Framework: Flask. Not a bigass framework that's confusing and full of footguns, unnecessary garbage and opinionated methods of doig stuff. Promoted by Heroku as the framework of choice to use with Python -¿ going to be a lot of documentaton regarding this stack when things go bad

SQLAlchemy

ical framework looked mature as well

other options considered

Clojure with compojure -¿ hugo knows some clojure, I'd be willing to learn some. However we were worried that if we got up shit creek clojure would make it very painful to punch our way out considering its functional nature

datetime and ical are obviously very good due to being on the JVM

Rust with Rocket framework. Inferior ical framework. Borrow checker seemed shitty.

Scala. I have a lot of Scala experience. Hugo and Charlie have Java experience so that knowledge was gonna carry over.

Options - Play framework, not microframework - HTTP4S, Doobie (database abstraction layer) (type system can get fucked up), Circe JSON encoding (typesafe, pretty cool magic), Cats so Scala wouldn't suck

Heavy, cripling build time, unfamiliar stackt the majority of the group, IDE support required (would rather not)

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)

Frontend

wanted to use a functional compile-to-js language, because fuck javascript

Understood that some of the solutions and young and immature, but we're hipsters so we want to do that (we're using a tried and tested solution in the backend, lets have some fun in the frontend)

- Escape hatches: import React components, decent Javascript FFI

ClojureScript and https://reagent-project.github.io/ (dismissed because difficult to use syntax, hugo didn't want to) Elm (dismissed because JavaScript FFI sucks balls, react integration too) PureScript with Pux (good javascript ffi, good react integration)

Chapter 2

Design

Chapter 3

Feature Coverage