Pack Manager

CS130: Software Development
Computer Science Department
University of California, Los Angeles
Spring 2015
Git repository: `https://github.com/asanciangco/PackManager`

Alex Sanciangco
Nick Westman
Steven Holtzen
Mitchell Binning
Kacey Ryan
Heather Blockhus

Contents

1 Introduction and Motivation

Pack Manager is a tool for helping you pack. It gives you a wardrobe list based on the weather of where you're going. It takes your (1) location, (2) time of travel, (3) personal dressing preferences, and gives you a set of clothes for you to wear and necessities that you can't forget. The goal is to make packing for an outing simple and fast. The application will have a very clean user interface; since its function is simple, its use must be equally simple. The application must be extensible, which means that adding new features must be a well-defined process that is easy.

1.1 User Benefits

Some examples of when a user may benefit from using our application is if they are frequent travelers. It is very difficult to track what to bring when traveling, especially if travel arrangements frequently change or users travel to many locations in sequence. Our application can update them on changing weather patterns for arbitrarily complex trips spanning many days and destinations.

Another example is someone traveling to an area with which they are unfamiliar with its weather patterns. This makes it difficult to predict what would be necessary to bring. This user would simply enter their information into our application, which would automatically keep them up to date regardless of whether or not they know about monsoon season in Indonesia.

2 Feature Overview

When the user first starts the application, he or she will be presented with a login screen where they can choose to log in with Facebook. Once logged in, the user will enter their preferences, at which point the application is ready to make packing suggestions. Once this initial login is complete, the preferences will be assigned to their account, making future trip planning seamless. A packing list is suggested after the user enters their destination, which can include multiple stops, date of departure, duration, and takes into account preferences and trip parameters. From the "My Trips" tab, the user can view and modify this packing list. The application will notify the user in case of weather changes; this will require updates to the packing list.

2.1 Requirements

2.1.1 General Requirements (G)

1. Able to pull relevant weather information from Open Weather Map for arbitrary dates. (Related: G2, G4, G6, API1, API3, API4, UX7, UX8)
   (a) If no projected weather data, pull historical.
2. Provide a framework for accepting and using user preferences. (Related: G5, G8)
   (a) User preferences must be consistent between sessions.
   (b) Must be able to export preferences.
   (c) Plan for eventual transition to decentralized storage (move to cloud).
3. Create packing lists for vacations of arbitrary durations. (Related: G7)
4. Provide a framework for trip preferences. (Related: G2, UX2)

   (a) Availability of laundry machines Level of cleanliness ("I don't care about wearing underwear 13 times")
   (b) Specific activities Purpose (business, leisure, etc.)
   (c) These options affect the packing lists but are optional with customizable defaults.

5. Sends user a push notification if weather changes in destination will affect packing list, or a critical weather alert is issued. (Related: G1, API4)

6. Ability to modify already created packing list. (Related: G4, G8)

   (a) Add items, remove items, reevaluate (pull weather again)

7. Ability to create an itemized list of clothing and other essentials for travel. (Related: G2, G7, UX1, UX4, UX6, API2)

   (a) List can be customized after suggestion by user.

8. Allow the user to login through Facebook. (Related: API3, API6)

2.1.2 User Experience (UX)

User Experience:

1. 3 user input (minimum) to get to a packing list: start date, duration, location. (Related: G5, G8)

   (a) Additional options available if necessary
   (b) Duration must be nonnegative
   (c) Destination: zip code or city, state/country

2. Prompt for user preferences on first launch. (Related: G3, G5)

3. Obtain user feedback after a trip to adjust user temperature preferences. (Related: API4)

4. User can store packing lists upon save (Related: G8, UX5, API2)

   (a) Possibly autosave

5. Export lists to other apps (reminders, calendar) (Related: UX4)

6. Packing lists formatted as a table. Each row composed of item and quantity (e.g. T-Shirt x3, Socks x4, Suntan Lotion x1, Umbrella x1) (Related: G8)

7. Error Reporting: If network unavailable, prompt user for solution (Related: API1, API4, G1)

8. Error Reporting: No relevant weather information for provided destination (Related: API1, API4)

2.1.3 API Interactions and Extensibility (API)

1. Application has a single point of failure for interacting with external APIs (Related: G1, API3, API6)
2. Packing and destination information can be exported in JSON format (Related: G8, UX4)
3. Build a wrapper around Open Weather API and Facebook for easy API usage (Related: G1, G9, API1)
4. Build a push notification framework for communicating with users in a device-agnostic way (Related: G6)
5. The system must have the capability to be localized (allow for arbitrary languages and character sets to be used)
6. Facebook API is used for login and identity management. (Related: G9, API1)

2.1.4 Non-Functional Requirements (NF)

1. Test coverage and automated building
2. Free to download
3. Security Outside
   (a) user cannot see your packing lists
   (b) Outside users cannot modify your packing lists through any API
4. iOS Application targeted for iOS 8 and above

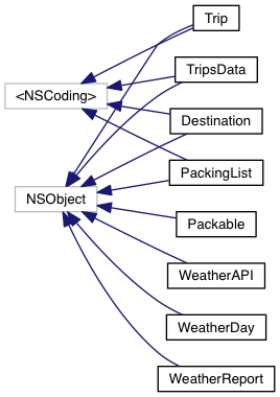
3 Design

The application is structured using the model-view-controller architecture dictated by the iOS development framework. In an iOS application, the overall flow is defined in storyboards. These files define transitions between panes, hold content. The main storboards for our application is in the `Main_iPhone.storypad` file. The storyboards are the views in this framework.

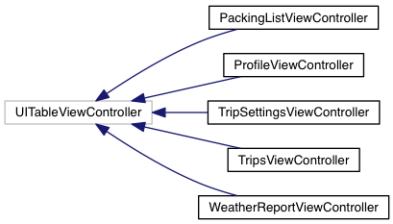
The controllers can be found in the `Controllers` subdirectory. These files dictate the flow of control within the application; they are the glue between the storyboards and the models, which define how data is stored and manipulated. By carefully separating the concerns between storing and loading data from the user interface logic, we get a clean modular structure that allowed Heather and Alex to both work simultaneously on different aspects of the application.

The Weather API is an important and very tricky component of our application. We currently use Open Weather Map[1] in order to gather weather data, which gives us a 16 day weather forecast for a particular geographic coordinate. This gives us enough data to create a packing list and act as a placeholder. In the future we hope to use the NOAA weather API[2], which includes historical data and can allow us to make coarse-grained projections over a much greater timespan and in a larger area.

[1] See `http://openweathermap.org/`
[2] See `http://www.weather.gov/`



(a) Class diagram of the models. An `NSObject` is the root object in Objective-C; the `NSCoding` interface allows a class to be encoded (serialized).



(b) Class diagram of the controllers. Each controller must inherit from the `UITableViewController` class in order to interface with the storyboards.

Figure 1: General class hierarchy for Pack Manager.

3.1 Class Diagram

The general structure of our classes can be found in can be found in Fig.1. This figure lays out the general class hierarchy for quick reference. See Fig.2, Fig.3, Fig.5 for an overview of our class design decisions.

3.2 Clothes Selection Algorithm

In order to generate a list of items one should bring on their trip we needed to come up with an algorithm. The packing algorithm would have to account for the trip's duration, each day's weather, and a number of user preferences (i.e. re-wearing jeans, access to laundry, their hot cold preferences, etc). Our algorithm first initializes a packing list, checks each day's weather and appropriately increments clothing, and finally produces estimates based on user preferences. The initialization creates a packing list that contains the bare essentials such as toiletries. Additionally, the initialization will check if the user has marked if they will be swimming or need formal attire. If they checked swimming then it will include swimwear as a function of days. Next, for each day in the trip the algorithm calls several incrementing functions, one for each type of clothing (tops, bottoms, etc). Each of these incrementing functions have a simple interface that hides its implementation details from its caller, exercising the information hiding principle. Each function checks the day's average temperature against the user's temperature preference and perform different clothing increments based on what the user thinks of that day weather (hot, warm, normal, cool, cold). Finally, now that the algorithm has produced a maximal list, it checks the user's preferences for things like access to laundry and rewearable jeans and reduces the list's numbers. For example, if laundry is checked it will cap the max number for each type of clothing article. Similarly, re-wearing jeans would cap just the bottoms category.

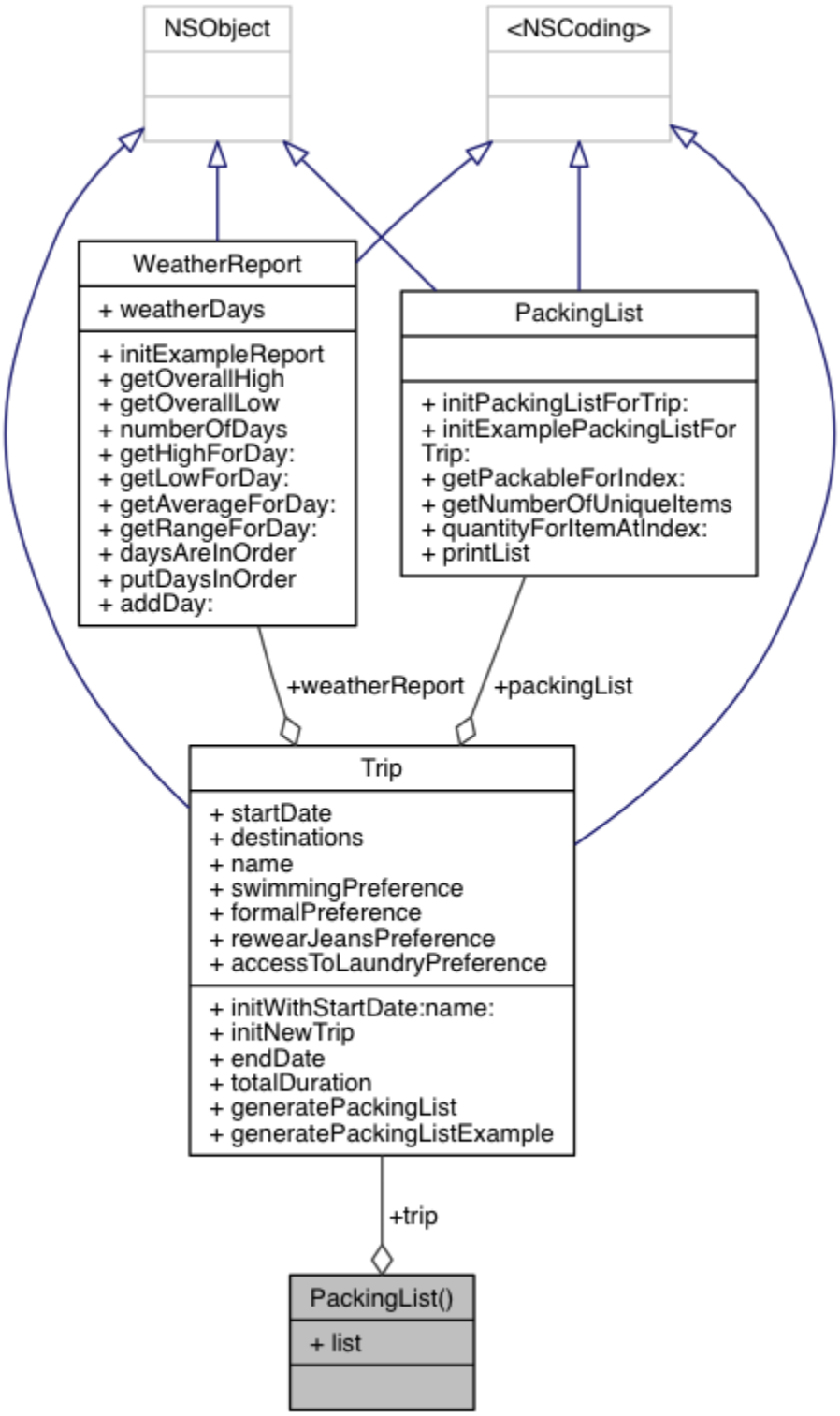


Figure 2: The `Trip` class is the core structure for holding details related to an individual trip. Currently trip attributes are stored as booleans within this structure; we have plans to make these easier to change in the future.

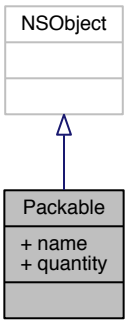


Figure 3: The packable interface defines the basic methods that are necessary to make an individual object packable. All articles of clothing will inherit this interface; it will likely need to be expanded as more required functionality is discovered.

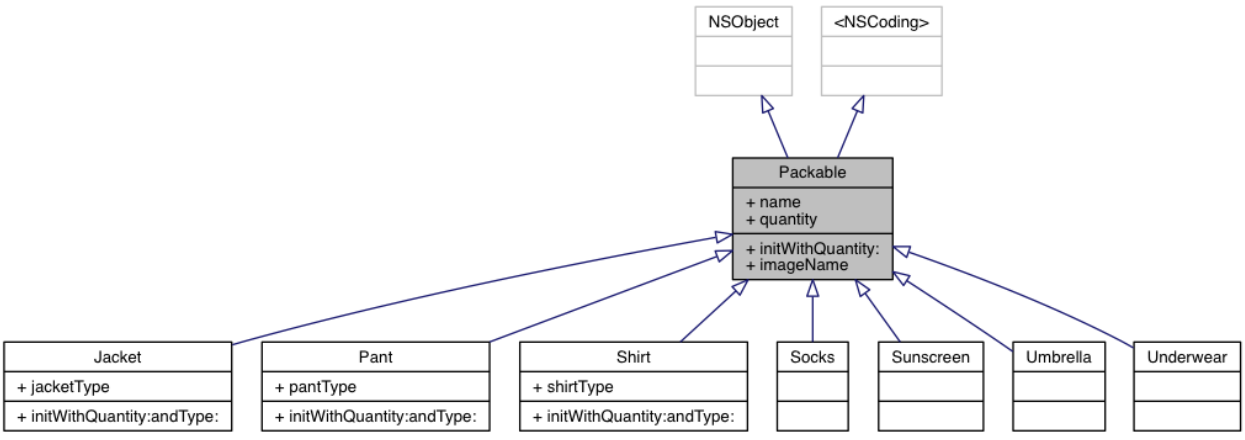


Figure 4: Packing lists are made up of packables and this diagram shows the packable class and the multiple types of packable items.

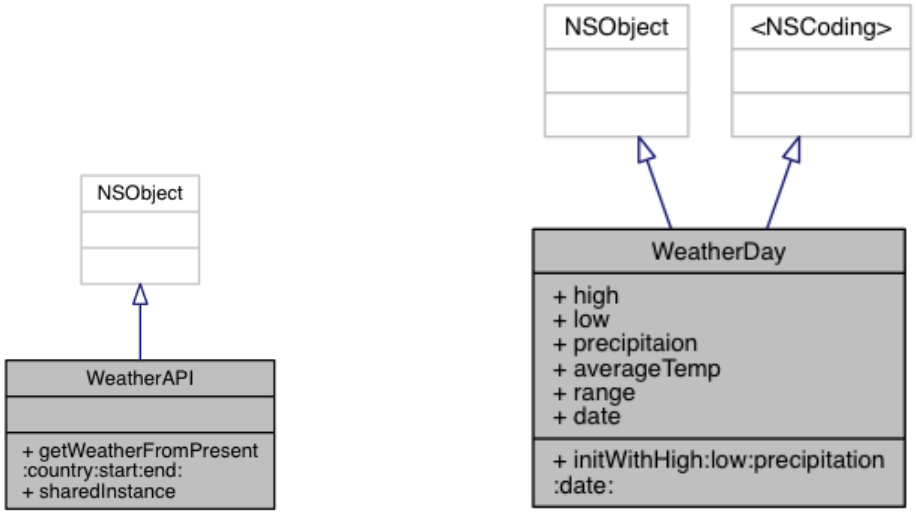


Figure 5: As shown in the first figure, the trip takes into account the weather report, which is generated by the Weather API shown above. Additionally the weather report consists of weather days, shown above.

4 Quality Assurance

4.1 Testing

We set up a test harness using Travis CI[3]. Travis CI is a continuous integration suite that automatically runs our tests whenever a pull request is performed on Github. Before a change is merged all the tests are run in order to guarantee that all code that gets checked in passes all the tests. See Fig.6 for an example Travis CI output.

Travis CI configured in the `.travis.yml` file, which specifies the language framework as well as the test harness. Our tests can be found in the `PackManagerTests` directory. We are cataloging our test coverage manually; at the time of this writing there are minimal tests, but it is our goal by the final project to have 90% branch coverage.

4.2 Change Analysis

Some potential change scenarios we considered were:

1. Porting the project to other mobile operating systems like Android. This would be a very large-scale transition and would require rewriting much of our code, but the essential components would be reusable. The user preferences and trip objects are serializable in a generic way (including XML or JSON formats), allowing for cross-device synchronization.

2. Once a packing list is automatically generated, a user may want to further refine the automatically generated suggestions by providing additional input. This would require some additional user interface, like an edit pane, as well as a flexible packing list object that allows for both automated and user-generated content.

[3]See `https://travis-ci.org/`

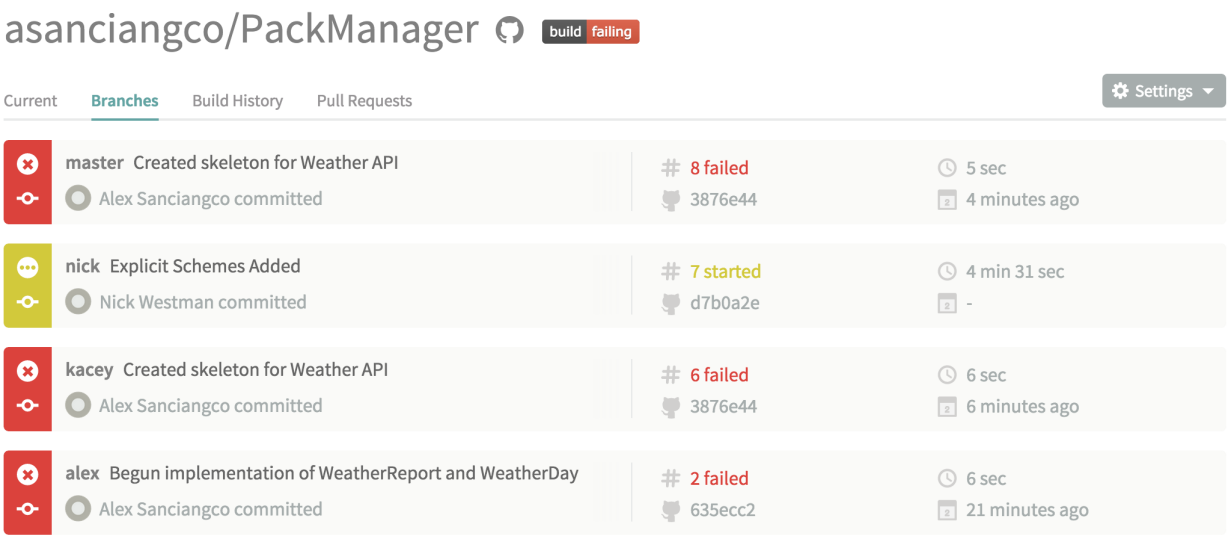


Figure 6: Output of Travis CI. It shows the status of all branches as well as whether or not they are currently passing the tests that are specified in `.travis.yml`. This will prevent ever merging a branch that does not pass tests, which will prevent regressions.

3. User accounts could be useful for synchronizing information across many devices or even a web service. Due to how we serialized the essential information our application produces in a generic way, these can also be supported.

4.3 Documentation

For documentation we are using Doxygen[4]. The documentation is automatically generated whenever we build by Travis CI. It includes automatically generated UML documents (as seen in iour Class Diagram section), as well as parsing descriptions for classes, parameters, returned values, and properties. We simply include a small specially formatted comment before the desired code block. These formatted comments integrate with Xcode, allowing us to refer to our documentation as we write our code, further decreasing the chance of errors.

Feel free to browse the documentation in our git repository, which can be found at `docs/html/index.html`.

5 User Interface

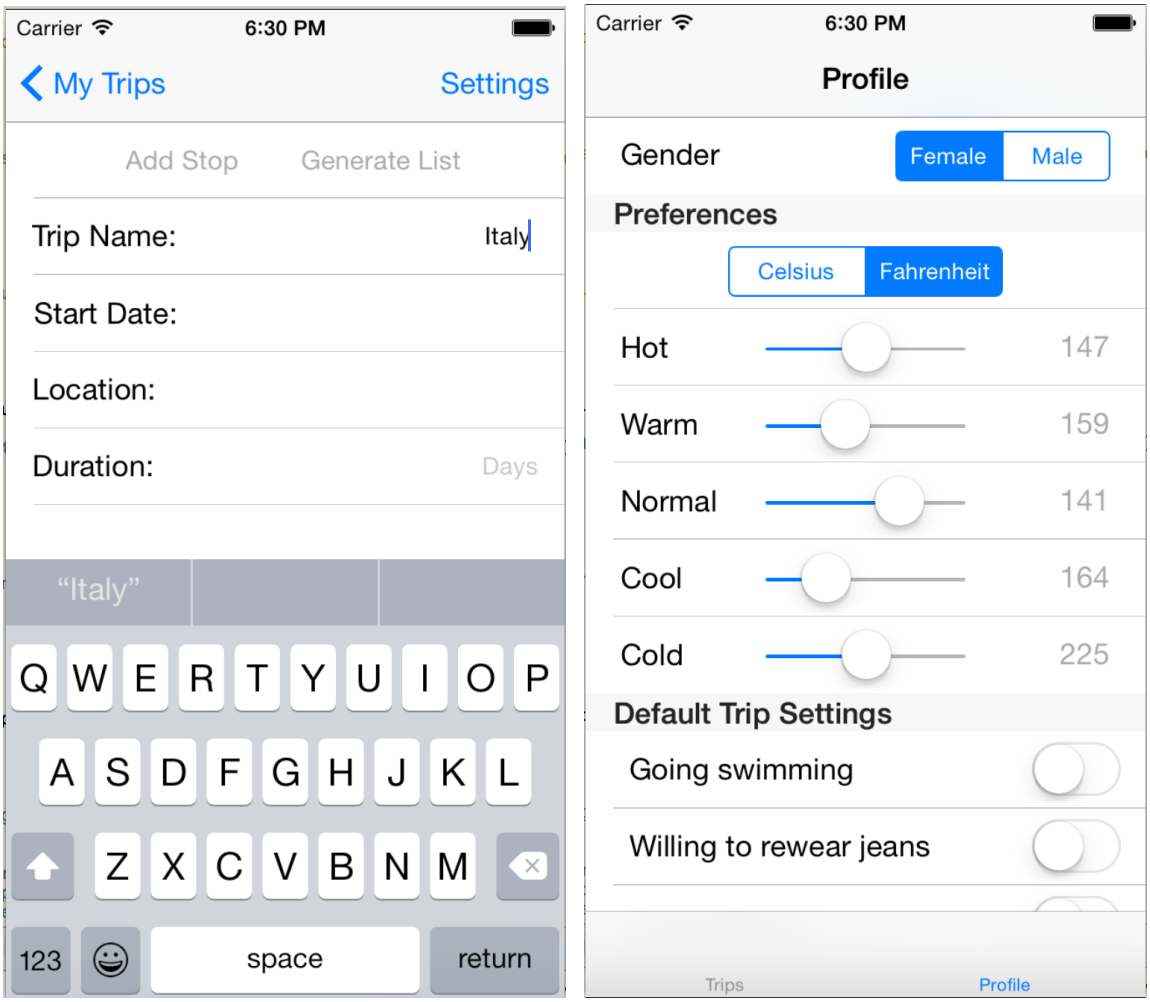
See Fig.7 for some screen shots of the current state of the user interface.

6 Contributions

1. Heather worked on the user interface. She worked on developing flow of the application, working closely with Alex. Heather is experienced with designing user interfaces for iOS.

2. Alex is the lead programmer. He worked on integrating Heather's user interface with Kacey's weather API, as well as being a source of knowledge on general iOS development and Objective-C, which many group members were not familiar with.

[4]See `http://www.stack.nl/ðimitri/doxygen/`



(a) The trip designing frame. Here we see the various options a user can select while making a trip.

(b) The user profile creation screen. There are many options (some not shown here).

Figure 7: The two main user interface panes designed so far.

3. Steven worked on writing the report and the documentation. He added Doxygen and used LaTeXto write the paper.

4. Kacey worked on the weather API, doing research and wrapping it to make it accessible and easy to use.

5. Mitchell worked on the presentation as well as general programming tasks.

6. Nick was in charge of build management, integrating Travis-CI and managing the test infrastructure.

7 Conclusion and Future Work

So far we have completed the major skeleton of our project. The main user interface elements, including making a trip, filling out a user profile, and selecting your current trip, are completed.

Our future work includes developing the algorithm for actually choosing which clothes to include in a packing list. We plan to use a simple rule-based algorithm based on user preferences; for example, every day has a minimum set of clothes that are required (at least one complete outfit), there are certain clothes that must always be included in a packing list (like socks and underwear), and there must be spare clothes depending on the length of the stay. The rules will be defined in an extensible way that will allow for easy fine-tuning.

We would also like to implement an edit feature so that users can change trip settings from the application. This will require an additional UI panel and scaffolding.