WordUp

Product Guide

Henry Stolz – Jacky Kong – Luis Legro

Table of Contents

0 – Overview

0.0 – Team

Henry Stolz* – hstolz
Jacky Kong – jacky
Luis Legro – llegro

0.1 – Description

WordUp is an iOS app providing a platform by which to connect with others, schedule a personal meetup, and practice oral skills in any foreign language supported by Princeton University's foreign language programs. The first portion of the following guide will show you how to set up the app and take you through its basic usage. The second section will provide an intermediate-detailed view of the internal design of the app for those with a more developer-oriented background.

1 – User Guide

1.0 – Installation

To obtain the code needed to run the app, email Henry Stolz (hstolz@princeton.edu) to receive access to the private Github repository. After downloading a copy of the app source code (Clone or download > Download Zip), unzip the downloaded archive. The full source code will be available to be opened from Xcode, an IDE provided by Apple. If not yet installed, Xcode can be found and installed from the App Store. Open the `WordUp.xcworkspace` file In Xcode. Follow the instructions on for CocoaPods installation provided in section 2.1.1 of this guide.

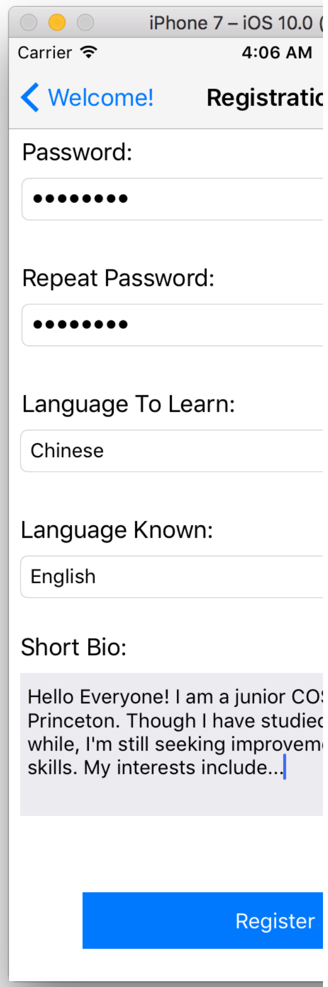
Once complete, click the Build and Run the Current Scheme button on the top left hand corner to open the app in the Xcode simulator. Ensure that a simulator (iPhone 7 preferred) is selected; if the only option shown is "Generic iOS Device," change the deployment target (WordUp > General > Deployment Info > Deployment Target) to 10.0. Once this step is complete, you should be able to simulate the app on various iOS devices.

0.1 – Quick Start

Navigate to the app's location on your iOS device and touch or click on the app to open it. When you first open the app, you can **Register** a new account and **Login** to access your existing account.

If you are using WordUp for the very first time, tap **Register** and provide all the information required on the registration page and an optional bio. By describing yourself, your language goals, and so forth within your bio, users can get a better idea of who you are and what you two might chat about should they choose to match with you.

If you already have an account, you can log in using the username and password you selected when you first registered.



Password:
••••••••
Repeat Password:
••••••••
Language To Learn:
Chinese
Language Known:
English
Short Bio:
Hello Everyone! I am a junior CO
Princeton. Though I have studie
while, I'm still seeking improvem
skills. My interests include...

Registration Page

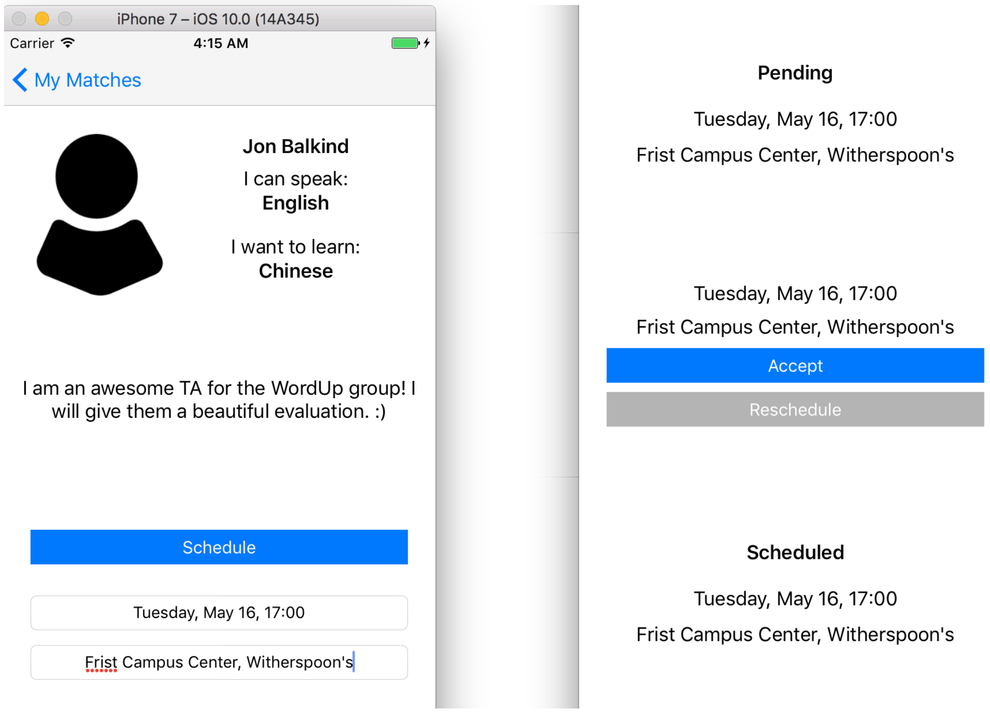
0.2 – Interface and Operation

When you first login to your account, you will be taken directly to the **My Profile** page, which shows all information specific to you. When you are done using the app, you can choose to logout from this page as well. Throughout the app, you can always navigate to the **My Profile**, **Matches**, and **Search** pages with the navigation bar found at the bottom of the screen.

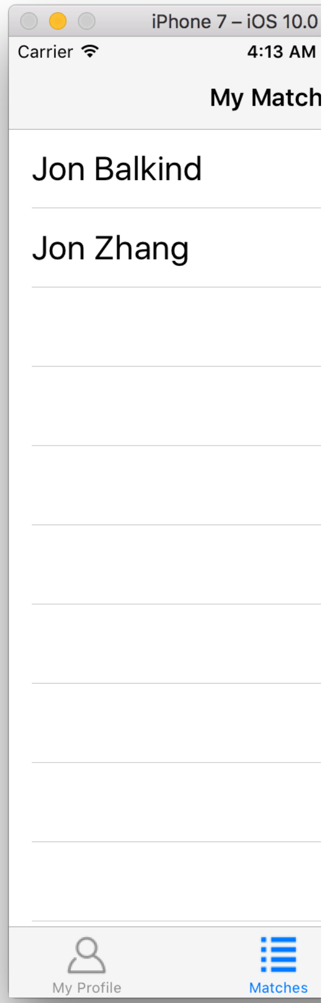
In the **Search** page, you can see a list of the people you have yet to match with and whose language preferences match your own (i.e., know the language you are learning and are learning the language you know). You can tap to view their profile, especially their bio, and choose whether to match with that specific user. If you have a hard time deciding, you can always tap the 'Random Match' button at the top right edge of the screen. Once matched, you will then find the user in the **Matches** page.

In the **Matches** page, you can see a list of the people you have already matched with, tap their entry to view their profile, and see at what point in the meet-up process you are in with this particular user. You can 'schedule' a meetup, after which your offer will be 'pending' as you wait for your partner to accept. As the receiver of an offer, you can choose to 'accept' the proposed meetup or give a 'counteroffer' if the proposed time or place does not work for you. Once the offer is accepted, the two parties are 'scheduled' and can access the meetup information at any time. Once the meeting time is passed, the meetup resets and one party can suggest a new time to meet.

By thusly limiting the actions a user can take to interact with their language partners, we to encourage users to meet up and get to know each other in person! Other apps give functions such as text chat and in-app translation, but we feel those features only hurt users in the long run. With that said, all of us on the WordUp team hope you enjoy using our app!


iPhone 7 – iOS 10.0 (14A345)
Carrier
4:15 AM
My Matches
Jon Balkind
I can speak:
English
I want to learn:
Chinese
I am an awesome TA for the WordUp group! I will give them a beautiful evaluation. :)
Schedule
Tuesday, May 16, 17:00
Frist Campus Center, Witherspoon's
Pending
Tuesday, May 16, 17:00
Frist Campus Center, Witherspoon's
Tuesday, May 16, 17:00
Frist Campus Center, Witherspoon's
Accept
Reschedule
Scheduled
Tuesday, May 16, 17:00
Frist Campus Center, Witherspoon's

Possible Meetup Stages



Matches Page

2 – Developer Guide

2.0 – Technologies

Latest Revision – Version 1.1, 2017-05-13
Build Requrements – Xcode 8.0.0+
Runtime Requirements – macOS 10.11+, iOS 10.0+

Technology Stack – Swift 3.0, Django 1.9, Django REST framework 3.6, MySQL 1.2.5, Google Cloud Platform

2.1 – Dependency Management

WordUp uses CocoaPods, a dependency manager for Swift and Objective-C Cocoa projects, to manage the third-party libraries in use in the app, such as Alamofire.

2.1.1 – Installing CocoaPods

We provide the pod files as part of our product package, but to allow Xcode to simulate the app, you must first install CocoaPods as follows. In Terminal, enter:

```
$ sudo gem install cocoapods
```

Enter your system password as requested and allow the installation to complete. If simulation still does not proceed properly, please click on the top directory of the project in Xcode and change the Deployment Target under Deployment Info to 10.1. Under normal circumstances, you would need to go to your project directory and enter the following command:

```
$ sudo pod init
```

This initializes what is known as a "Podfile". This is as Ruby file where you would configure the dependencies you would like CocoaPods to install and manage for you.

In our case, we have already provided a Podfile for our application. Thus, in the project directory you must enter the following command to properly install the dependencies from a podfile:

```
$ sudo pod install
```

Next, click on the schemes menu next to the Stop Running Application button in the top left corner of Xcode and click manage schemes. Make sure everything is checked and Alamofire is present. The simulation should now work.

2.2 – Backend

WordUp uses Django and Django REST framework for all its data management functionalities with Google App Engine as the host site for all schemas. Django was chosen for its relative simplicity in manipulating relational data, for its provided security features, namely the token-based HTTP Authentication scheme appropriate for mobile clients and its integration with the extendable user model, its relatively abundant documentation, as well as for its compatibility with the Google App Engine (up to version 1.9 as of May 2017).

Recommended reference + reading:

```
docs.djangoproject.com/en/1.11/
docs.djangoproject.com/en/1.11/topics/db/queries/
django-rest-framework.org/tutorial/quickstart/
```

2.2.0 – Database Structure

Our database stores two models as defined in `models.py`. An example entry of both is shown below, in JSON for readability:

Profile:

```
{
     “id”:“1”
     “username”:“hstolz”
     “password”:“password” # salted + hashed
     “first_name”:“Henry”
     “last_name”:“Stolz”
     “known_lang”:“en”
     “learn_lang”:“zh”
     “bio”:“Hi everyone! I am...”
}
```

Match:

```
{
     “match_id”:“1”
     “user_id1”:“1” # foreign keys
     “user_id2”:“8”
     “time_1”:“2017-05-20 18:30:00 +0000”
               # datetime object
     “location”:“Small World Coffee”
     “status_code”:“1”
}
```

2.2.1 – Model Implementation Details

The definition of models is relatively straightforward, with much of the logic behind the implementation readily apparent by viewing the model field definitions in `models.py` and reading through the Django documentation.

The Profile model extends Django's AbstractUser class. This gives access to functions like `authenticate`, `login`, and handles password salting+hashing, facilitates token creation, and enables the login requirement for access of certain views.

One area of note is the use of `status_code`. Value 0 indicates the match is created but currently no meetup is scheduled. Value 1 indicates that the Profile `user_id1` proposed a meetup but the Profile `user_id2` has yet to accept (and vice-versa for 2). Value 3 indicates that the meetup has been succesfully scheduled.

2.2.2 – View Implementation Details

To the right, you can see the endpoints provided by our API and their allowed HTTP methods. Most are self explanatory, with the exception of `/token/`, which is explained below, and `/times/`.

The latter gives the existence/status of a match by providing a Profile `id` as opposed to Match `match_id` as is the case with the `/matches/` endpoint. It is primary used in showing your match status with another user in the profile view of that user.

All methods with PUT and POST require the parameters to be passed as JSON. `/matches/` will create a match with a specific user if `a_username` is specified, otherwise it will randomly match you with an eligible user. `/matches/#/` is the main endpoint used in making offers, counteroffers, and confirming meetups.

Querying of the database is done not with raw SQL commands but through Django's query abstractions. For instance, `Match.objects.filter(time_1__lte=t).update(status_code=0, time_1=None)` will select matches satisfying the given time command and update the specified fields, all in one statement.

API Outline –

`/profiles/`
`/matches/`
GET, POST

`/profiles/#/`
`/matches/#/`
GET, PUT, DELETE

`/times/#/`
GET, PUT, DELETE

`/login/`
POST

`/logout/`
POST

`/token/`
POST

`/register/`
POST

2.2.3 – Security

Every HTTP request to the Google App Engine server requires that the client authenticate by including an Authorization HTTP header that is created in this format by the Django Rest framework TokenAuthentication scheme, example value below:

```
Token: 9944b09199e62bcf2918ad846dd0d4bbdfc6ee4b
```

Whenever a user is registered, their information is stored, password is salted and hashed, and django generates a semi-permanent token for that user. From then on, whenever the user logs into the app, they provide their credentials to /login/, which authenticates their credentials against those stored in the database and marks them as logged in. The token endpoint also authenticates the credentials and provides the user with their token. Every request the user makes includes this token in the header; failure to do so will result in `HTTP 401 Unauthorized` response. Attempts by logged-in users to access, generate, or alter matches that they are not members of will result in an `HTTP 403 Forbidden` response. Once a user logs out, the user information and token are cleared from the app.

2.3 – Middleware

WordUp uses Alamofire, a Swift-based HTTP networking library for iOS and Mac OS X, to create RESTful web services that link the backend and frontend within the Xcode development environment. We chose Alamofire because of its easy integration with Swift and with the aforementioned security features provided by the Django REST framework, since it can easily be made to require a token-based authentication header with the "header" parameter of all Alamofire request functions.

2.4 – Frontend

2.4.0 – Assets

We use generic images for our menu items, which include search, profile, list and default profile image. These can be found under the WordUp file hierarchy in `Assets.xcassets`, along with all other aspects of the UI.

2.4.1 – Storyboard

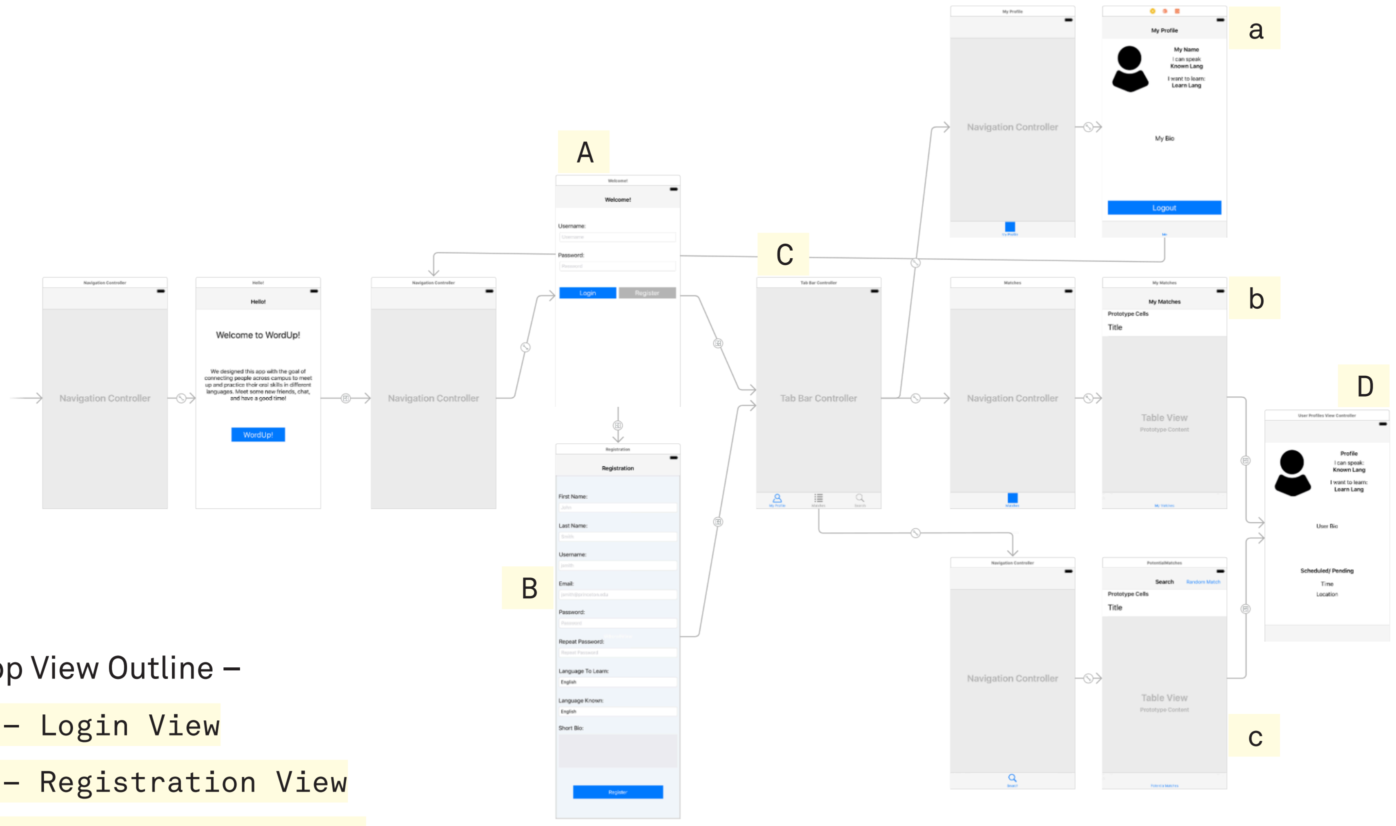
The app's workflow is outlined in `main.storyboard`, which allows relatively easy implementation of a Tab Bar Controller that permits navigation using an easy-to-use tab bar found at the bottom of all three UIViews in the app. This way, no matter what feature the user is in the middle of using, all other features can be immediately accessed.

In both of the two UITableViews, item can be touched or clicked; doing so provides a UIView that shows a more detailed view of that item. All transitions between the different views are implemented using "segues." Some segues, such as the ones between the two TableViewControllers and the UserProfilesViewController that provides the profile detail, are manually activated; this allows for the easy transfer of user information and quick generation of user data. This also prevents lag between the two views. All other segues exhibit default behavior without any programmatic intervention in the source code.

We also use Navigation Controllers to enable the display of a Back button on the Top Bar in certain situations. For instance, a separate one is provided for each view in the Tab Bar so that the user can navigate back to the relevant TableView after entering a User Profile View.

2.4.2 – Tabs

The **My Profile** tab provides all relevant account information for the user including their username, their chosen language to learn, their chosen known language, and a bio, a free-form text description of the user and their language goals. These are UILabels whose text is dynamically set using Alamofire requests. This view also provides a Logout button, which is a simple UIButton with an attached function, providing the erasure of user data upon the end of a session. Finally, the view provides an ImageView that defaults to the image included in our assets folder.



App View Outline –

A – Login View

B – Registration View

C – Tab Bar Controller

a – My Profile View

b – My Matches Table View

c – Search Table View

D – User Profiles View

Xcode Storyboard View

The **Matches** tab provides a UITableView of all the matches relevant to the user. Tapping each matched profile reveals a UIView that shows the details of that user profile. The matched user information is also provided with simple UILabels.

The **Search** tab provides a UITableView of all the potential matches for that user. A potential match is defined as a user who knows the language you are trying to learn and who is learning the language you know. The main purpose of this tab is for the user to be able to search specific other users they have in mind, such as friends or language teachers on campus that need a platform to schedule their oral interviews. Additionally, the **Random Match** button provided at the top of the search UITableView randomly generates a match amongst the potential matches for that user. This is a core feature of our app since, apart from allowing users to practice their language, we also aim to facilitate new connections amongst app users with similar interests. Once this button is touched, a request will be sent to the appropriate endpoint and Django generates a random match as described in View Implementation Details.

2.4.3 – User Profile

Both the Search and Matches tabs are connected to a **User Profile** UIView, the main area of interaction between users.

If accessed from the Search tab, a Match Us UIButton is provided within the Make Match UIView. When tapped, it sends a request for the two users to be added to the matches schema. Once this request is processed, a user can interact with this now-matched user in the Matches tab.

If accessed from the Matches tab, a number of variations on the view may be scene. To implement the scheduling functionality described in sections 0.2 and 2.2.1, we have a group of 4 nested UIViews. The highest in the hierarchy shows the Make Match view as described in the above paragraph. Nested below this are the Schedule View, Accept Reschedule View, and Scheduled Pending View. Depending on the status of the match between us and the displayed user, one of these views will be pulled to the top of the hierarchy to be displayed.

In case there is no current meetup, the Schedule View is shown; users can input a time in a customized PickerView, the value of which gets formatted by two dateFormatters, one for display and one to be sent to the backend. They also provide a location within a TextField.

In case we initiated the meeting and are waiting on a response, or if the meeting has been scheduled, we display the proposed or agreed-upon time, respectively.

In case a meeting has been proposed to us, the Accept Reschedule View is shown. Accordingly, the proposal can either be accepted, which finalizes the meetup, or another offer can be made. In the latter case, the Schedule View is displayed.