Wingit Docs

So you *really* want to wingit? Just follow these easy instructions!

Installation

Wingit consists of a number of separate components which must be setup and installed individually before they can be linked together to form the awesomeness that is Wingit™ . These components are:

Overall:
- An amazing team (absolutely necessary)

Server:
- Ubuntu 14.04.4
- Apache

Website:
- Python Packages
- Django web framework
- MongoDB
- Microsoft Azure Interface

Application:
- Android Studio

Note: all file paths are relative to the directory which this was extracted to
Furthermore, these files are assumed to be extracted to `/var/www/html` on the server once the server is set up.

Server:

Ubuntu:
Install Ubuntu 14.04.4 server edition. This can be found at the following web address
http://releases.ubuntu.com/14.04/ubuntu-14.04.4-server-amd64.iso.torrent
After installation, set up SSH and configure port forwarding as necessary. If you have a domain name, forward the A record to your IP.

Apache:
In order to install apache, which is the web server itself, run the following commands:

```
$ sudo apt-get update

$ sudo apt-get install apache2
```

WSGI:
Django applications are served through the mod_wsgi (web server gateway interface) Apache module. In order to set up mod_wsgi, run the following:

```
$ sudo apt-get install libapache2-mod-wsgi
```

The default virtual host file used by apache needs to be modified:

```
$ sudo nano /etc/apache2/sites-available/000-default.conf
```

Add the following lines to the file, within the `<VirtualHost *:80>` tag

```
<VirtualHost *:80>
    ...

    Alias /static /var/www/html/wingit/static/static_root

    <Directory /var/www/html/wingit/static/static_root>
            Require all granted
    </Directory>

    <Directory /var/www/html/wingit/wingit>
            <Files wsgi.py>
                    Require all granted
            </Files>
    </Directory>

    WSGIDaemonProcess wingit
python-path=/var/www/html/wingit:/usr/local/lib/python2.7/dist-packages
    WSGIProcessGroup wingit
    WSGIScriptAlias / /var/www/html/wingit/wingit/wsgi.py

</VirtualHost>
```

These instructions are adapted from DigitalOcean's tutorial on how to serve django applications. [1]

[1] Ellingwood, Justin. "How To Serve Django Applications with Apache and Mod_wsgi on Ubuntu 14.04." 18 Mar. 2015. Web. 26 May 2016.

Site:

Django:

What you need to enter in the terminal to set this up:

```
$ pip install -r pip-requirements.txt
```

AKA the "I ain't got time fo dis" option.
Feel free to skip to the MongoDB section below.

In excruciating detail:

Python 2.7.11, and specifically the included pip module is used to install django, mongo, and other modules.

All of the packages that need to be installed can be found in /pip-requirements.txt
These can be installed individually, e.g.:

```
$ pip install django==1.9.5

$ pip install django-crispy-forms==1.6.0

...
```

Or they can all be installed in one command e.g.:

```
$ pip install -r pip-requirements.txt
```

What's being installed:

Requests
: Standard python library for the handling of HTTP POST/GET requests

Django version 1.9.5
: Base installation of the Django web application framework which serves as the backbone of the wingit site

Django Crispy Forms
: Sexy rendering of forms on html pages with integration for bootstrap3

Django Registration Redux
: Registration backend, which handles user registration, login, and logout

MongoEngine

Interface between Django and MongoDB

PyMongo

MongoEngine dependency, interface between python and MongoDB

Azure

Azure libraries in order to interface with Microsoft Azure, where our machine learning is done

MongoDB:

In order to install mongodb, the following sequence of commands need to be ran:

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

$ echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc)"/mongodb-org/3.0
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list

$ sudo apt-get update

$ sudo apt-get install -y mongodb-org
```

This sequence of instructions "add[s] the MongoDB repository details so APT will know where to download the packages from…create[s] a list file for MongoDB… updates the packages list…[and] install[s] the MongoDB package itself." [2]

Upon installing mongodb, a few further steps are needed in order to create the database used in wingit as well as the admin user used in Django to interact with the database.

```
$ mongo

> use wings

> db.createUser({user: "wingitAdmin",pwd: "fuckitwingit",roles: [ { role:
"userAdmin", db: "wings" } ]})

> exit
```

Note: the database name, username and password here may be set to anything as long as the corresponding line in /wingit/wingit/settings.py is changed as well:

[2] Papiernik, M. (2015, June 15). How To Install MongoDB on Ubuntu 14.04. Retrieved May 26, 2016, from https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-14-04

```
162 |   connect('DATABASE',username='USERNAME',password='PASSWORD')
```

Microsoft Azure:
In order to install the Microsoft Azure SDK for Python, the following command needs to be run:

```
$ sudo pip install azure
```

AKA the “I ain't got time fo dis” option.

In further detail:

What is being installed when you run this is:

Client libraries that allow us to access Azure storage accounts

```
$ pip install azure-storage
```

Azure Service Bus runtime libraries that allow us to create queues and events.

```
$ pip install azure-servicebus
```

Azure’s Resource Manager to update the containers that holds our stored results.

```
$ pip install azure-mgmt
```

Azure’s service management API to add programmatic functionality to the Azure portal.

```
$ pip install azure-servicemanagement-legacy
```

The Neural Network has already been implemented and the existing net can be used by utilizing the api keys provided. The steps below describe how they’re implemented within Azure and can be skipped in terms of installation.

The predictive web service APIs for users and non users are used to predict preference values and are named:

`nonuser_dinner_net_predictive.py`
`user_dinner_net_predictive.py`

The training web service APIs for users and non users are used to train these neural networks for updated results and are named:

`nonuser_dinner_net_training.py`
`user_dinner_net_training.py`

All of the below files that are about to be mentioned can be found in the /keys folder in our package.

For the predictive web service APIs, the only value that has to be changed is the api_key value in the code. These values are stored in:

`dinner_net_nonuser_predictive_api.md`
`dinner_net_user_predictive_api.md`

For the nonuser and user predictive web services, respectively.

For the training web service APIs, the api_key value in the code has to be changed just like in the predictive web service. These values are stored in:

`dinner_net_nonuser_training_api.md`
`dinner_net_user_training_api.md`

For the nonuser and user training web services, respectively.

However, for the training web service, training results are uploaded to azure storage and require that that you fill in the storage_account_name, storage_account_key, and storage_container_name values that are in the code as well. These values can be found, respectively, in:

```
storage_account_name.md
storage_account_key.md
storage_container_name.md
```

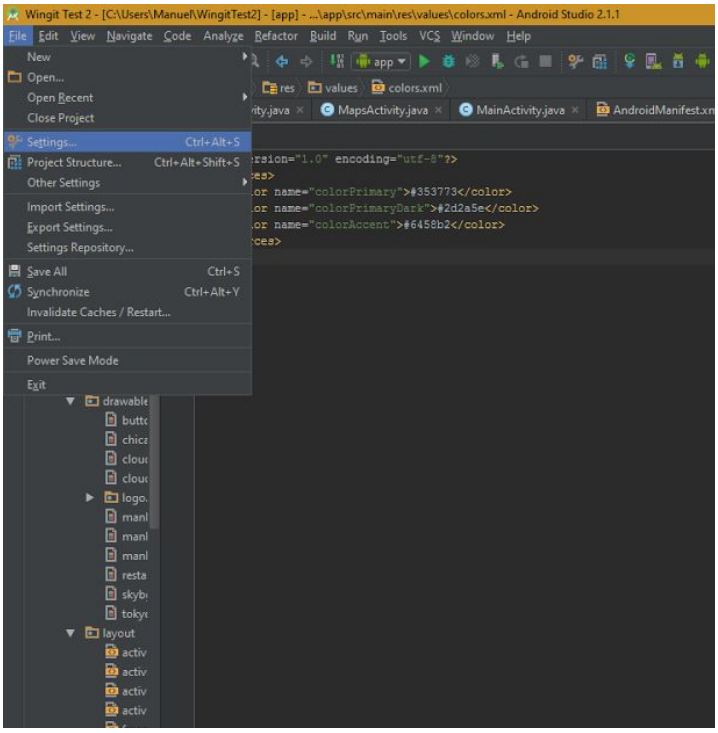
These values are used for both the user and nonuser training web services.

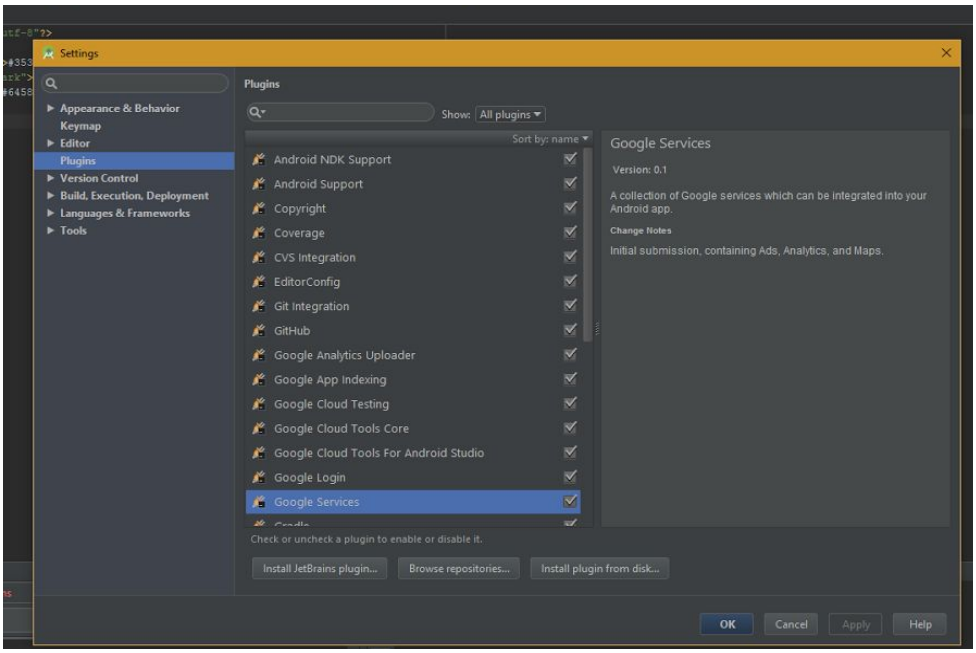
Android:

Application:
If you only wish to sideload the app yourself, you need to navigate to Android\\WingitApp\app\build\outputs\apk and sideload the .apk onto any Lollipop or Marshmallow device. Make sure your device allows the installation of third party applications, usually under Security in Settings.

Android Studio:
To compile and edit the application yourself, install the latest version of Android Studio, currently 2.1, as well as the relevant SDK and emulators if necessary. The application is designed to run on Lollipop and Marshmallow devices, or on an emulated image of SDK lvl 21-23.

In Android Studio, go to file - settings - plugins, and locate and install Google Services from the list if it is not pre-installed.





Open the folder WingitApp as a project, and you will be good to go.

Please note the API keys in the following locations in the project.
AndroidManifest.XML - google maps API key
MainActivity.Java - google places webservice API key

Features -
Where we wanted to go vs How far we went

From the onset of this undertaking, we set out to provide users with a fast and simple method of making their everyday decisions. In particular, starting with dinner then given enough time, moving on to bars and events using dinner as the base.

In that regard, we've certainly done way more than we initially thought feasible. Not only is the food feature fully fleshed out, it also provides the scaffolding to extend functionality to any other type of activity imaginable. This includes the machine learning functionality provided by the neural networks.

As of now, the website's main functionality consists of suggesting a restaurant/bar/cafe. It presents the user with an image from Google Places Image API, the name and address of the place, and links them to the place in Google Maps upon clicking.

Of course, the website also features fully working account registration, login, and logout.

Originally, we planned on gauging the user's preferences with an initial survey. However, after albeit a small number of trials, we found that this wasn't particularly helpful. Turns out people tend to like food in general. Who knew? So in the end, the process of gauging preferences was streamlined and integrated with the "winging it" process overall. After the user is shown a place, if they press the Wingit! button again, their preferences are updated accordingly as they've demonstrated their dislike of the option provided. Similarly, if they don't press the wingit! button again, it is assumed that they enjoyed the last place suggested.

In the future, this should be changed to closely mirror the application process of actually asking the user about their event to see if they actually went to the place suggested and how it turned out.

The upside of this change is the increase in ease in terms of extensibility, as surveys do not need to be created for each category of event as they're added to the service.

The android application is a good mock up of what we want to achieve but is incomplete.. This is mostly due to inexperience and hitting many dead ends when trying to figure out how to do certain things. At the moment, the

application has the login and signup forms set up, complete with error checking and messages, but there isn't a connection to our server. In the main screen, only the food tab works at the moment, and it isn't connected to our machine learning backend. The application could be finished with more work, something we plan to do this summer. A lot of the features were in progress but not completed. We are very happy with how the login and register pages reflect the design philosophy of the website.

What's left to do:

- Work on the accuracy of the location grabbing, It sometimes breaks.
- Connect to the remote servers for authentication and Azure purposes
- Better designed results card. Implement map view in card.
- Make cards swipeable rather than clicking wingit again.
- Get the other tabs working.
- Store past results in Mongo database and make mechanism to store feedback on past results.
- Notify the user the next day to give feedback on their search results and feed it back to Azure server
- Save last login.
- Some sort of user profile view with sign out button

In the end Wingit is everything we hoped it could be and more!