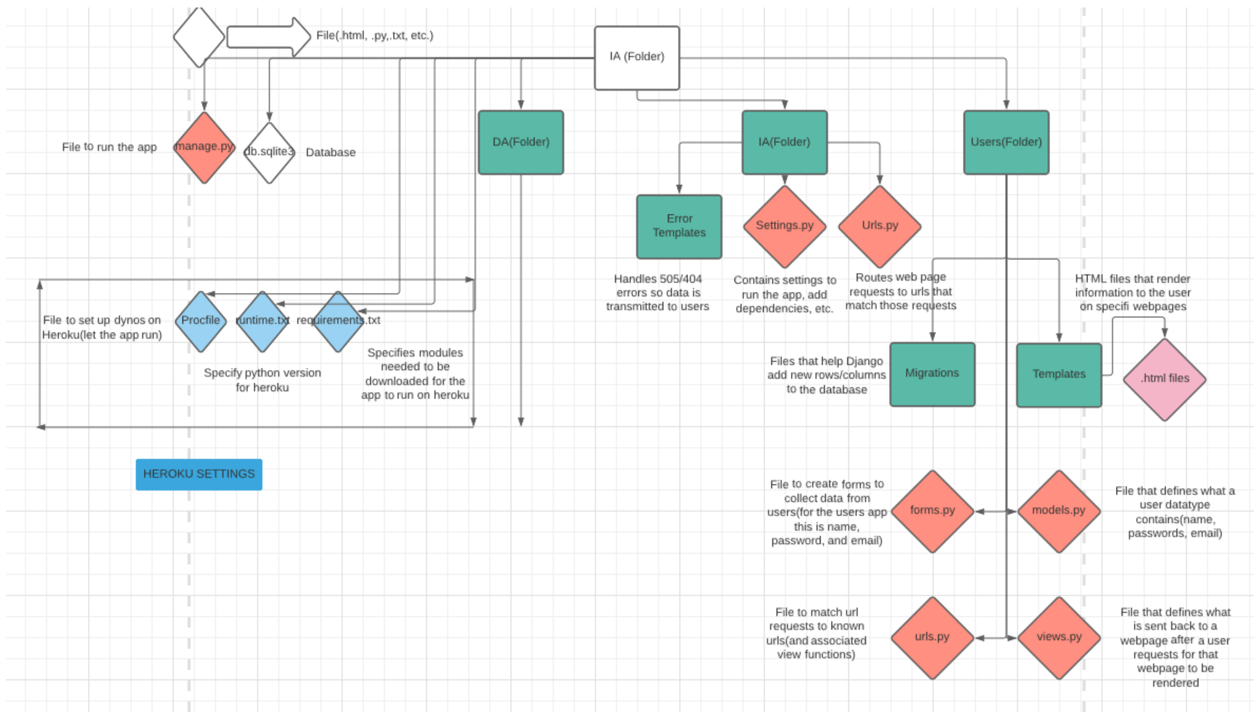
Criterion B

Design

A website has a frontend and backend; displayed below is a view of the file structure.



File(.html, .py,.txt, etc.)
IA (Folder)
File to run the app
manage.py
db.sqlite3
Database
DA(Folder)
IA(Folder)
Users(Folder)
Error Templates
Settings.py
Urls.py
Handles 505/404 errors so data is transmitted to users
Contains settings to run the app, add dependencies, etc.
Routes web page requests to urls that match those requests
HTML files that render information to the user on specifi webpages
File to set up dynos on Heroku(let the app run)
Procfile
runtime.txt
requirements.txt
Specify python version for heroku
Specifies modules needed to be downloaded for the app to run on heroku
Files that help Django add new rows/columns to the database
Migrations
Templates
.html files
HEROKU SETTINGS
File to create forms to collect data from users(for the users app this is name, password, and email)
forms.py
models.py
File that defines what a user datatype contains(name, passwords, email)
File to match url requests to known urls(and associated view functions)
urls.py
views.py
File that defines what is sent back to a webpage after a user requests for that webpage to be rendered

Figure 1. Overall File Structure.

The blue Heroku Settings are files to allow heroku to host this application.

The IA folder contains Django settings, a way to provide 504/505 errors to the user, and overall url listings.

The User folder contains the files necessary to add users to the database.

The db.sqlite3 database holds all entry and user data; the exact structure is handled by Django.

Each user has three data fields

- **name:string (public)**
- **Password:string (private)**
- **Email:string (private)**

Additionally, they have three actions

- **Register -** Create a new user

- **Login -** Login with a name and password
- **Logout -** Exit the current session / account

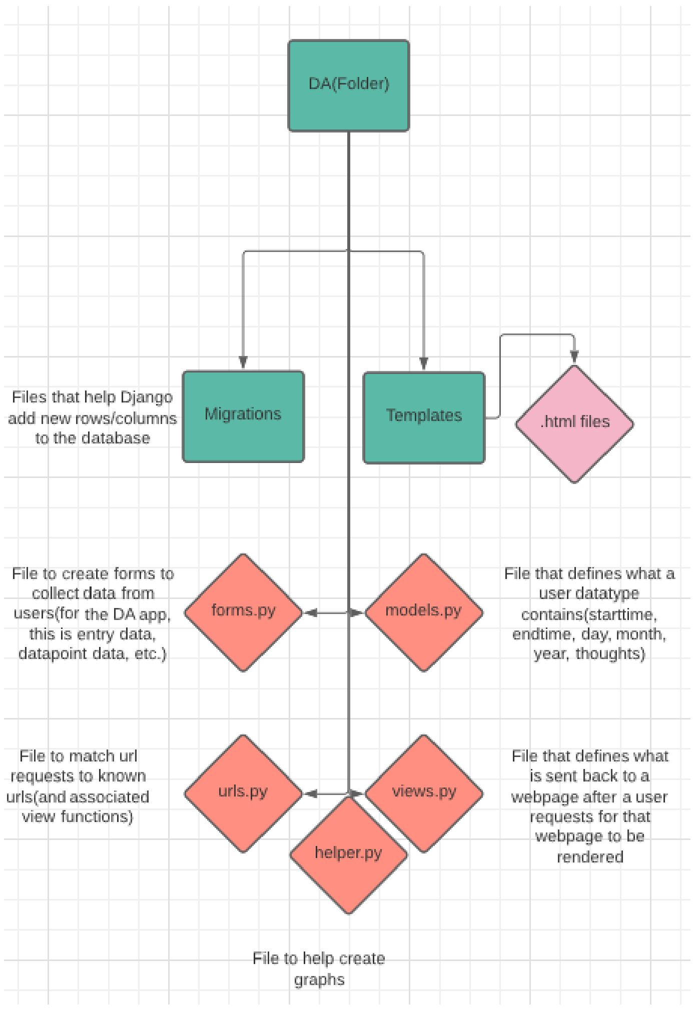

DA(Folder)
Files that help Django add new rows/columns to the database
Migrations
Templates
.html files
File to create forms to collect data from users(for the DA app, this is entry data, datapoint data, etc.)
forms.py
models.py
File that defines what a user datatype contains(starttime, endtime, day, month, year, thoughts)
File to match url requests to known urls(and associated view functions)
urls.py
views.py
helper.py
File that defines what is sent back to a webpage after a user requests for that webpage to be rendered
File to help create graphs

Figure 2. File structure in the DA folder.

The DA folder contains the methods to display the web pages and gather/store data.

Each session has a couple of public data fields

- **activityThoughts:string**
- **startHM:Django.timemodel**
- **endHM:Django.timemodel**
- **Month:string**

- **Year:string**
- **Day:string**
- **Dateadded:datetime object**

Additionally, each session has two actions

- **DeleteData -** delete the session
- **EditData -** edit any of the above fields for a session

Navigating from page to page depends on Django receiving a url request from a user, accessing the url in the urls.py file, and then using a views.py file in addition with an html file to display the page. Data is then entered through forms in forms.py, which is then sent as POST data to the database.

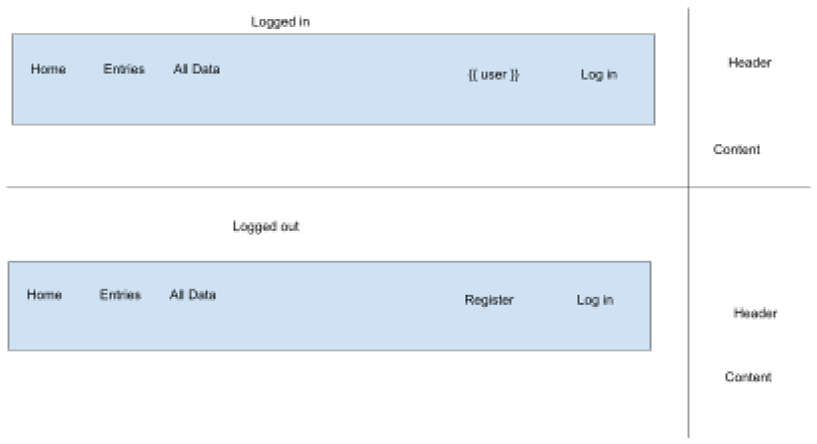


Figure 1. The base.html template

Figure 1 illustrates the base.html template. Each other webpage on the websites inherits from this; they all differentiate by adding in content in the content section. There is a necessity for a base.html because certain links such as logging out and accessing the home page should be available from all pages.

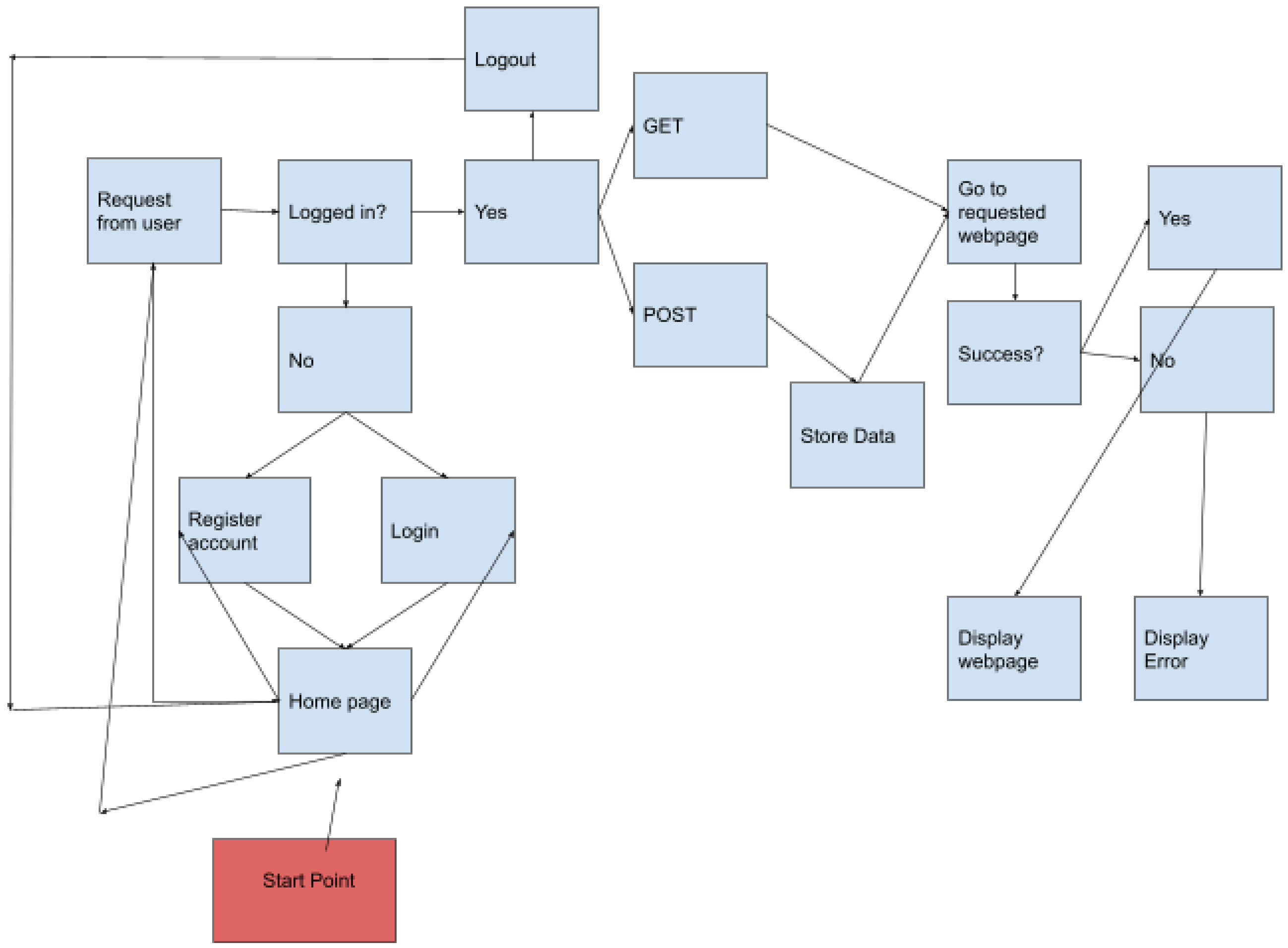


Figure 2. Overall logic of the website.

Figure 2 illustrates how the webpage deals with a request from the user. One loop deals with if the user is logged in or not: the outcome is that the user logs in. Afterwards, the request is separated into GET/POST and dealt with.

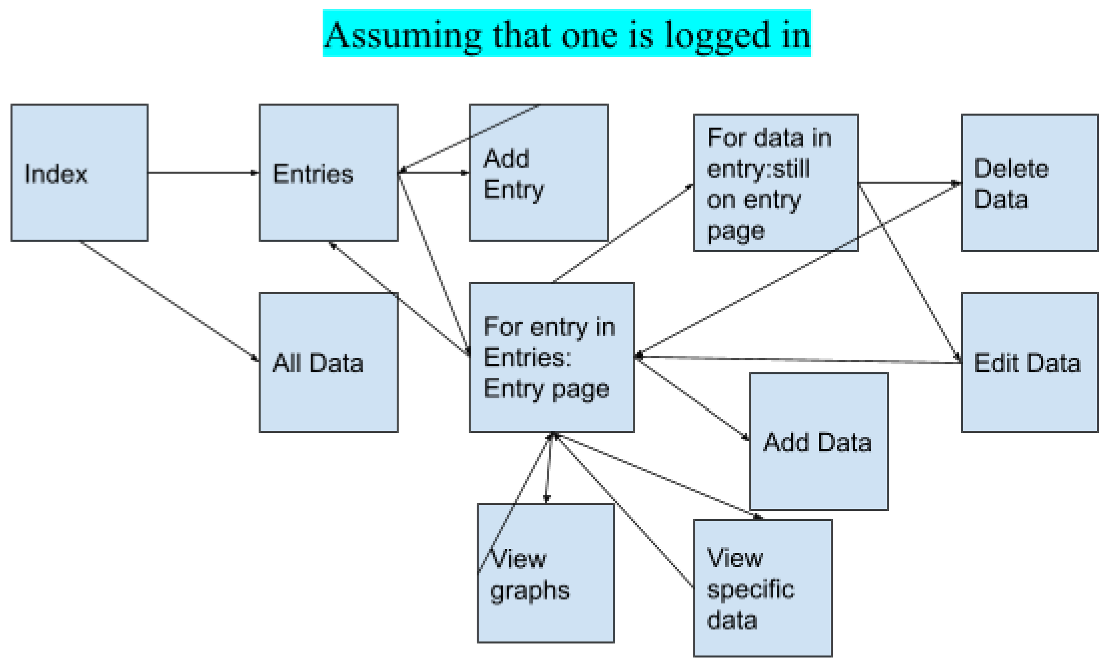

Assuming that one is logged in
Index
Entries
Add Entry
For data in entry:still on entry page
Delete Data
All Data
For entry in Entries: Entry page
Edit Data
Add Data
View graphs
View specific data

Figure 3. Overall site structure

The site functions through adding entries, data to those entries, and viewing that data. Navigating is made easier through the use of back buttons on each page.

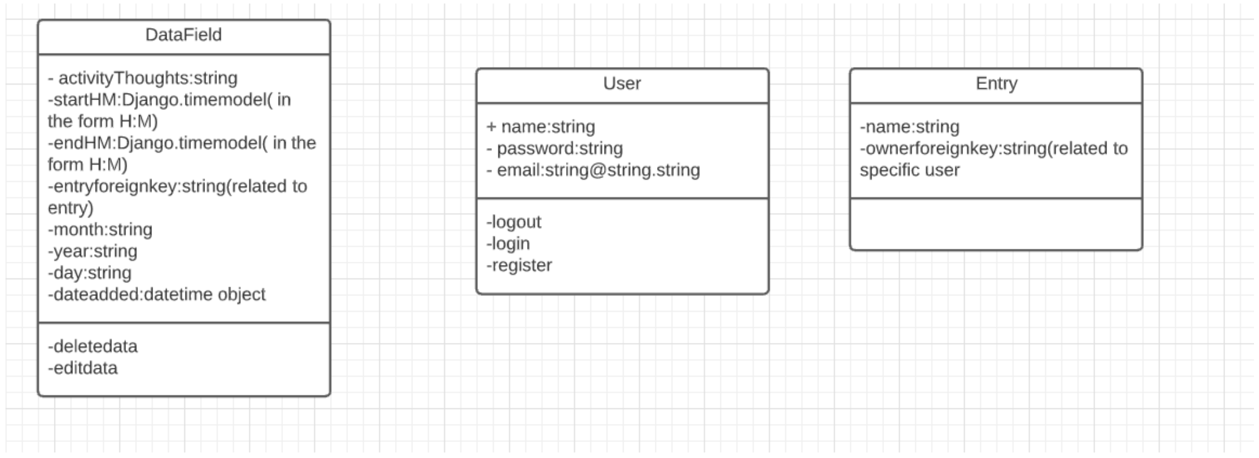

DataField
- activityThoughts:string
-startHM:Django.timemodel(in the form H:M)
-endHM:Django.timemodel(in the form H:M)
-entryforeignkey:string(related to entry)
-month:string
-year:string
-day:string
-dateadded:datetime object
-deletedata
-editdata
User
+ name:string
- password:string
- email:string@string.string
-logout
-login
-register
Entry
-name:string
-ownerforeignkey:string(related to specific user

Figure 4. The basic data structures for the project

At the highest level, there is a user with typical user attributes; each user can have many different entries, which have a text attribute. Then, each entry can have many data points with the aforementioned attributes.

Testing

Test Criteria	Method of testing	Adjustments-made
Users can be created and use the website	Manually creating users and exploring the functionalities of the website.	Users have an email attribute as well.
Entries can be added	Manually adding entries	Make entries have an “owner” attribute relating them to a specific user.
Individual data points can be added with a many-to-one relationship to an entry	Adding data through the website, and checking in the shell if each datapoint has the accurate “entry” attribute.	Added a foreign-key attribute
Only logged in users/admin can view their data.	Manually checking pages on different accounts seeing if error pages pop up.	Added a @login_required class decorator to all views.
Able to access website on Heroku despite web browser, pc, etc.	Manually typing in the url and exploring the functionalities of the website.	Made heroku accurately display website.
Data-querying form transmits data to backend, and displays requested data	Checking through shell sessions if transmitted data is saved to database. Additionally, manually checking on the website if data is displayed properly.	Made forms receive data and add it to the database so all fields are filled out.
Graphs display the relevant data	Manually checking the graphs + checking the data inputted into the graphs through print statements.	Made graphs work with plotly.
All links back and forth between different webpages work	Manually clicking on links as well as checking in the code that links work	Added links back/forward on each webpage + essential links on the base template.
Deleting/Editing entries works	Manually editing/deleting entries and checking that the website displays them. Also, entering a shell session and checking that the database has the edited data.	Added delete/edit functionality.
Make sure that only valid dates, information, days, months, years, etc can be added	Manually checking that an error pops up if incorrect information is entered + accurate information is submitted	Added form data checker models to ensure only relevant data is submitted.

Word Count: 156