MealMatcher Internals
Kevin Wang, Drew Wallace, Andreas Dias, Valerie Morin, Andrew Charette

Environment Requirements

The code for MealMatcher is written mainly in Python, HTML, CSS, and JavaScript. We used Django as a Python web framework and Bootstrap as the front-end framework. Everything we have on there should be supported on any operating system, though we almost strictly developed on Mac OSX (one of us used Windows).

Back-end Requirements
The Python version used for development was Python2.7 (specifically, 2.7.9 on the Heroku server). The Python libraries needed to run our system are the standard Python libraries with 2.7 plus the following:
dj-database-url==0.3.0
dj-static==0.0.6
Django==1.7.7
gunicorn==19.3.0
psycopg2==2.6
static==0.4
static3==0.5.1
django-post-office==1.1.1
pytz==2013.7
They can be installed using pip (for instance, pip install django django-post_office). To download pip and set up Django, see instructions at the following link: http://www.tangowithdjango.com/book17/chapters/requirements.html.
Note that site also contains very good tutorials for Django that we heavily used to get started. Also, our site uses a newer version of Django, 1.7.7, not 1.7 as instructed in the link above. Django Post Office is used to send email notifications.

For the database, we used Postgresql, which is supported on Heroku. It can be installed locally for the Mac from http://postgresapp.com/. Make sure to have psycopg2 installed to ensure compatibility with Postgresql if running locally (i.e. pip install -U psycopg2). Note that the database needs to be set up in a specific way following the settings in `src/mealmatcher_project/settings.py` in order to work locally (comments are included in the source code for further instruction). We also use PG Commander (https://eggerapps.at/pgcommander/) for admin control to disable users who've missed too many meals. The *index.html* page has detailed instructions on that procedure. Our discussed policy on disabling users is if they miss 2 scheduled meals within 2 weeks, which results in a 2-week suspension from the site. The reason we didn't use Django admin was we ran into issues described in *report.pdf*. The website and database are hosted on Heroku servers (which use Amazon Web Services). Heroku scheduler is used to send the scheduled emails asynchronously.

Front-end Requirements
Bootstrap can be downloaded here: http://getbootstrap.com/. We also used BootBundle (http://www.bootbundle.com/) and Bootswatch (https://bootswatch.com/) for web page templates

and css. We also made use of jQuery, CLNDR.js, Underscore.js, Moment.js, Bootbox.js, and jQuery.scrollTo.js. The JavaScript libraries can be downloaded through their respective Git repositories, which can all be found through a Google search.

Browser
We only used Google Chrome to view the website both locally and on Heroku, and also on mobile devices. We see no reason why it wouldn't work on the latest versions of other browsers.

Source Code and Running Locally

We used GitHub to collaborate, and our repository containing the source code can be found at https://github.com/mealmatcher/mealmatcher_project. If you download everything there and have the above requirements, you should be able to run the server locally with the command python manage.py runserver. Note that a local database should exist, and in both `src/mealmatcher_project/settings.py` and `src/mealmatcher_project/wsgi.py`, certain areas of code need to be commented/uncommented. Those files both have specific comments describing what needs to be done to run locally vs. for the version running on Heroku.

Model, View (Templates), Controller (Views)

Our website follows the Model-View-Controller architecture. Note that following Django syntax, the Templates correspond to the views and the Views correspond to the controller.

Models
Our models can be found under `src/mealmatcher_app/models.py`.
The UserProfile model is used in a one-to-one relationship with the Django built in User type to create the user accounts. Our website relies on the Princeton's Central Authentication System (CAS) for authentication only, so we don't have to worry about password encryption. Afterward CAS returns the valid netid, we find the corresponding UserProfile and User account in order to log the user in using Django's built-in user system. Note that only the login relies on CAS. Once you are logged out, you remain logged-in even if you log out of CAS, and need to use the log-out button on our site to log out of it. For more on CAS, see https://sp.princeton.edu/oit/sdp/CAS/Wiki%20Pages/Anatomy%20of%20CAS.aspx, as well as the tutorial set-up from our TA Tom Wu at http://fathomless-citadel-4589.herokuapp.com/.

The Meal model is used in a many-to-many relationship with the UserProfiles. A user can be in multiple meals and a meal can have multiple (2 users). It includes a datetime field, a location field, a meal_time field that specifies which meal of the day it is, and also attire1 and attire2 plus user1 and user2 fields, which tell you which attire goes with which username. The user1 and user2 text fields are needed even though we have the many-to-many relationship because the order of the UserProfiles associated with each meal is undeterminable from a Query alone.

Views
Our views (controllers) can be found under `src/mealmatcher_app/views.py`. The code contains enough comments to be easily followed. A brief description of each view is below:

- index: homepage
- site_login: contains CAS check and logs in user, making a new account if needed

- site_logout: log out of Django user, then redirect to log out of CAS as well
- find_meals: find-a-meal page backend, receives and processes the create-a-meal form with several checks to make sure it is valid. Also renders a list of open meals for the front-end calendar to display.
- view_meals: my-meals page backend, renders list of meals sorted by date into happening now, upcoming, and expired meals.
- edit_attire: Redirected to process the forms for editing an attire from the my-meals page. Renders view_meals after complete.
- delete_meal: Redirected to process the forms for deleting a meal from the my-meals page. Renders view_meals after complete.
- join_meal: Redirected to process the forms for joining open meals from the find-a-meal page, and renders view_meals after complete.
- match_meal: not a view. A helper function for join_meal
- error: General error page for when a form check is invalid, or something happens with our database queries. Should never be reached.
- disabled_user: view that is always redirected to when the user has been manually disabled for missing too many scheduled meals. Should otherwise never be reachable.

Please see `src/mealmatcher_app/urls.py` for which URLs are mapped to which view. See `src/mealmatcher_app/forms.py` for how the forms work.

Templates
Django templates are the viewed pages. CLNDR.js and Underscore.js generate the calendar Open Meals portion of the Find a Meal page, and Moment.js is used for operations on datetime objects in the Create a Meal form. Bootbox.js provides the popups on the site, and jQuery.scrollTo.js handles automated scrolling. Specific templates are all located under `src/templates/mealmatcher_app`

Code Organization

All source code for our project is found in the main `mealmatcher_project` directory. This includes manage.py which allows for starting the server, the shell, making migrations, etc.

`src/templates/mealmatcher_app` contains all the HTML files (templates) corresponding to each web page.
`src/static` contains sub-directories for the CSS files, fonts, images, and JavaScript libraries.
`src/mealmatcher_project` includes Python files for the Django site settings and the urls.py there links to the urls.py of mealmatcher_app.
`src/mealmatcher_app` includes the models, the views, the forms, and the URL mappings used (contained in previous urls file), as well as migrations made to the database from changing the models.

To add a new page to the website, add the HTML file to the templates, add a view (if necessary) and link view to the website URL with the mealmatcher_app urls.py. Add any required static files in the appropriate static directories.