Space Scheduler - Info sheet

A scheduling app by Gavin Ray

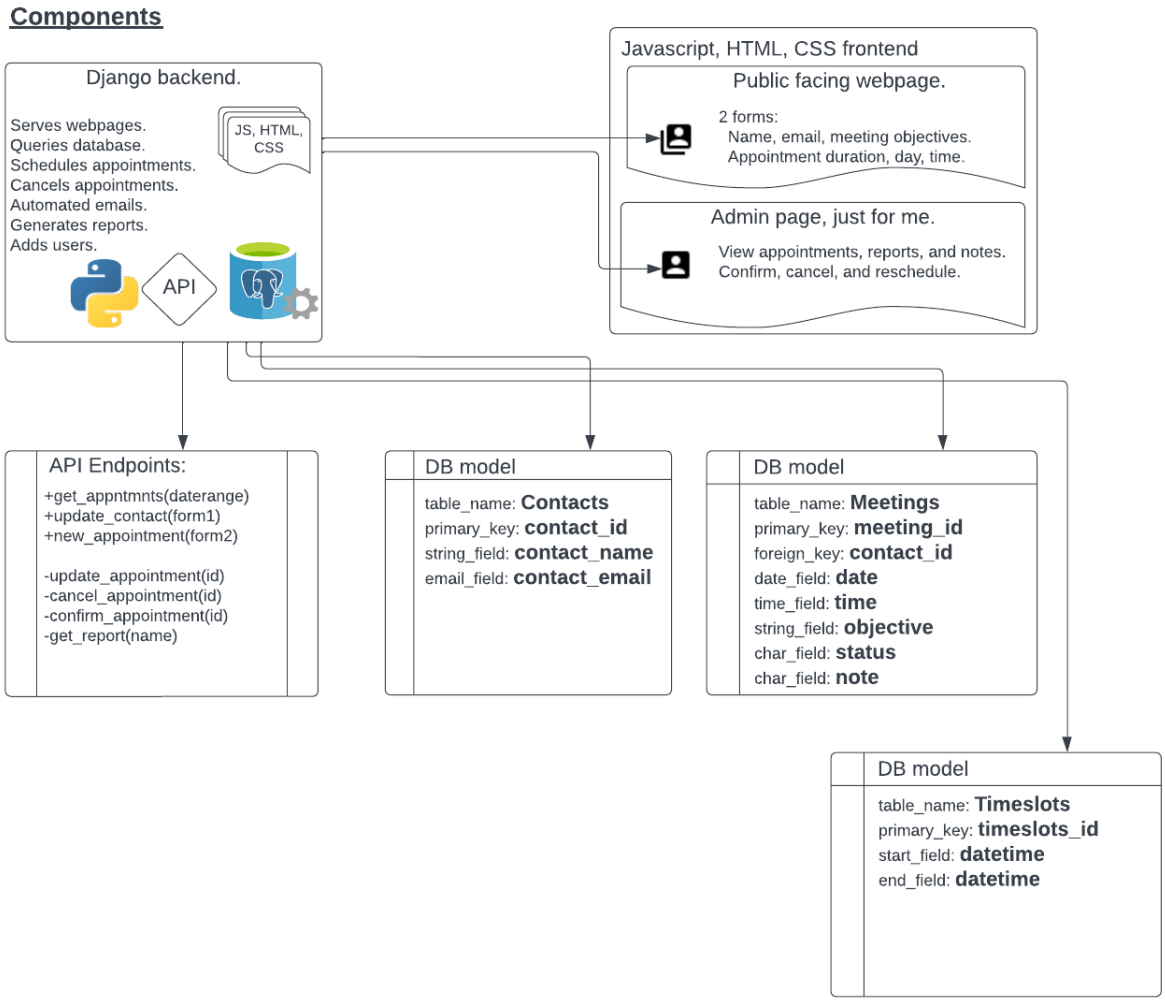
This project was created as a capstone project at the end of my Software Development program at WGU. It is a meeting scheduler that you can use to schedule a meeting with me. Once you have scheduled a meeting, the backend will automatically send you an email. There is also a secure dashboard that I can use to view meeting information, send automated emails, print reports, and add notes to meetings.

While this project uses Python (Django) for the backend, and JS/HTML/CSS for the frontend, I also have experience with other languages like Java, C++, and C#. Mostly what I hope you will learn about me is that I love to learn new tools.

This application is a work in progress, as all good projects are. Please continue to visit the live version of this site at https://gavin-ray-group.herokuapp.com/ often to see what I am adding. (Future additions are likely NOT to be included in the public repo of this project for security reasons.)

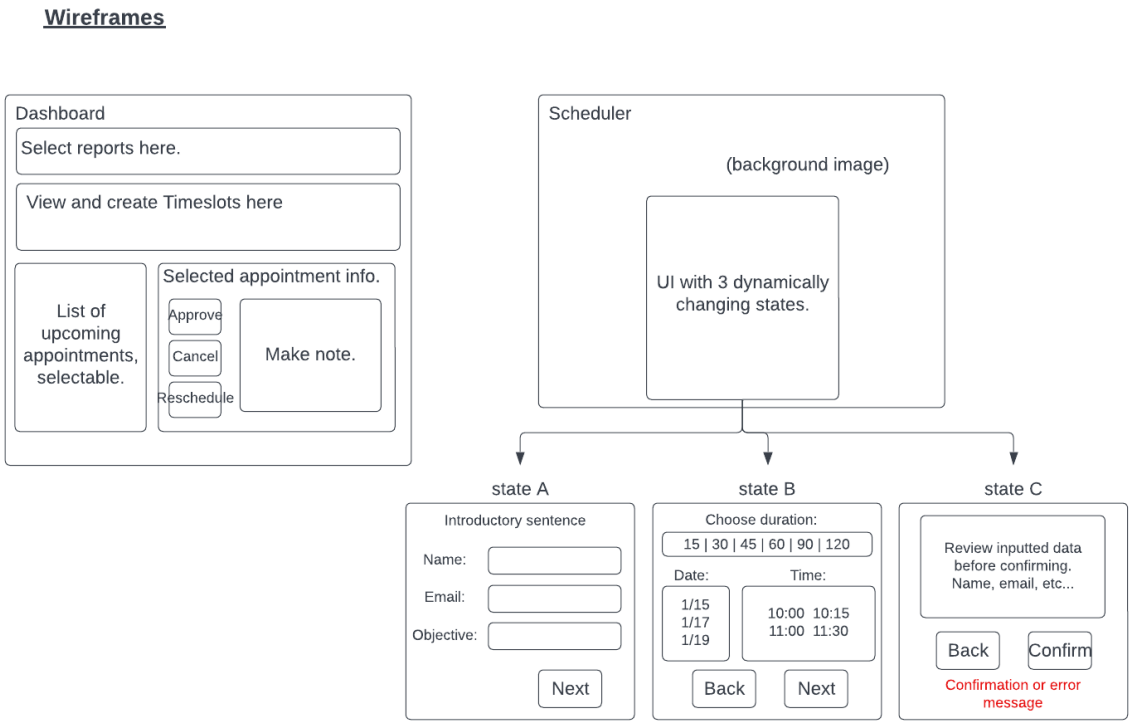
Table of Contents

Application Design:



The application will include a Django backend to facilitate information exchange between the JS, HTML, CSS frontend and the database. The frontend contains two pages. The first is a scheduler, shown above as “Public facing webpage” because this page will be publicly available for anyone to use. The second site is labeled here as “Admin page” because it will require a username and password. I also refer to the admin page as a dashboard, but the files are named admin.js and admin.css. Some of the API endpoints are shown in this image as well, which are located in the Views.py script. Three database models will be built to support the application. One will record user information, one for scheduled meetings, and the last for available times (timeslots).

UI Design



The Dashboard will be an internal site page, and therefore will be minimally stylized to reduce cost. It will have sections to get report data, update Timeslots, view and select upcoming meetings, and a section to view and update detailed meeting information.

The Scheduler has a background image that I created, and a UI area that can dynamically change between three states. The first state will allow for the inputting of name, email, and a meeting objective. The second state will allow for selecting a duration, date, and start time. The third and final state will display all the inputted information so that the user can verify its accuracy before submitting.

Test Plan

Purpose

Unit testing took place during the second phase of development. The tests assisted in building the program's backend components, and focused on transferring data into and out of the database. Tests were written for each function at the start of development of the function.

Needs

Pytest, Python's built in testing framework
Django.test.TestCase, Pytest wrapper for Django webapps.

Overview

The order in which I built this application was strategically planned out to allow for testing throughout the development process. I started by building the database models, and so the first tests were for creating and checking these entries. Next I was able to add tests for the database helper functions and api endpoints, starting with Contacts, then Timeslots, and finally Meetings. The reason for this order is that Meetings depend on Contacts and Timeslots.

Items

Pytest was used to create a four script testing suite. One test each will be found for each of the three database models and their helper functions, and one additional for testing some of the api endpoints. Each database model contains the attributes for defining the sql table fields, as well as helper functions for processing data as it is transferred into and out of the database. Most of the api endpoints are not tested, because they were not built during the second phase of development when testing took place. Validation of the api was easier to do once the front and back ends had been combined. Testing of the API was simple enough that it did not require much testing.

Deliverables

By utilizing Django's test framework (a wrapper of Pytest), I was able to create a temporary database at the beginning of each test. Dummy data needed for each test is loaded in first before the test is acted out. Most tests then use variations of the Assert command to find failures. If a failure is found, this is indicated in the terminal and the Assert command's message is printed.

Features

The following functionality is tested for by the test suite.

Insert new and existing contacts into db. Existing contacts are found by filtering existing contacts with the same email. If the email already appears in the database, the name of the contact will be updated if different. The Contacts.object is received by the test.

Insert contact with invalid email. When attempting to create a contact with invalid email, no changes to the database are made. A message is received by the test explaining that the contact was not created.

Insert new timeslot into db. This test ensures that new timeslots can be effectively entered into the database. If the new timeslot's times overlap with an existing timeslot, all overlapping timeslots are combined into one. Trying to insert a timeslot with invalid datetimes will error, and the test will not pass.

Insert new meeting into database. First, Meetings are checked to ensure that there is no overlap with existing meetings. Timeslots are also checked because a time slot encompassing the meeting times must exist for a meeting to be created. If the meeting is not created for these reasons, a message is returned, otherwise the Meetings.object will be received by the test.

Delete times lots and meetings. When a timeslot is deleted, the meetings during the time slot are not affected but no new meetings will be made for that time slot.

Query meetings and time slots. Functions are tested which return meetings and time slots during specific date time ranges, as well as meetings during a specific time slot. I'm also testing the ability to query meeting data by id.

Tasks

Running the tests can be done through the Django CLI.

##From the root directory:
>>>python manage.py test [test_file_name]

File names are as follows:

Scheduler/tests/test_contacts.py - - - - - - - - script for testing Contacts DB Model
Scheduler/tests/test_meetings.py - - - - - - - - script for testing Meetings DB Model
Scheduler/tests/test_timeslots.py - - - - - - - -script for testing Timeslots DB Model
Scheduler/tests/test_api.py - - - - - - - - - - - - script with limited testing

So to run a test an Contacts, you would run this command:
(Notice dot notation is used and extension is not provided.)

>>>*python manage.py test Scheduler.tests.test_contacts*

Pass Fail Criteria

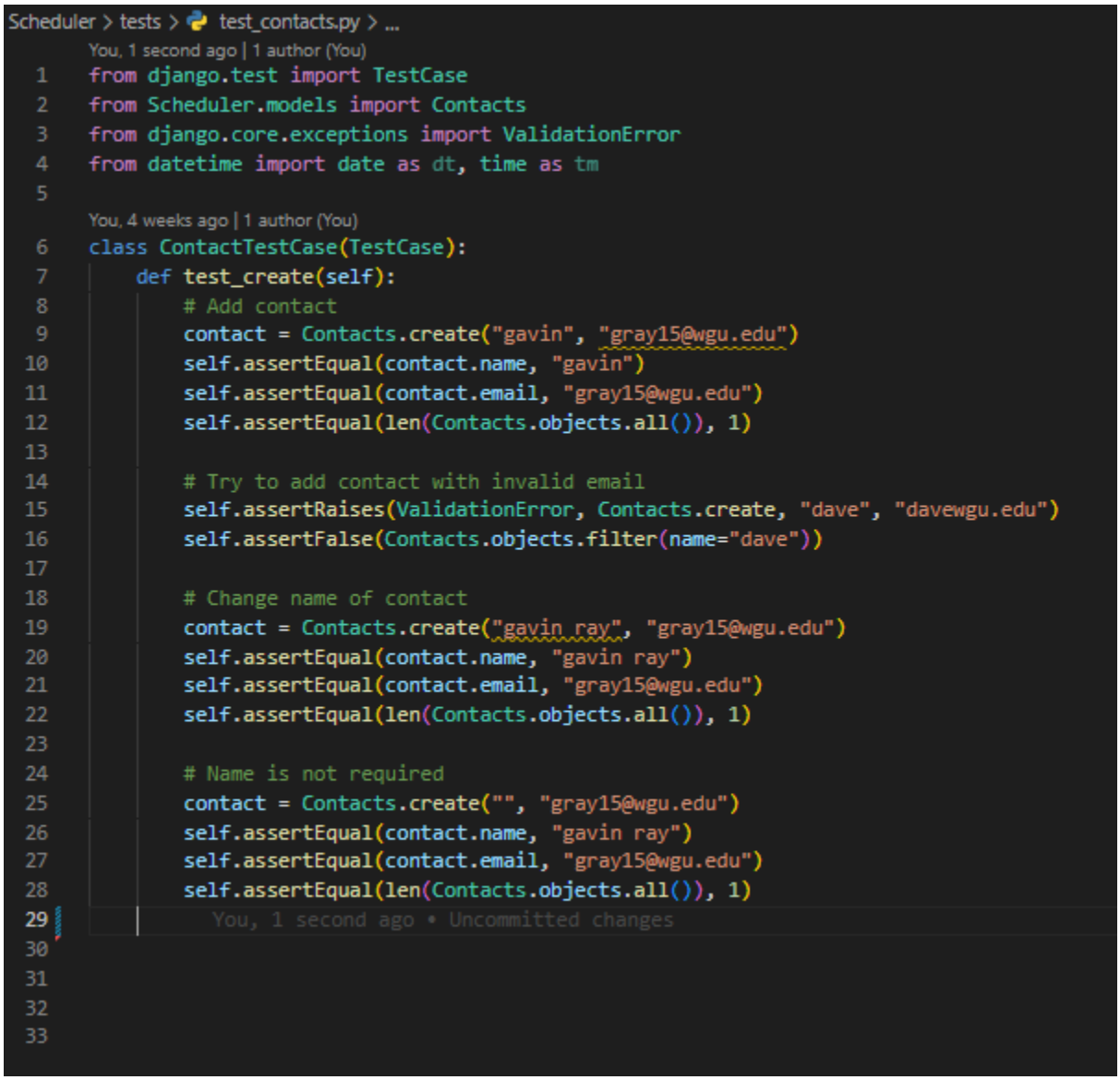
All test passes indicate no issues were found in the code. For tests that intentionally use invalid inputs (like invalid email addresses for new contacts), a pass indicates that the program is able to recover from failure.

Specification

The following is a screenshot of the Contacts test. You will see that the testing class inherits from TestCase, and contains one test: test_create(). This test looks at four actions.

- Creating a contact.
- Attempting to create contact with invalid email address
- Using create contact to change the contact's name

- Querying a contact by email, no name change.



```
Scheduler > tests > test_contacts.py > ...
      You, 1 second ago | 1 author (You)
1     from django.test import TestCase
2     from Scheduler.models import Contacts
3     from django.core.exceptions import ValidationError
4     from datetime import date as dt, time as tm
5
      You, 4 weeks ago | 1 author (You)
6     class ContactTestCase(TestCase):
7         def test_create(self):
8             # Add contact
9             contact = Contacts.create("gavin", "gray15@wgu.edu")
10            self.assertEqual(contact.name, "gavin")
11            self.assertEqual(contact.email, "gray15@wgu.edu")
12            self.assertEqual(len(Contacts.objects.all()), 1)
13
14            # Try to add contact with invalid email
15            self.assertRaises(ValidationError, Contacts.create, "dave", "davewgu.edu")
16            self.assertFalse(Contacts.objects.filter(name="dave"))
17
18            # Change name of contact
19            contact = Contacts.create("gavin ray", "gray15@wgu.edu")
20            self.assertEqual(contact.name, "gavin ray")
21            self.assertEqual(contact.email, "gray15@wgu.edu")
22            self.assertEqual(len(Contacts.objects.all()), 1)
23
24            # Name is not required
25            contact = Contacts.create("", "gray15@wgu.edu")
26            self.assertEqual(contact.name, "gavin ray")
27            self.assertEqual(contact.email, "gray15@wgu.edu")
28            self.assertEqual(len(Contacts.objects.all()), 1)
29                You, 1 second ago • Uncommitted changes
30
31
32
33
```

Procedures

Testing was approached through iteration. Building out the test suite during development really allowed me to change functionality when needed changes were discovered, and was greatly supported by the RAD methodology. About half way through development I realized that I would have issues with time zones because I was storing date and time separately. I had to refactor the database models to use datetime objects instead, and then update the tests accordingly. Testing also led me to find many bugs in how the helper functions were processing data for use by the api. When a test was insufficient for determining the cause of a bug, I just added another test.

Results

All tests successfully concluded.

```
setup-ext > commands.txt
You, 2 minutes ago | 1 author (You)
python manage.py makemigrations
python manage.py migrate
python manage.py test Scheduler.tests.test_contacts
python manage.py createsuperuser
heroku login
git push heroku main
heroku open
```

```
PROBLEMS 16   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS   SQL CONSOLE

(gavin-ray) C:\Users\GavinRay\Documents\GitHub\gavin-ray>python manage.py test Scheduler.tests.test_contacts
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.006s

OK
Destroying test database for alias 'default'...

(gavin-ray) C:\Users\GavinRay\Documents\GitHub\gavin-ray>python manage.py test Scheduler.tests.test_meetings
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.007s

OK
Destroying test database for alias 'default'...

(gavin-ray) C:\Users\GavinRay\Documents\GitHub\gavin-ray>python manage.py test Scheduler.tests.test_public_api
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
C:\Users\GavinRay\anaconda3\envs\gavin-ray\Lib\site-packages\django\db\models\fields\__init__.py:1365: RuntimeWarning: DateTimeField Timeslo
  warnings.warn("DateTimeField %s received a naive datetime (%s)"
.
----------------------------------------------------------------------
Ran 1 test in 0.064s

OK
Destroying test database for alias 'default'...

(gavin-ray) C:\Users\GavinRay\Documents\GitHub\gavin-ray>python manage.py test Scheduler.tests.test_timeslots
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..C:\Users\GavinRay\anaconda3\envs\gavin-ray\Lib\site-packages\django\db\models\fields\__init__.py:1365: RuntimeWarning: DateTimeField Times
  warnings.warn("DateTimeField %s received a naive datetime (%s)"
C:\Users\GavinRay\anaconda3\envs\gavin-ray\Lib\site-packages\django\db\models\fields\__init__.py:1365: RuntimeWarning: DateTimeField Timeslo
  warnings.warn("DateTimeField %s received a naive datetime (%s)"
C:\Users\GavinRay\anaconda3\envs\gavin-ray\Lib\site-packages\django\db\models\fields\__init__.py:1365: RuntimeWarning: DateTimeField Timeslo
  warnings.warn("DateTimeField %s received a naive datetime (%s)"
C:\Users\GavinRay\anaconda3\envs\gavin-ray\Lib\site-packages\django\db\models\fields\__init__.py:1365: RuntimeWarning: DateTimeField Timeslo
  warnings.warn("DateTimeField %s received a naive datetime (%s)"
...
----------------------------------------------------------------------
Ran 5 tests in 0.023s

OK
Destroying test database for alias 'default'...

(gavin-ray) C:\Users\GavinRay\Documents\GitHub\gavin-ray>
```

Source Code

Root Folders

- Mysite: Project directory with settings, urls, and some django specific configuration files.

- Scheduler **(My code is located in Scheduler folder.)**: Application directory with database migrations and models, static files (CSS and JS files), templates folder (html and email templates), tests, views (api logic), and some django configuration files.

Root Files

- Manage.py: auto generated django file for accessing command line tools
- Requirements: heroku friendly requirements list
- Requirements-conda: Anaconda friendly requirements list