Phase-4 Flask

1. Intro to Flask

2. REST APIs with Flask

3. REST APIs with Flask pt2

4. Client Server Communication

5. Authentication (Part 1)

6. Authentication (Part 2)

7. Deploying

SWBATs are our learning goals for lecture. After each lecture review the SWBAT. If there are any SWBATs you are struggling with, stay to ask questions at the end of lecture (time permitting) or follow up with your Cohort Instructor or Technical Coach outside of lecture.

1. Intro to Flask

SWBATs:

- ☐ Initialize a Flask application
- ☐ Understand how to use SQLAlchemy within Flask
- ☐ Use Flask routing and create views
- ☐ Use the Flask shell

What is the request-response cycle?	
What is a web API?	
What does each of the following status codes normally mean about their request? 200 400 500	
What HTTP Verbs aline with CRUD actions?	
How do we access the SQL alchemy toolkit necessary for initializing our models in flask?	
How does initializing a model in flask_sqlalchemy differ from initializing a model in SQLAlchemy in phase-3?	
How do you initialize a flask app?	
How do you configure flask to work with SQLAlchemy?	

Using the @app.route decorator build a route that returns an image to the route ‘/image’	
Using the @app.route decorator, build a dynamic root that will dynamically display a pet's name. The root ‘/pets/rose’ should display rose or any other name placed in that position in the path.	
What is application context?	
What is request context?	
Give an example of a request hook.	

2. REST APIs with Flask

SWBATs

- ☐ Discuss MVC architecture
- ☐ Explain the concept of REST and RESTful naming conventions
- ☐ Build and execute a GET and POST request
- ☐ Use Postman to interact with Flask
- ☐ Use serializers

Review: What is a web API?	
What is MVC?	
What is REST?	
List the RESTful routes for a resource called ‘animal’. Include the HTTP Verb, path, and a description of the route. `GET ‘/animal’ reads all`	
How do writing restful routes differ from the non-restful routes written yesterday?	
Will it still be necessary to know how to write non-restful routes?	
Write a get all restful route for ‘/cats’. Include the class and api.add_resource.	
Write a get one restful route for ‘/cats/<int:id>’. Include the class and api.add_resource.	
What is serialization?	
How is serialization implemented in the models?	
How do we gain access to the json body of a request?	
Write a create restful route for ‘/cats’.	

Review: What is a web API?	
What is MVC?	
What is REST?	
Include the class and api.add_resource.	

3. REST APIs with Flask pt2

SWBATs

- ☐ Build and execute a PATCH and DELETE request
- ☐ Discuss the importance of handling exceptions
- ☐ Handle exceptions
- ☐ Use Flask validations

Review: Is REST?	
How is werkzeug to handle exceptions such as NotFound	
How can you end a request and send a 404 error to the client if a resource is not found?	
Write a Patch request to ‘/cats/<int:id>’	
Write a Delete request to ‘/cats/<int:id>’	

4. Client Server Communication

SWBATs

- ☐ Connect a React app to a Flask API
- ☐ Execute requests from React
- ☐ Handle errors in React
- ☐ Discuss CORS

Write a GET all fetch request in Javascript	
Write a Create fetch request in Javascript	
Write an Update fetch request in Javascript	
Write a Delete fetch request in Javascript	
Why is validation important?	
What are two ways to validate columns in the models?	
How can we use yup to validate our form data before a request is sent?	

5. Authentication (Part 1)

SWBATs

- ☐ Discuss the importance of authentication in web apps
- ☐ Explain the difference between authentication and authorization
- ☐ Discuss the relationship between cookies and sessions
- ☐ Use token-based authentication using cookies and sessions

Why is Authorization important?	
What's the difference between Authentication and Authorization?	
What are cookies?	
What are sessions?	
How can we keep a user logged in using sessions?	
Write a login route	
Write a logout route	

6. Authentication (Part 2)

SWBATs

- ☐ Use authorization
- ☐ Handle authorization errors on the front end

What is Authentication	
What is the issue with storying plan text passwords?	
How do we protect our users' passwords?	
What is Bcrypt?	

How do we set password hashing on the user model?	
How do we set up authentication on the user model?	
Write a login route that authenticates our user.	

7. Deploying

Checklist
Deploying a Flask App with Render

- [ ] Explain what it means to deploy an application
- [ ] Observe how to configure an application for deployment
- [ ] Observe how to deploy an application to render

Steps for deploying

- [ ] Sign up for Render https://dashboard.render.com/register
- [ ] Connect your GitHub account.
- [ ] Install PostgreSQL (You can skip this step if postgres is installed
- [ ] **PostgreSQL Window**
    - [ ] `sudo apt update`
    - [ ] `sudo apt install postgresql postgresql-contrib libpq-dev`
    - [ ] `psql --version`
    - [ ] `sudo service postgresql start`
    - [ ] Create a database user, check what your operating system user name is
        - [ ] `whoami`
    - [ ] `sudo -u postgres -i`
    - [ ] `createuser -sr your_user_name_here`
- [ ] **PostgreSQL Mac**
    - [ ] `brew install postgresql`
    - [ ] `brew services start postgresql`

- [ ] Create a Database on Render (skip this portion if you already have a Postgres instance on Render)

- [ ] On the dashboard click `New +` and select `PostgresSQL`
- [ ] Name the database something generic like `postgres_instance` or `my_database`
- [ ] Set the PostgreSQL version to 15
- [ ] Scroll to the bottom and click `Create Database`

- [ ] Create a flask app
  - [ ] Useful libraries: python-dotenv gunicorn psycopg2-binary Flask-SQLAlchemy Flask-Migrate SQLAlchemy-Serializer Flask-RESTful
  - [ ] In app add `import os` and `from dotenv import load_dotenv` set app.config['SQLALCHEMY_DATABASE_URI'] os.environ.get('DATABASE_URI')
  - [ ] Create a .evn file
    - Note: https://github.com/motdotla/dotenv Dot env will allow you to set environment variables.
- [ ] Create a file requirements file that will be used by Render with `pipenv requirements > requirements.txt`
- [ ] At this point, create at least one model and one route to test your Flask Web API.

- [ ] Start to build out the React Client with at least one component that a request to the test route you built for your backend.

  Note: Your React app and server should be in the same directory. The root directory structure should look like this.
  |- client
  |- server
  |- .env
  |- .gitignore
  |- Pipfile
  |- Pipfile.lock
  |_ requirements.tx

- [ ] Configure your client by creating a static react app.
- [ ] In your client's package.json add a proxy server `"proxy": "http://localhost:5555",`
  - [ ] Remove `http://localhost:5555/` from any of your fetch requests but keep the endpoint. For example: A GET all to productions should only be passed the end point '/producitons' as the url.

- [ ] In the root directory of your app run ` npm install --prefix client`
- [ ] Run `npm run build --prefix client`
  - Note: You'll notice there's a new build folder in client. This is your static app

- [ ] Configure your Client routes. Our web API will interfere with our routes from react-router, so we must handle them on our server.
  - [ ] In app.py add:

```
app = Flask(
    __name__,
    static_url_path='',
    static_folder='../client/build',
    template_folder='../client/build'
)
@app.route('/')
@app.route('/<int:id>')
def index(id=0):
    return render_template("index.html")
```

- [ ] Create a local environment variable that connects to your render db.
    - [ ] On your Dabases page on Render, click the 'Connect' select the External Connections tab, and copy the "External Database URL"
    - [ ] In the .env file, create an environment variable 'DATABSAE_URI' and set it to the external database URL. IMPORTANT: replace postgres with postgresql
- [ ] Migrate your database
    - [ ] Migrate your database
    - [ ] flask db init
    - [ ] flask db revision --autogenerate -m 'Create tables'
    - [ ] flask db upgrade
    - [ ] Test your app locally to make sure it works by running `gunicorn --chdir server app:app`
- [ ] Push to GitHub
- [ ] Create a repo on git hub
- [ ] Connect your local project to get hub.
    - [ ] `git remote add origin git@github.com:<your-github-name>/your-repo-name.git`
    - [ ] `git add .`
    - [ ] `git commit -m 'my first commit'`
    - [ ] `git push origin main`
- [ ] Create the Web Service on Render
    - [ ] On the Render dashboard click `New +` and select Web Service.
    - [ ] Connect your repo by selecting it from 'connect a repository' or passing render the repository's URL
    - [ ] Name your application
    - [ ] Change the "Build Command" to
      `pip install -r requirements.txt && npm install --prefix client && npm run build --prefix client`
    - [ ] Change the "Start Command" to
      `gunicorn --chdir server app:app`
    - [ ] Scroll down and click Advance
    - [ ] Add Environment Variable `PYTHON_VERSION` as 3.8.13

- [ ] Add Environment Variable `DATABASE_URI``
    - [ ] In a new tab open your Postgres instance on Render. Click “Connect” and copy the Internal Database Url
    - [ ] Past that url as the value for `DATABASE_URL` IMPORTANT: In the url replace postgres with postgresql
- [ ] Create your webserver and get a snack!!!

How do you update your deployment?	
Where’s the first place you should check if your deployment is failing?	