Project Proposal

By: Harshith Moningi, Ishaan Bhandari, Murray Ahmed, Isaac Exposito

Pitch

This application will allow students wanting to sublease their apartments a chance to advertise their property to other students on campus. This would be useful for students who are paying rent and planning on leaving the area for a set amount of time. It would also be useful for students looking for an affordable place to stay during the summer.

Functionality

- Users will be able to post listings for subleases
- Users will be able to remove posts for their subleases
- Users will be able to expand and view each post
- Contact information, i.e Users can click on email of post creator to pull up their email
- Users can filter or search through posts

Components

Backend: Flask, Python & Database: MongoDB

Most of our team is familiar with Python so we chose to use a Python backend framework. We'd use Flask for our backend to process frontend server requests and would store all of our data(different posts) in our MongoDB database.

We expect to use PyMongo to help interact with the MongoDB database. We will explore using Flask-RESTful to help with API calls. We will use Heroku to deploy the website in the end.

For backend testing we plan to utilize the `pytest` unit testing framework along with integration testing libraries like `Flask-Testing` for API testing and `MongoMock` for simulating MongoDB interactions in tests.

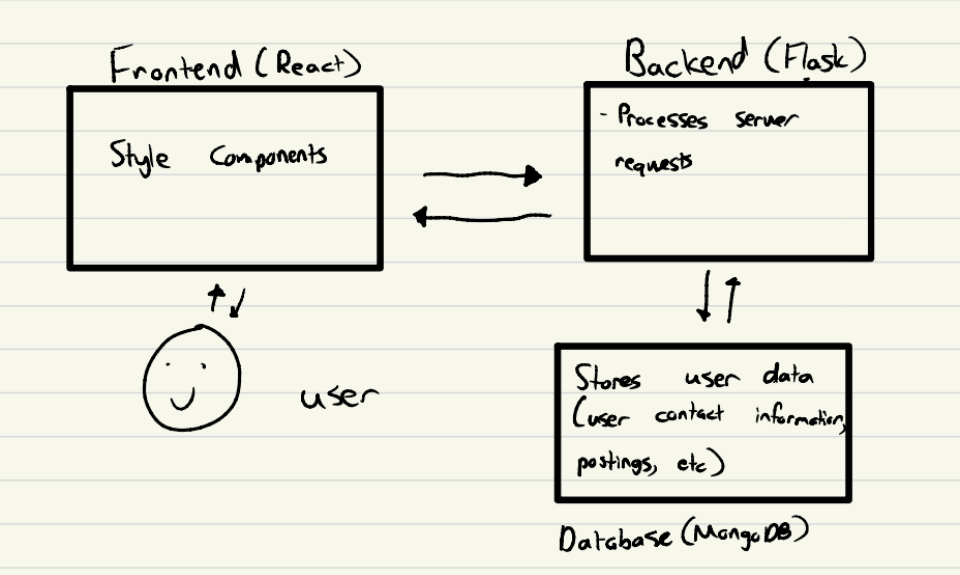
Frontend: React.js, CSS, HTML

We're planning on using React for our frontend because of its component style nature, which simplifies the process of creating new listing posts. We'd also use HTML and CSS for styling.

For JavaScript code in our React.js frontend, we use the `istanbul` package, integrated with `Jest`, to compute code coverage. Jest will also be used to for the general unit testing of the frontend.

We will perform unit testing to evaluate individual components of our frontend and backend in isolation. This will help us verify that each piece of code functions correctly on its own. We will also implement integration testing to make sure that all of our components are interacting properly with each other. For example, we would test to see how our React components

communicate with the Python backend and how the backend interacts with the MongoDB database.



Schedule (for each week [bullet point])

1. Install required packages and technologies (if we run into errors, should take 3-5 days); Create the repository and get everyone to clone it (should take 1 – 3 days depending on errors)
2. Design an example layout of how we want the website to look like (should take most of our time); implement the buttons and other style features without functionality (should take 1-2 days)
3. Add functionality to create post objects with the details we want for the frontend; Post request should be processed by backend populate the MongoDB database on button click
4. Make the post appear on screen appropriately after creation; create the ability to expand post to see details
5. Add functionality to edit the posts; Also add button to delete posts from the frontend
6. Make sure to remove posts from database after deletion; Start writing some unit tests for our application's core functionality
7. Connect Email to each post so the user can contact the creator of the post; Also add implementation for opening up location in a new Google Maps tab.
8. Finish unit testing to make sure the basic parts of the code are working; start integration testing
9. Add authentication of users; add functionality for searching/filtering posts
10. Finish functionality for filtering posts; Finish integration testing; deploy the website

Potential Risks

1. We may encounter issues calling backend methods from the frontend, more specifically with connecting our database (MongoDB) to our code, since this is something we've

never done before. We want to be able to create, update, and delete values in MongoDB, and so if we have issues, we plan to confer with our mentor, who has experience with this and shouldn't push us back more than a few days.

2. The learning curve for some of our technologies may be tougher than expected. Flask, react, and MongoDB may be tougher than expected. We should be prepared to explore alternatives (e.g. Node.js for backend, SQLite for database) should they take longer than expected. This should not delay us more than a few days, since this should be apparent from early on. It is also unlikely that we will have to use different technologies since the ones we chose have a reputation for being beginner friendly.
3. We may have issues syncing frontend with React to our Flask backend and MongoDB database. We may run into issues where what's displaying on the screen for the user is different from what we have in the database. It would likely be because whenever we are adjusting something in the frontend, we are not sending out a server request to the backend. Or, we are creating a request to the backend and not adjusting it in the frontend. This would probably take a few days to debug.
4. Unit Testing and Integration Testing are new concepts to all of us, and so getting used to writing good unit tests and integration tests will be key to our success. We will probably have to spend several days going over some basic tutorials on the subject, and then try our best to write them to have maximum coverage. Our mentor has experience using unit tests, so we will look to him for help if we get stuck.

Teamwork

Since we are a team of four people, we will divide into two sub teams of two people each: a front end (Harshith and Murray) and a backend team (Ishaan and Isaac). The concept of an apartment listing website in and of itself is not unique, so it is up to the front-end team to get creative while the backend can use their technical capabilities. This is a fair distribution of labor given each of our backgrounds in computer science, specifically website development. Given how we will typically not be in proximity of each other, and we have experience with Git and version control we will utilize Github to reduce team friction. In addition, we will set up a discord server to communicate more directly.

Continuous Integration

For testing and computing test coverage in our development stack, we use the following specific libraries and tools:

1. Testing Library:
   - Backend (Flask with Python): We utilize the `pytest` testing framework along with libraries like `Flask-Testing` for API testing and `MongoMock` for simulating MongoDB interactions in tests.
   - Frontend (React.js, HTML, CSS): We rely on `Jest` and `React Testing Library` for unit and integration testing of React components. For end-to-end testing, we use `Cypress`.

2. Test Coverage Computation:
   - To measure test coverage, we use `coverage.py` for Python code in the Flask backend. It generates detailed reports showing which parts of the code are covered by our tests.

- For JavaScript code in our React.js frontend, we use the `istanbul` package, integrated with `Jest`, to compute code coverage. It provides insights into how much of our codebase is exercised by tests.

Here's an example of our pull request workflow:

1. When working on a new feature or bug fix, a developer creates a new branch from the `main` branch, naming it descriptively to reflect the task's purpose.
2. Coding and Testing
3. Commit and Push
4. Pull Request Creation
5. Review and Feedback
6. Addressing Feedback
7. Approval and Merging

This workflow ensures that code changes are thoroughly reviewed, tested, and integrated into the codebase while maintaining a high level of quality and consistency.