CSE183 Fall 2021 - Assignment 8

In this assignment you create, as a team, Mugtome Bazaar, a Facebook Marketplace style goods trading system, as a Single Page Full Stack Web App using the NERP Stack: Node.js, Express, React, and PostgreSQL, plus Material-UI.

This assignment is worth 20% of your final grade.

Late submissions will not be graded.

Installation

See instructions in Assignment 2 and ensure you are running the current LTS version of Node.js. You will also need to have downloaded and installed Docker Desktop: https://www.docker.com/products/docker-desktop.

Setup

Download the starter code archive from Canvas and expand into an empty folder. We recommend, if you have not already done so, creating a folder for the class and individual folders beneath that for each assignment.

The starter code archive contains some files you will modify:

```
backend/api/openapi.yaml
backend/src/app.js
backend/sql/data.sql
backend/sql/schema.sql
backend/sql/indexes.sql

frontend/src/components/App.js
frontend/src/components/__tests__/App.test.js
```

Some you should ***not*** modify:

```
package.json

backend/.env
backend/docker-compose.yml
backend/package.json
backend/src/server.js
backend/src/dummy.js
backend/sql/database.sql
backend/src/__test__/db.js
backend/src/__test__/dummy.test.js

frontend/package.json
frontend/public/index.html
frontend/public/favicon.ico
frontend/src/index.js
```

And some you may want to remove after basing your tests on them:

```
e2e/tests/Dummy.test.js
frontend/src/components/Dummy.js
frontend/src/components/__tests__/Dummy.test.js
```


To setup the development environment, ***navigate to the folder where you extracted the starter code*** and run the following command:

```
$ npm install
```

This will take some time to execute as `npm` downloads and installs all the packages required to build the assignment.

To start the database Docker container, in the `backend` folder run:

```
$ cd backend
$ docker-compose up -d
$ cd ..
```

The first time this runs it will take a while to download and install the backend Docker PostgreSQL image and start the database service for your server to run against.

To start the frontend and backend dev servers, run the following command:

```
$ npm start
```

The frontend and backend servers can be run individually from their respective folders by running `npm start` in separate console / terminal windows.

To run the linter against your API or UI code, run the following command in the `backend` or `frontend` folder:

```
$ npm run lint
```

To run API tests run the following command in the `backend` folder:

```
$ npm test
```

To run UI tests run the following command in the `frontend` folder:

```
$ npm test
```

To run end-to-end tests run the following command in the `e2e` folder:

```
$ npm test
```

To stop the database, run:

```
$ cd backend
$ docker-compose down
$ cd ..
```


Web App Specification

This system must be a Single Page Web Application using the following technologies.

Browser:	React & Material-UI
Server:	Node.js & Express
Storage Tier:	PostgreSQL

The server must present an OpenAPI constrained API to the SPA running in the browser and the server must store its information in a PostgreSQL database.

Do NOT have the user interface simply download the contents of the database at startup and run all subsequent operations from an in-browser-memory cache. Submissions taking this approach will be *severely* marked down.

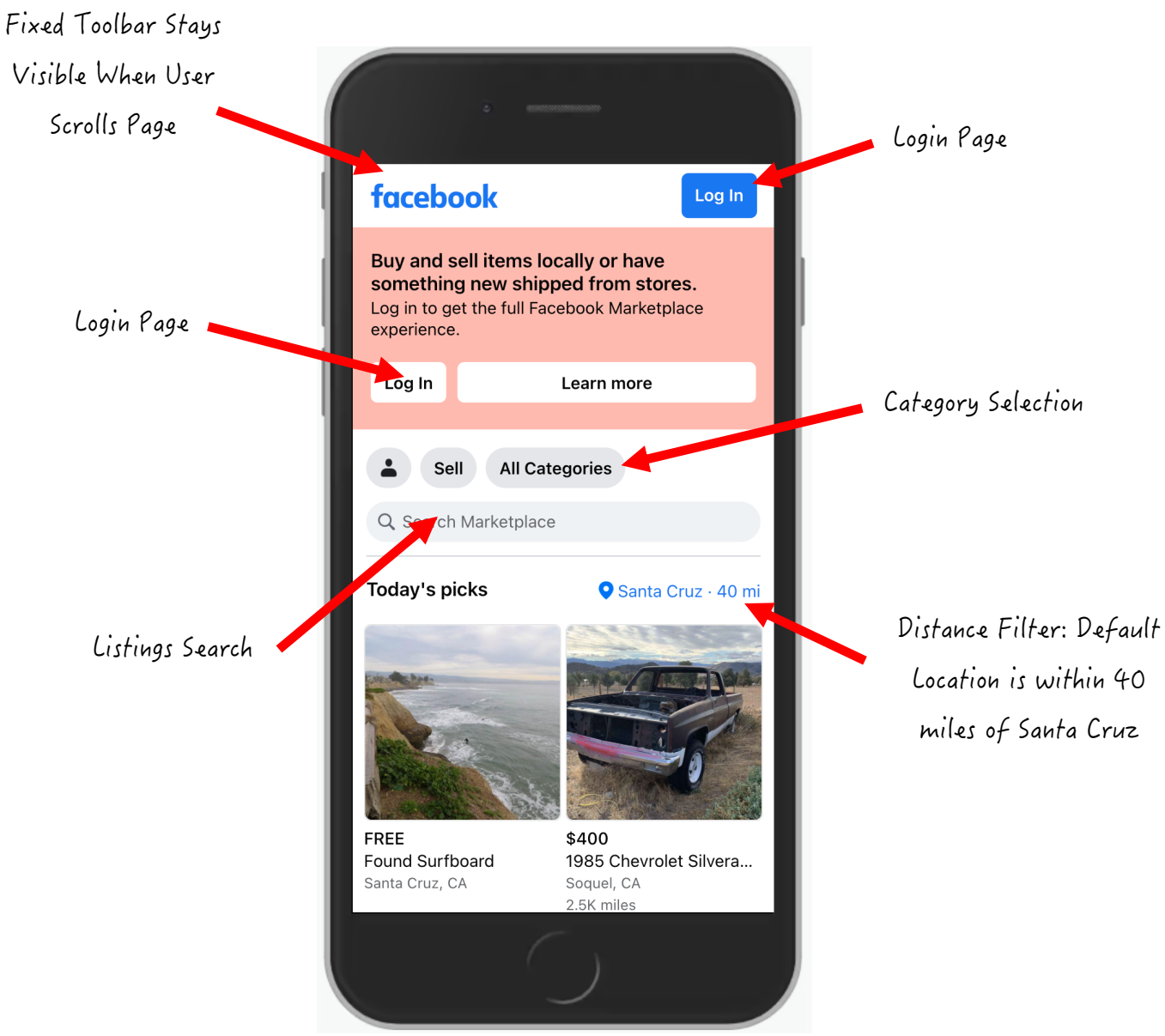
A “mobile first” approach should be taken; the principal operations being:

- Users:
  - Login Screen
  - Create New Account
- Categories
  - Category Selection
  - Subcategory Selection
- Filters
  - Category & Subcategory Specific filters
  - Location
- Navigation
  - Breadcrumbs
- Search
  - Across all Categories
- Listings
  - Infinite Scroll
  - Creation (only if logged in)

For the desktop version, the principal operations are as for mobile, but category selection and filters should be in a permanent drawer rather than full screen. See the real Facebook Marketplace for details.

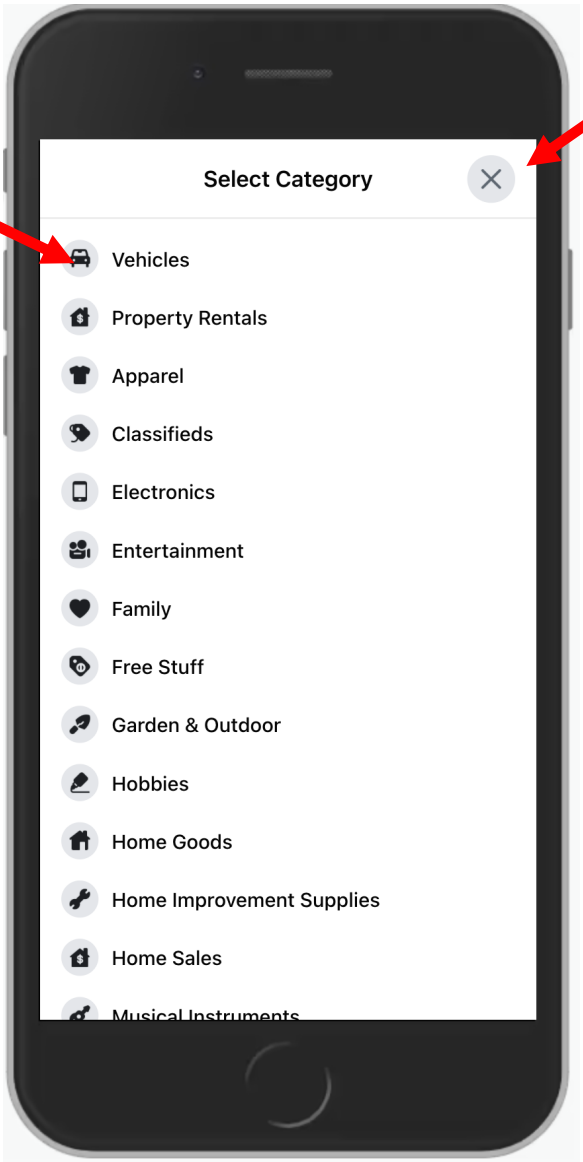
Most of the above are examined in detail on the following pages but if you’re unsure how something should work, use the genuine Facebook Marketplace to see how it does it.

Home Page



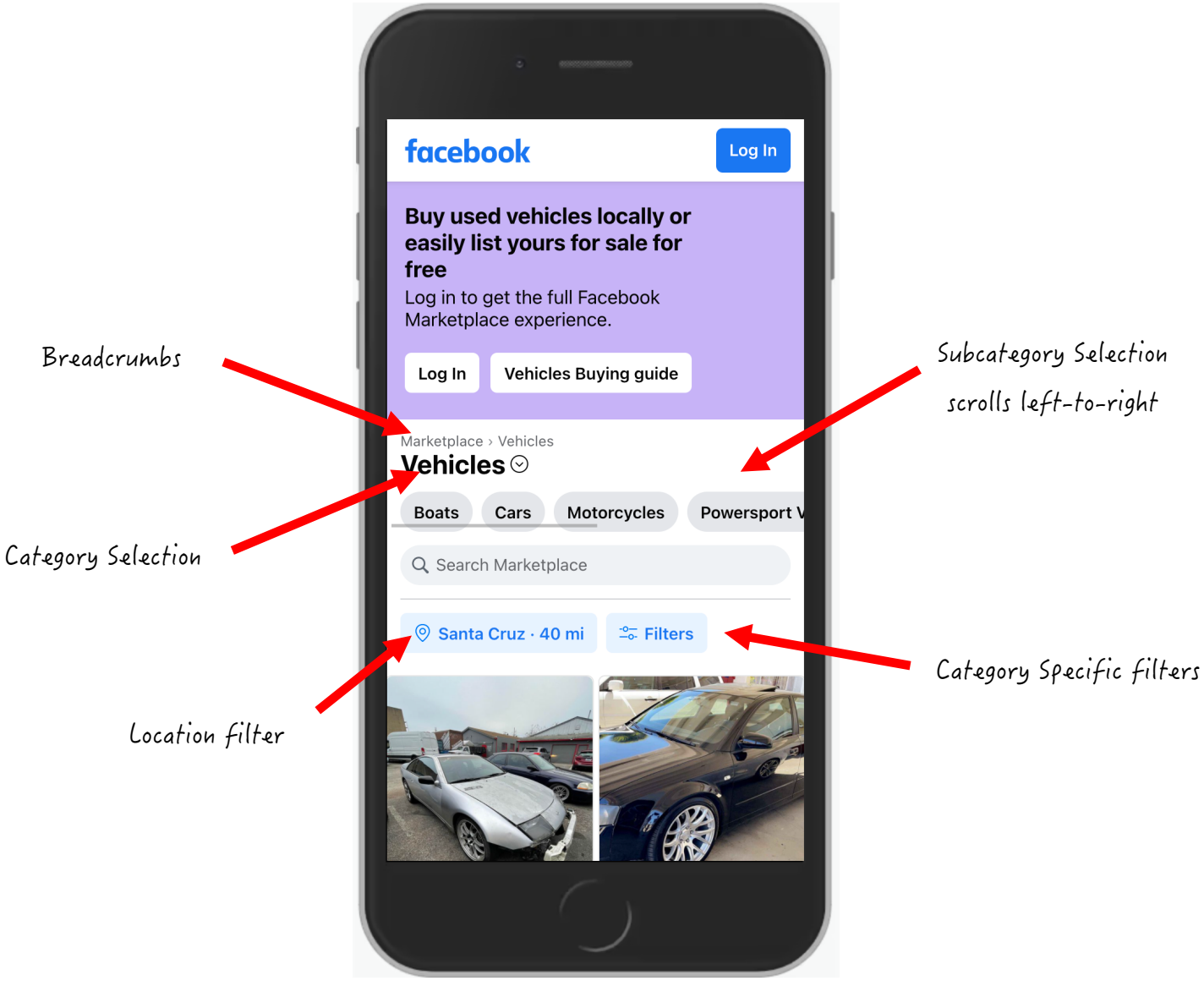
Category Selection

Selecting a Category closes this component and returns to the home page with that category selected



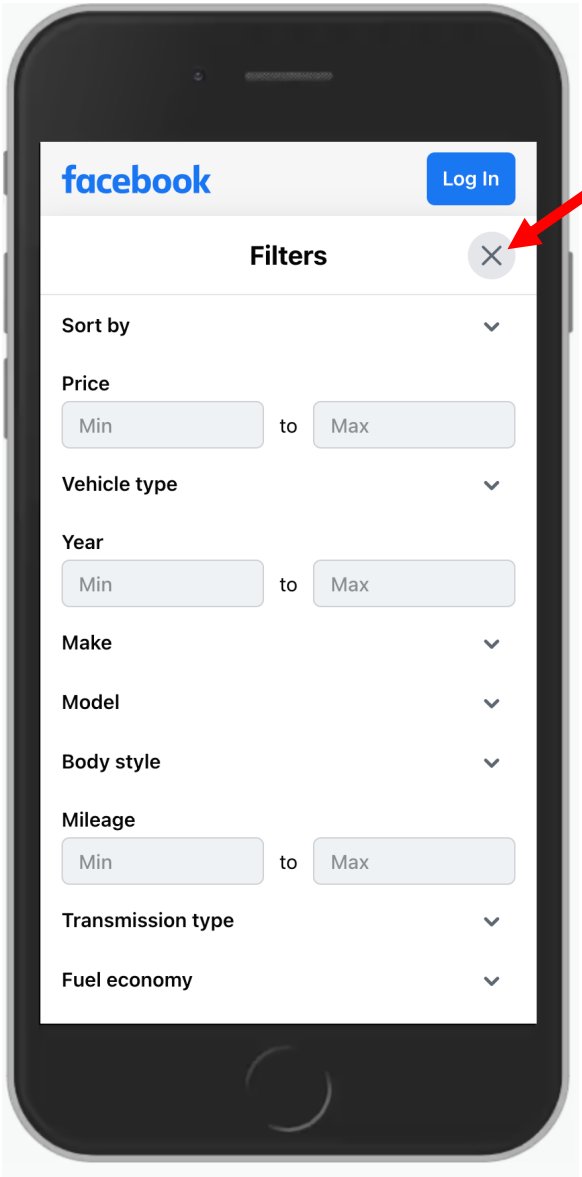
Closes Category Selection without having selected one

Home Page – Category Selected



Category Specific Filters

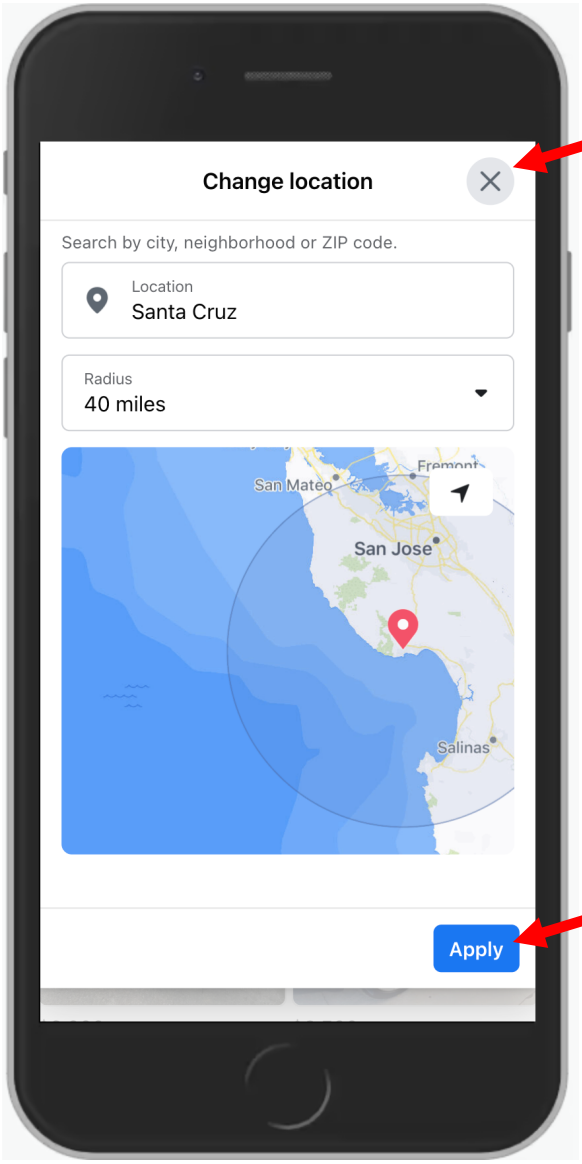
This example is for vehicles, the “apply button is off the screen at the bottom – arguably hot agood UI design, feel free to improve on it.



Closes without having modified the filters

Location Filter

Only attempt this when more basic functionality is complete.

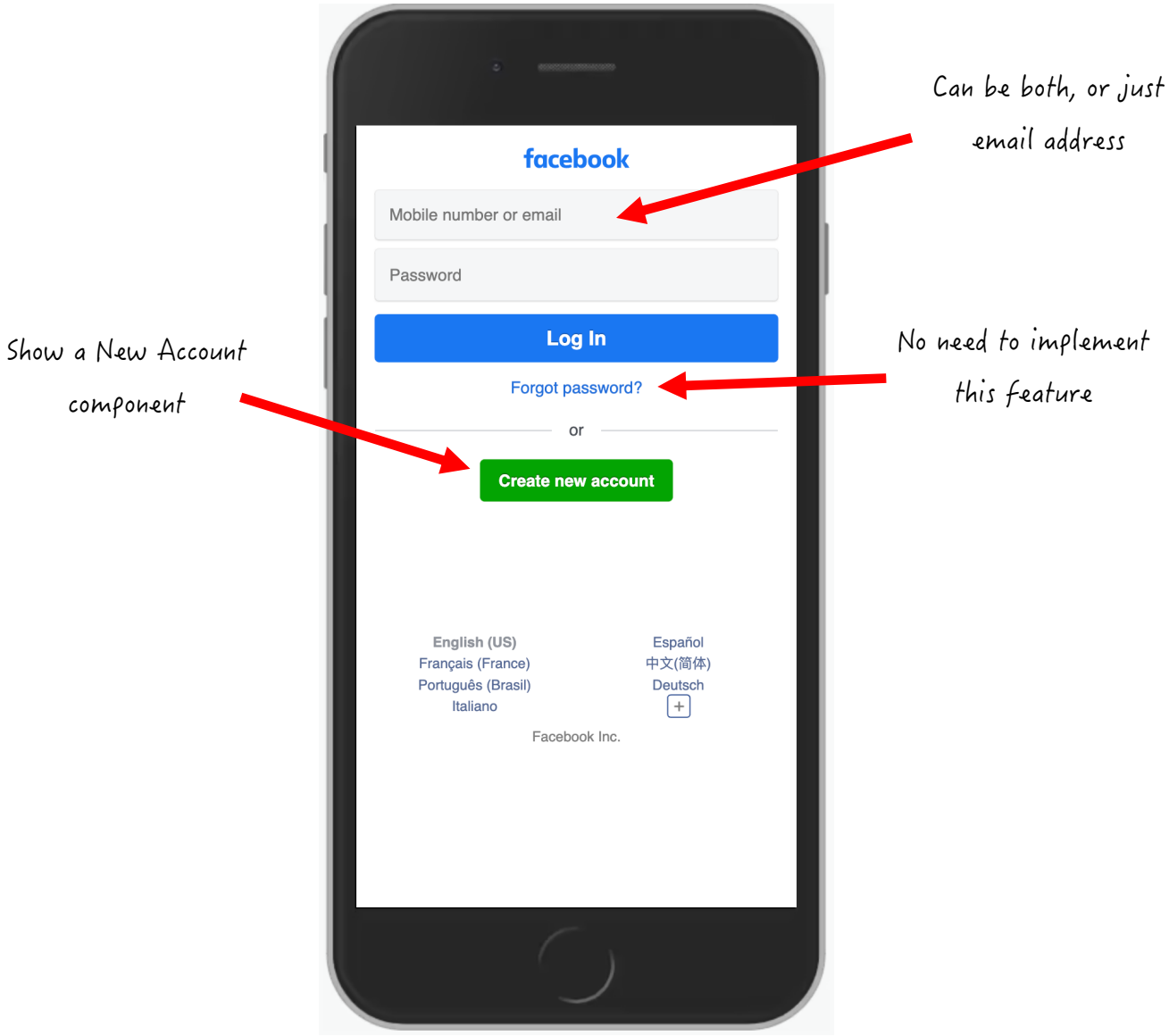


Closes without modifying the location filter

Applies the location filter and returns to the home page

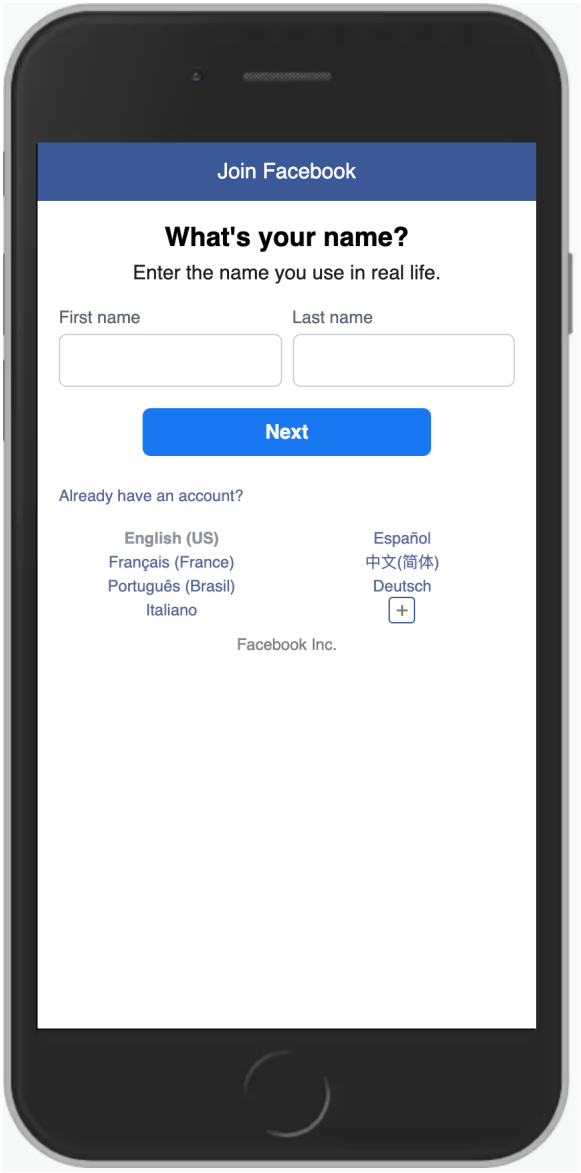
Login Screen

Use The Authenticated Books Example as a guide.



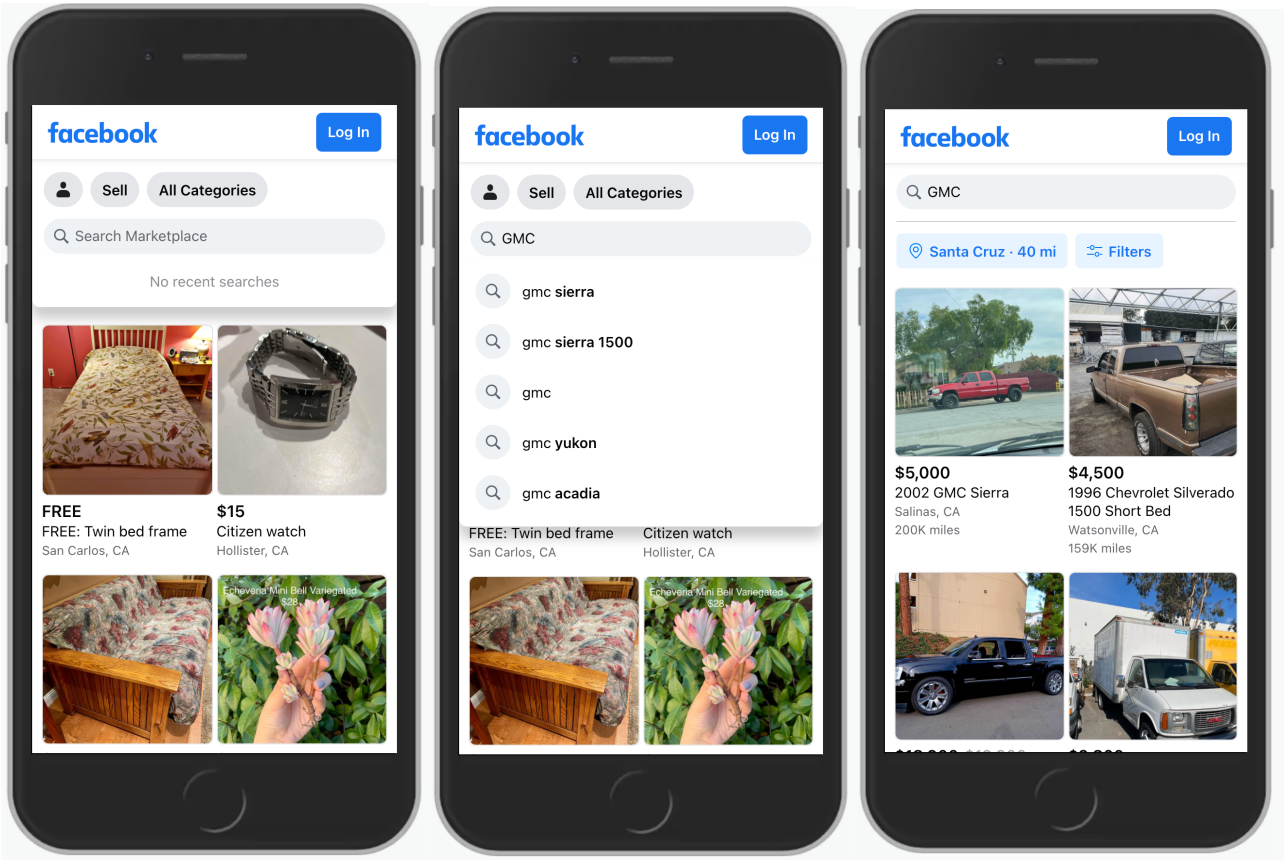
New Account Screen

Follow the “lead-me-by-the-hand” (keep clicking the next button) to get to each stage of registration. Does not need to be that sophisticated, but ask for name first, then email address and telephone number, the password and you’re done.

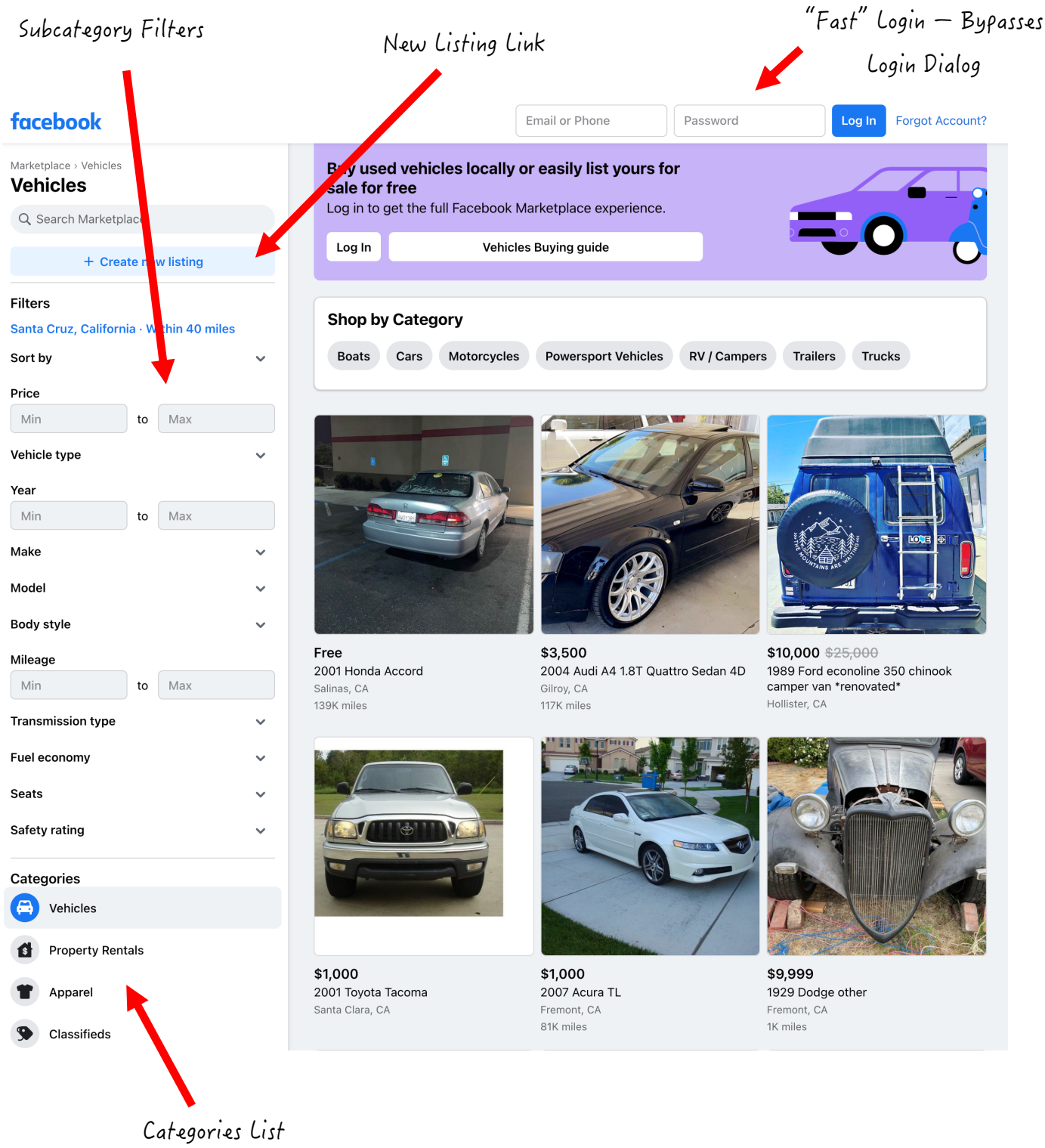


Search

As user types into the search field, suggestions are presented. Here the user is searching for GMC vehicles but note when the results are returned the Vehicle Category is not selected as, in theory, anything in the marketplace could be described as GMC.



Desktop View



Development Guidelines

Whilst you can construct the user interface any way you like, this finished front end must be a Single Page, Material UI, React Application; React Routes may prove useful for the mail reading and composition pages.

Background

Listings in this system can have any properties you deem necessary to complete the assignment.

Note that your API must be constrained by an OpenAPI version 3.0.3 schema. You should endeavor to make this schema as restrictive as possible.

For example, your schema should specify the acceptable format of any e-mail property. If an e-mail property in any other form is presented, your system should reject it simply by performing the validation provided in the starter code.

When you start the backend, use a browser to visit http://localhost:3010/v0/api-docs/ where you can manually test the API as demonstrated in class.

Initially, the only operation available will be:

Which if executed will return the current data and time and the date and time the backend database was initialised.

Database Schema

The supplied database schema can be found in `backend/sql/schema.sql`. It defines a single table '`dummy`'. You will want to define the tables you need in this file.

Optionally, you can also define indexes to be built against your tables in: `backend/sql/indexes.sql`

Resetting Docker

If you run into problems with your dev database, or simply want to re-set it to its initial state, issue the following commands (you will need to be in PowerShell on Windows):

```
$ docker stop $(docker ps -aq)
$ docker rm $(docker ps -aq)
```

What steps should I take to tackle this?

You should base your implementation on your solutions for Assignments 5 and 7 plus the Books Database Example and the Authenticate Books Example.

There is a huge amount of information available on OpenAPI 3.0.3 at https://swagger.io/specification/ and http://spec.openapis.org/oas/v3.0.3. Also consult the Petstore example at https://petstore3.swagger.io/ and make good use of the on-line schema validator at https://editor.swagger.io/.

A good resource for querying JSON stored in PostgreSQL: https://www.postgresqltutorial.com/postgresql-json/

Many Material-UI Examples and sandboxes for experimentation are available: https://material-ui.com/

A plausible development schedule

There are any number of ways of approaching this task, but items that need consideration in approximately the order they need considering are:

- Decide on your data entities
  - For example (not an exhaustive list, not necessarily complete lists of attributes)
    - User
      - Name
      - E-Mail address
      - Password hash
    - Category
      - Name
      - Associated Filters
      - Subcategories
    - Listing
      - User who created it
      - Creation date & time
      - Content
        - Text
        - Image links
      - Replies

- Decide on the end points for you API
  - Login / Authentication
  - Categories
  - Subcategories for a given Category
  - Filters for a given Category
    - You may choose to include these as properties of Categories
  - Listings
    - By Category
    - By Subcategory
  - Search
  - Etc.

- Create UI Components
  - Home Page
  - Login Dialog / Screan
  - Categories List
  - Listings List
  - Filter Panel
  - Distance Filter Dialog
  - Search panel
  - Etc.

Remember, this is only a guideline. You can develop in any sequence you like but do work together as a team. Keep in touch with each other and have a Zoom call at least once a day, preferably two or three times a day.

Most importantly, keep on top of your code coverage. A submission that meets the Basic and Quality Requirements (see below) will score 80% so write tests as you go. A very simple implementation with perfect code coverage is likely to score better than a fancy one with poor code coverage.

How much code will we need to write?

This is a difficult question to answer, but not including your OpenAPI Schema, something in the order of 1000 to 2000 lines of JavaScript might be a reasonable estimate.

Grading scheme

The following aspects will be assessed:

1. (50%) **Basic Requirements**

    - All data is stored in PostgreSQL
    - Server exposes an authenticated OpenAPI restricted RESTful interface
    - UI is reasonably similar to Facebook Marketplace
    - Anyone can view listings
    - Listings can be viewed by category & subcategory
    - Listings can be filtered by category specific attributes
    - Anyone can search for listings
    - Users can sign up
    - Users can login
    - Only logged in users can post new listings
    - Only logged in users can respond to a listing
    - Logged in users can view their listings
    - Logged in users can see which of their listings other users have responded to

2. (10%) **Advanced Requirement**

    - End-to-End tests exist for Basic Requirements
        - Note this requirement is independent of the Quality Requirement below

3. (10%) **Stretch Requirement**

    - Users can geographically restrict their search

4. (30%) **Quality Requirement**

    - No linter warnings/errors 5%
    - Mean of Statement, Branch, Function, and Line Code Coverage
        - 100% 25%
        - >= 99% 15%
        - >= 98% 5%
        - < 98% 0%

5. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

    b. Your submission is determined to be a copy of a past or present student's submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 35% copied code No deduction
        - 35% to 50% copied code (-50%)
        - > 50% copied code (-100%)

        Note that code distributed in class does not count against the copied total but must be cited.

Additional Requirement

You are also required to make a short YouTube video of your Web App demonstrating the features you have successfully implemented.

- Create it under your UCSC CruzID Account - all members of your team should upload it
- Make it “unlisted” - only those with the link can see it
- No more than five minutes
- Demonstrate all the features you successfully implemented
- Screen capture from your laptop is fine
- Keep it simple - no marks for fancy effects and/or editing

Source code repository

Your team must create a private GitLab repository for this assignment. Make all members of the team contributors and invite dcharris@ucsc to join the project as a viewer.

Your repository will be checked for relative contributions so both members of the team must commit changes frequently with a roughly even distribution of work.

What to submit

Run the following command to create the submission archive:

```
$ npm run zip
```

You must also paste a link to your YouTube video into the canvas submission – more details to follow.

****** UPLOAD** `Assignment8.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ******

