

SENG 401 – Final Project - Testing Document

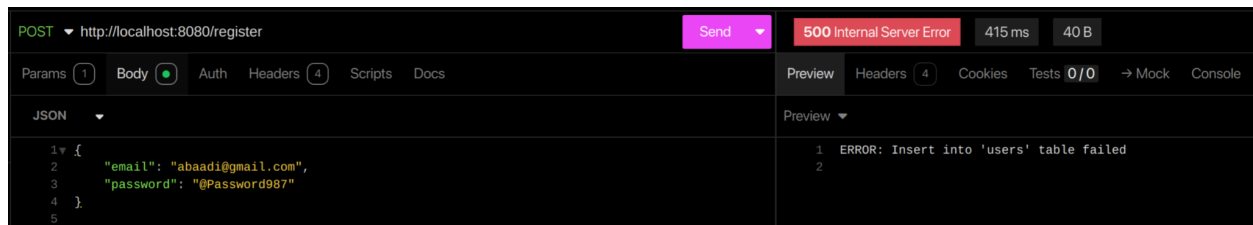
Test Cases/System Tests

API manual unit tests

API connectivity was tested using a suite of API requests that mocked frontend calls on both the local host, and the production cloud server. The testing program, "Obsidian" was used to store and manage these test cases. These tests were used to verify the shape of return data, the network routes, the performance of AI calls (runtime delay), and our API's HTTP error handling outputs. The testing program provided the tester with info relating to header content, cookie use, and HTTP response codes.

Testing Local Register API

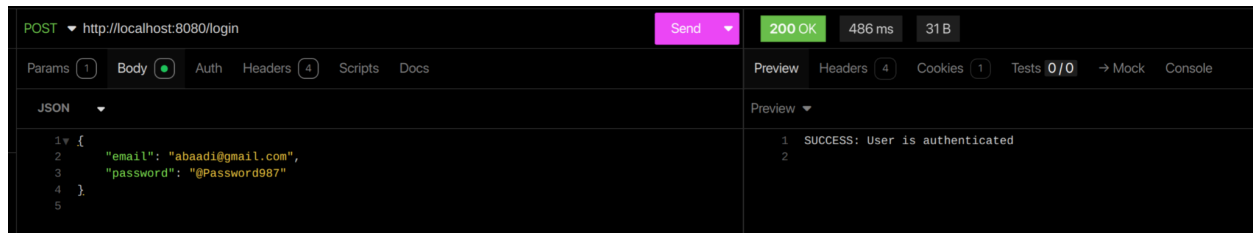
This test posts the register route with an existing user. This throws an error, as this user already exists.



The screenshot shows a REST client interface for a POST request to `http://localhost:8080/register`. The status bar indicates a **500 Internal Server Error** with a response time of 415 ms and 40 B of data. The request body is a JSON object: `{ "email": "abaadi@gmail.com", "password": "@Password987" }`. The response preview shows an error message: `1 ERROR: Insert into 'users' table failed`.

Testing Local Login API

This test posts the login route with an existing user. The credentials are correct, and thus its a success.



The screenshot shows a REST client interface for a POST request to `http://localhost:8080/login`. The status bar indicates a **200 OK** response with a response time of 486 ms and 31 B of data. The request body is a JSON object: `{ "email": "abaadi@gmail.com", "password": "@Password987" }`. The response preview shows a success message: `1 SUCCESS: User is authenticated`.

Team 7

Testing Local AI API #1 (autotrader)

This test posts the AI route using an autotrader input URL. This allows the tester to evaluate the AI data scraping performance, the shape of returned JSON data, and the data types present.

The screenshot shows a REST client interface with the following details:

- POST** `http://localhost:8080/ai`
- Params** (1), **Body** (selected), **Auth**, **Headers** (4), **Scripts**, **Docs**
- Preview** (selected), **Headers** (3), **Cookies**, **Tests** (0/0), **Mock**, **Console**
- JSON** (selected)
- Request Body:**

```
1 {
2   "message":
3     "https://www.autotrader.ca/a/Hyundai/Elantra%20GT/Calgary/AB/5_65656281_20190916130550228/?loc=2A2L&store=20190916130550228&utm_source=google&utm_medium=cp&utm_medium=cp&utm_campaign=roc_sen_en_vla_priority_calgary&utm_campaign=roc_sen_en_vla_priority_calgary&ad_source=1dgc1id=CJ0KQJws-S-BhD2ARtSALss60ak_c-1v9LwG81Tkq2yQmhlz3HUWQDx3yB7bJ3sYARUaok14DgPm1IaAsycEALw_wcB"
4 }
```
- Response Body:**

```
1 {
2   "upload_user_uid": "0195abb-22b2-7411-954c-f6762498b441",
3   "source":
4     "https://www.autotrader.ca/a/Hyundai/Elantra%20GT/Calgary/AB/5_65656281_20190916130550228/?loc=2A2L&store=20190916130550228&utm_source=1dgc1id=CJ0KQJws-S-BhD2ARtSALss60ak_c-1v9LwG81Tkq2yQmhlz3HUWQDx3yB7bJ3sYARUaok14DgPm1IaAsycEALw_wcB",
5   "title": "2018 Hyundai Elantra GT",
6   "price": "18995",
7   "dealer": "dealer",
8   "dealer_rating": "unknown",
9   "odometer": "unknown",
10  "transmission": "Automatic",
11  "drivetrain": "FWD",
12  "desc": "This 2018 Hyundai Elantra GT is a stylish and practical hatchback with a comfortable interior and great fuel efficiency. It comes with a range of features including a touchscreen infotainment system, Bluetooth connectivity, and more.",
13  "specifications": "unknown",
14  "creation_date": "unknown",
15  "free_carfax": "unknown",
16  "vin": "unknown",
17  "condition": "Used",
18  "insurance_status": "unknown",
19  "recall_information": "unknown",
20  "listing_summary": "This listing provides a good overview of the car's features and condition. However, it lacks detailed information such as the odometer reading and dealer rating. Rating: 3.5/5"
21 }
```

Testing Local Login API #2 (kijiji)

This test posts the AI route using a Kijiii input URL. This allows the tester to evaluate the AI data scraping performance, the shape of returned JSON data, and the data types present.

POST

http://localhost:8080/ai

Send

200 OK

468 s

2 KB

3 Days Ago

Params

Body

Auth

Headers

Scripts

Docs

JSON

Preview

Headers

Cookies

Tests

Mock

Console

```

1 {
2   "message": "https://www.kijiji.ca/v-cars-trucks/calgary/2jz-swapped-s13-silvia/1711113307"
3 }

```

```

1 {
2   "response": ""
3 }

```

Team 7

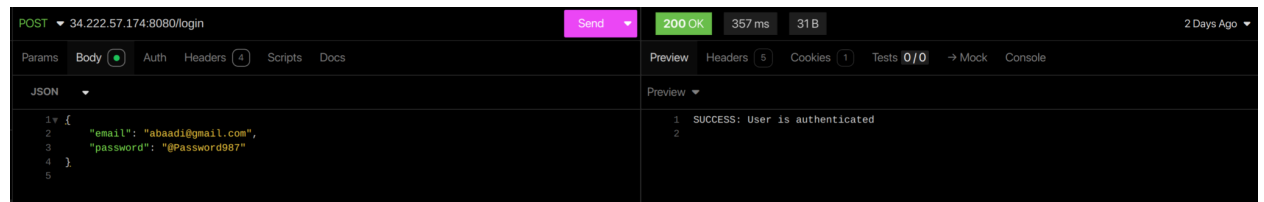
Testing Remote Register API

This test posts the Register API route on the production server. This test is used to test network availability and delay.



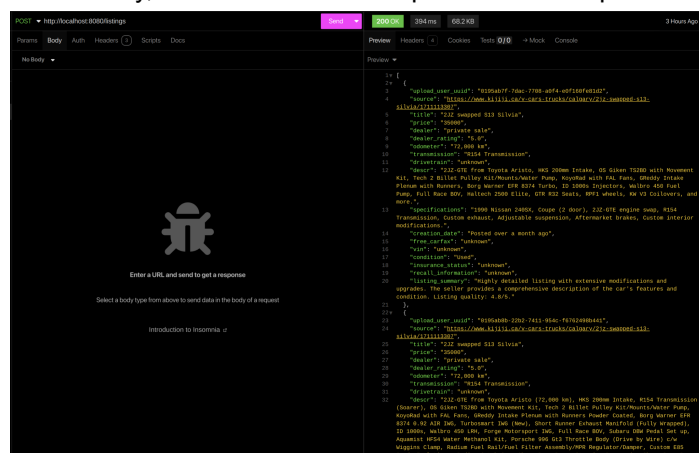
Testing Remote Login API

This test posts the Login API route on the production server. This test is used to test networking performance under a typical feature use case.



Testing Local Database Access API

This test posts the Database fetch route on the local host. This test is used to check remote database accessibility, as the local host API posts the the supabase API URL.

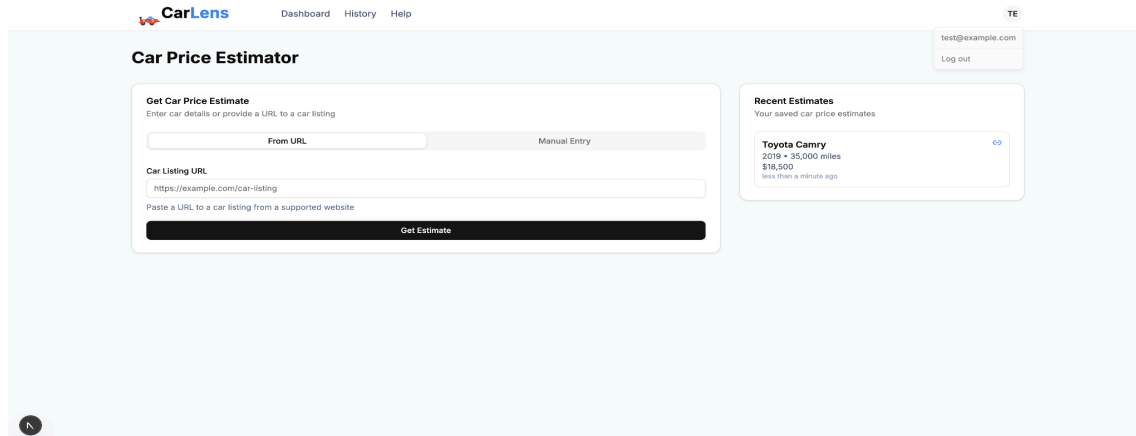


Team 7

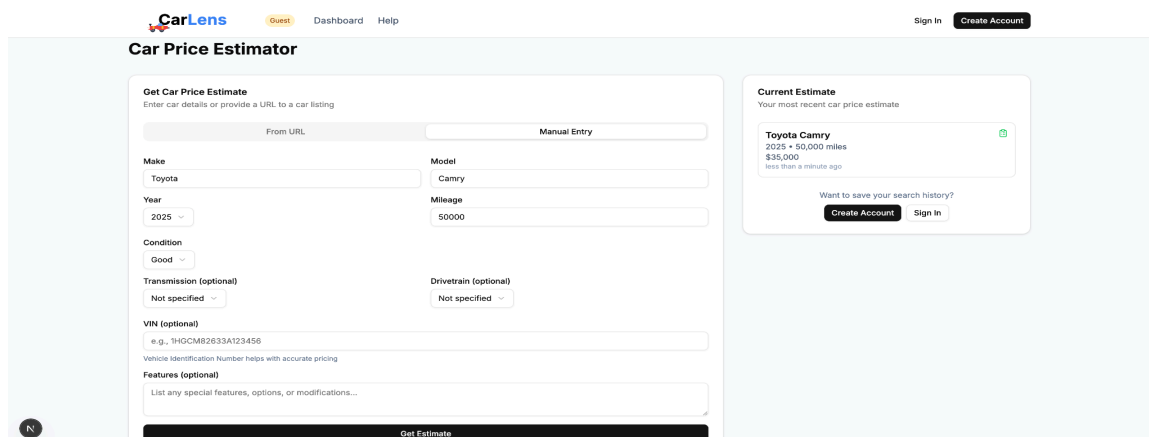
Frontend Testing Using Selenium

The frontend of the application was tested using selenium chrome extension. By recording the screen and tracking button clicks making sure the responses were valid. Through this process we were able to validate that all frontend responses met expectations. A .side file will be provided that can be run in order to replicate the tests.

Testing Avatar Click



Testing Manual Car Entry



Test Data Set

Mock AI response data

Mock AI response data is used to test the frontend. This was performed in order to reduce API ticket wastage, and minimize downtime used to fetch responses. This mock data was provided using a stub API written in GoLang which imitates the port of the production AI.

```
// Main, AIHandler function
func AIHandler(w http.ResponseWriter, r *http.Request) {
    data := map[string]interface{}{
        "upload_user_uuid": "0195ab8b-22b2-7411-954c-f6762498b441",
        "source": "https://www.kijiji.ca/v-cars-trucks/calgary/2020-toyota-rav4-xle/1713552694",
        "title": "2020 Toyota RAV4 XLE",
        "price": "28000",
        "dealer": "private sale",
        "dealer_rating": "5.0",
        "odometer": "300000",
        "transmission": "Automatic",
        "drivetrain": "All-wheel drive (AWD)",
        "descr": "2020 Toyota RAV4 XLE Premium with hail damage. Features include Apple/Android CARPLAY, remote starter, 3M paint protection film, air conditioning, alloy wheels, Bluetooth, cruise control, navigation system, push button start, sunroof",
        "specifications": "Air conditioning, alloy wheels, Bluetooth, cruise control, navigation system, push button start, sunroof",
        "creation_date": "unknown",
        "free_carfax": "unknown",
        "vin": "unknown",
        "condition": "Used",
        "insurance_status": "unknown",
        "recall_information": "unknown",
        "listing_summary": "This listing is detailed and includes multiple photos and a comprehensive description of the vehicle's features."
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(data)
}
```

Test Validation Plan

Model

Ai Model

- The validation of our Ai implementation will be based on its ability to correctly return prompt data given a specific URL or manual user input.
- Reading Inputs:
 - Use temporary API implementations to prototype the Ai calls for our chosen language (GoLang). Early during our development cycle we used "AiTestAPI" to test different prompt combinations to test the Ai model without the need of a formal frontend or network structure.
- Designing Outputs
 - Use a mock/stub API to simulate ideal Ai API outputs to allow for independent testing of frontend components intended to interface with the Ai. Frontend development team does not need to be concerned with Ai implementation details or development progress.

Database

- Design the database layout on paper before implementing any API calls or choosing a provider. Prototype the chosen columns in pseudocode to gauge their potential usefulness and necessity.
- After designing the database, host it on the chosen provider (supabase) and use a mock/Stub api to become familiar with the proprietary database access API libraries/syntax. Study database API outputs
- After studying database to code behavior, design structs and accompanying parsing logic to accommodate the database.

Team 7

View

- Use exploratory testing upon implementing each interactive element on the page. Ensure base functionality of each component/module.
- Populate page with working components and attempt to connect parameters/view logic using React libraries. Use exploratory testing to check pagewide functionality and accessibility.
- Make other group members perform exploratory testing to perform acceptance testing.
- Upon completing the page and its redirects, design basic automated GUI test cases using Selenium. Run test cases after each modification for regression testing.

Controller

API Router

- On paper, design the route layout based on the functional requirements of our system. Decompose these requirements until the most basic routes are defined.
- Set up "shell" APIs in GoLang and test for "200" http code replies to test for connectivity. Use curl and obsidian to check for API connectivity over localhost.
- Host the shell API on a remote server and repeat curl and obsidian unit test cases to verify remote connectivity.
- Merge functionality with Model API logic and test for http response body contents.