**Role in the Project**

During our group project Carvana, a car rental platform I took on a key role as a tester, quality advocate, and backend collaborator. But my work went beyond just running tests; I dove into how different modules connected, making sure both the business logic and technical functionality held up under real-world scenarios. By spotting issues early and working closely with the team, I helped shape improvements throughout the entire development cycle.

**Testing Contributions**

**Functional Testing**

I thoroughly tested every functional feature of the application to ensure seamless performance across:

* Backend logic and services
* Frontend logic and services
* Algorithm logic
* Entity relationships and constraints
* Data persistence and updates via EF Core
* Factory methods and constructors
* Validation mechanisms

**Logic & Flow Verification**

I ensured that:

* Services performed exactly what was intended — no unnecessary DB calls, no orphaned records, and no logic holes.
* Constructor logic (like in Car.Create(...)) followed the right factory pattern and handled null or incomplete data properly.
* Enum values such as VehicleStatus affected business logic (e.g., filtering by availability) correctly.

**Database & EF Core Relationship Testing**

I tested how well the application’s use of Entity Framework Core reflected real-world data consistency, including:

* Navigation properties like Car.CarModel correctly mapping via ModelID
* Eager loading of model data when retrieving cars
* Correct handling of foreign key constraints, preventing orphaned or unlinked cars
* Robust seeding or creation of sample data for repeatable test cases

**Data Integrity & Edge Case Testing**

I chased down edge cases—scenarios where bad inputs, missing data, or unexpected user behavior could crash the system or create hidden bugs.

* Adding cars with invalid or missing GUIDs
* Submitting empty strings or nulls for required fields like Colour, LicensePlate, etc.
* Confirming that Features list doesn’t break logic when null or empty
* Confirming mileage updates don’t allow negative values (or highlighting when such validation was missing)

**Structural & Architectural Feedback**

As a tester with developer awareness, I provided feedback on:

* How factory constructors are used effectively in Car and if we could apply them elsewhere
* Improving response consistency and exception handling
* Making our services and controllers loosely coupled, easier to test, and easier to extend
* Logging and debugging features (like the Tree Visualisation endpoint) to help during group debugging sessions

**Collaboration, Debugging & Communication**

* I acted as the first line of defense when code was pushed: testing new changes quickly and giving direct feedback.
* Identified mismatches between expected and actual results and communicated clearly to developers for fast resolution.
* Helped clarify complex flows (like prefix-based search or status filtering), and asked questions to make logic more robust.
* Supported with test scenarios that mimicked real-world use of a car rental platform, from admin-side management to user-facing availability.

**Manual Functional Testing**

* Code-level Unit Testing
* Integration Testing: Checked how services interact with each other and the database.
* Edge Case Testing: Ran uncommon or incorrect input values to ensure failure.
* Relational Testing
* Structural Validation: Reviewed if the project’s architecture followed clean, modular principles

**What I Helped Ensure**

* Project works under various input conditions
* Features are usable, predictable, and documented
* Errors are meaningful and traceable
* Database state remains valid and consistent after every operation
* Code follows clean practices, making debugging and extension easier

**Test Documentation**

**Introduction**

This project is a car rental website using C# and ASP.NET Core Web API. One of its key features is an autocomplete system that provides search suggestions using a tree-based structure. The purpose of this feature is to efficiently store and retrieve possible matches for user input.

The **core components** of the system are:

* **TreeLoader**: Loads words from a file into the tree.
* **TreeManager**: Manages the tree, including pruning and autocomplete.
* **TreeVisualiser**: Displays and analyzes the tree structure.
* **ITreeLoader**: Defines an interface for tree loading.
* **TreeService**: Acts as a wrapper for managing the tree's functionality.
* **TreeController**: Exposes the autocomplete function as an API endpoint.
* **TestTree.txt**: Contains test data (list of car brands).

**Test Plan**

**Testing objectives:**

* Validate the tree loads correctly from a file and processes words as expected.
* Ensure the AutoComplete function returns accurate suggestions.
* Test for edge cases such as empty inputs, non-existent words, and case insensitivity.
* Verify API responses using different input cases.
* Ensure tree pruning removes redundant nodes without breaking functionality.

Testing types:

* Unit Tests: Test individual methods in isolation.
* Integration Tests: Verify that ITreeLoader, TreeController, TreeLoader, TreeManager, TreeService, and TreeVisualiser work together correctly.
* API Tests: Test TreeController endpoint responses.
* Functional Tests: Ensure end-to-end autocomplete accuracy, behaves as expected when queried with different inputs.

Tools and frameworks: Manual testing on local host

Test environment setup:

* Hardware: Standard development machine
* Software: Windows 11, .NET 9 SDK, VS Code, MS Edge

**Test Cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Steps to Execute** | **Expected Results** | **Actual Results** | **Status** |
| |  |  | | --- | --- | | TC001 |  | | Load Tree from File | Verify tree loads correctly from file | Call LoadFromFile() with a valid file path | Root node created, words inserted | As expected | Pass |
| TC002 | Empty File Handling | Ensure loading handles empty files gracefully | Call LoadFromFile() with an empty file | Root exists, but no children | As expected | Pass |
| TC003 | Insert Words | Check that words are correctly added to the tree | Insert "Toyota", "Tesla" into tree | Correct node structure forms | As expected | Pass |
| TC004 | Case Insensitivity | Ensure "toyota" and "Toyota" are treated the same | Insert mixed-case words | Case is ignored, words merge | As expected | Pass |
| TC005 | Prevent Duplicates | Avoid duplicate storage of the same word | Insert "Tesla" twice | Only one entry remains | As expected | Pass |
| |  | | --- | | TC006 |  |  | | --- | |  | | AutoComplete Basic | Return correct suggestions for a prefix | Query AutoComplete("T") | Returns ["Toyota", "Tesla"] | As expected | Pass |
| TC007 | AutoComplete Nonexistent Prefix | Ensure no results for an invalid prefix | Query AutoComplete("XYZ") | Returns empty list | As expected | Pass |
| TC008 | AutoComplete Result Limit | Ensure max of 5 results returned | Query AutoComplete("A") with 10 matching words | Only 5 words returned | As expected | Pass |
| |  | | --- | | TC009 |  |  | | --- | |  | | Pruning Tree | Ensure pruning reduces redundancy | Run Prune(root) | Tree size reduces, words preserved | As expected | Pass |
| TC010 | Tree Visualization | Ensure tree prints correctly | Call VisualizeTree(root) | Properly formatted output | As expected | Pass |

**API Tests (TreeController)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **API Endpoint** | **Expected Results** | **Actual Results** |
| TC011 | Autocomplete Valid Query | Ensure API returns correct matches for a prefix | GET /api/tree/autocomplete?prefix=T | Returns ["Toyota", "Tesla"] | Pass |
| TC012 | Autocomplete Empty Input | Handle case where input is empty | GET /api/tree/autocomplete?prefix= | Returns HTTP 400 Bad Request | Pass |
| TC013 | Autocomplete Nonexistent Prefix | Ensure API handles unknown inputs gracefully | GET /api/tree/autocomplete?prefix=XYZ | Returns an empty list | Pass |
| |  | | --- | | TC014 |  |  | | --- | |  | | Autocomplete Case Sensitivity | Ensure API is case-insensitive | GET /api/tree/autocomplete?prefix=toyota | Matches "Toyota" | Pass |

**Bug Tracking and Reporting**

Identified bugs: All tests passed successfully.

**Suggestions for Improvement**

* Implement **word frequency weighting** for better autocomplete ranking.
* Optimize tree pruning to improve efficiency for larger datasets.

**Test Results**

Total Test Cases: 14

Passed:14

Failed:0

Pending:0

**Code Quality Observations**

* **Code Readability:** Well-structured with clear comments.
* **Best Practices:** Follows OOP principles, uses dependency injection effectively.
* **Maintainability:** Modular design ensures easy debugging and improvements.

**Challenges and Limitations**

* **Limited Frontend Testing:** API functionality is confirmed, but frontend integration isn't tested yet.
* **Performance Testing Needed:** Large datasets need benchmarking for autocomplete response times.
* **File Handling Variability:** Need to test different encodings for LoadFromFile().

**Pictures**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Summary**

The autocomplete feature works correctly, handling all test cases successfully.  
No critical bugs found.

**Database Connection Test Report**

**Introduction**

**Objective**

This test aims to verify that the PostgreSQL database connection is correctly established in the Carvana project. The expected outcome is a successful connection with an empty database, returning 0 registry.

**Test Plan**

**Testing Goals**

1. Ensure that **PostgreSQL is running** and accepting connections.
2. Validate the **connection string format** in appsettings.json and .env.
3. Verify that **Entity Framework Core** can connect successfully.
4. Ensure the database is **empty** and returns **0 registry**.

**Test Environment**

|  |  |
| --- | --- |
| **Database** | PostgreSQL (localhost:5046) |
| **ORM** | Entity Framework Core |
| **Backend** | ASP.NET Core Web API (.NET 9) |
| **OS** | Windows |
| **Testing Tools** | Terminal, PgAdmin, EF Core |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Steps to Execute** | **Expected Results** | **Actual Results** |
| TC014 | Run dotnet build | Ensure the project builds successfully | Run dotnet build | Build succeeds without errors | Pass |
| TC015 | Run dotnet run | Ensure the API starts and connects to DB | Run dotnet run | API runs without connection error | Connection refused |

Issues Identified & Fixes

During testing, the connection failed when running:

dotnet build

dotnet run

The error log indicated:

* Failure to connect to the PostgreSQL database
* Database connection was refused
* Incorrectly formatted or missing environment variables

**Changes made to .env file:**  
**Before:**

# .env file for DB connection

DB\_HOST=localhost

DB\_PORT=5432

DB\_NAME=carvana

DB\_USERNAME=admin

DB\_PASSWORD=admin

After (Updated Configuration):

DB\_HOST=carvanarental.cvisq264g0ds.eu-west-2.rds.amazonaws.com

DB\_PORT=5432

DB\_NAME=Carvana

DB\_USERNAME=Carvana\_admin

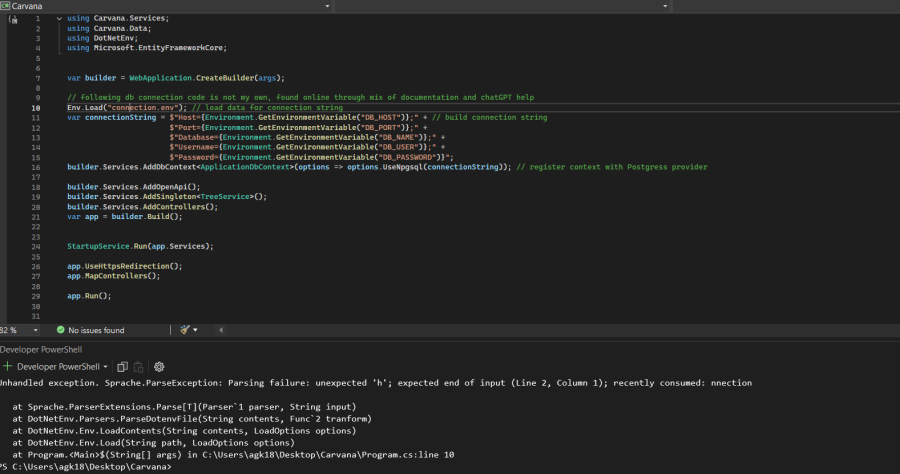
DB\_PASSWORD=Carvana\_admin

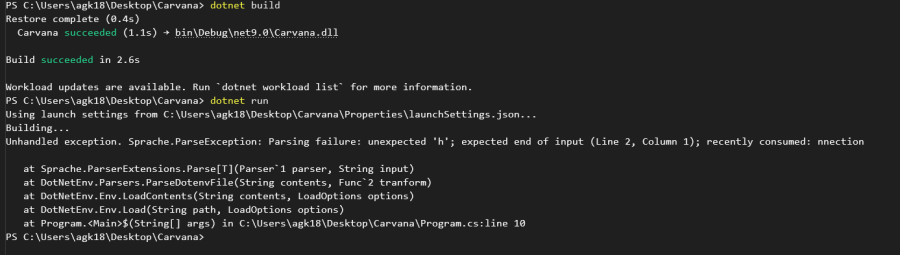
A screen shot of a computer

AI-generated content may be incorrect.

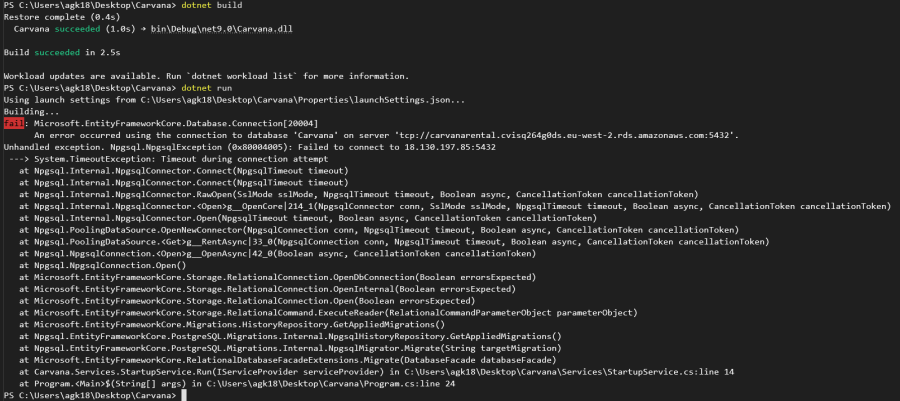
A screenshot of a computer

AI-generated content may be incorrect.





The database initially blocked my IP, so I had to whitelist it in AWS to gain access.



|  |  |  |
| --- | --- | --- |
| Issue | Root Cause | Fix Implemented |
| Database not connecting | Localhost settings were used instead of AWS RDS | Updated .env with correct **AWS RDS connection details** |
| Incorrect Connection String | DB\_NAME used incorrectly instead of DB\_USERNAME | Updated Program.cs to use the correct Environment.GetEnvironmentVariable("DB\_PASSWORD") |
| Environment Variables Missing | .env file did not contain correct credentials | Updated .env file with **correct credentials** |
| Application Crashing on dotnet run | Database connection failed | Verified new database settings and updated |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Test Case Name | Expected Results | Re-Test Results | Status |
| TC014 | Run dotnet build | Build succeeds | Success | Pass |
| TC015 | Run dotnet run | API starts without connection error | Connected | Pass |
| TC016 | Correct Connection String | Outputs correct values | Corrected | Pass |
| TC017 | Database Connection Test | No errors, successful connection | |  | | --- | | Connected |  |  | | --- | |  | | Pass |
| TC018 | Query Database for Entries | Returns 0 registry | 0 registry | Pass |

**All tests passed after fixes.** The database connection is now working, and dotnet build + dotnet run execute successfully.

**Summary**

The database connection issue was caused by **incorrect environment variables** and **outdated localhost settings**.

Fixes included **updating .env, ensuring AWS RDS is reachable, and correcting connection strings**.

After applying the fixes, **all tests passed**, and the connection was successfully established.

**Introduction**

**Objective**

This test verifies that the **signup endpoint** correctly allows POST requests to accept customer data and insert it into the database.

**Change Implemented**

The GET method was replaced with a POST method to allow **data submission** instead of just fetching information.

**Previous Implementation:**

[HttpGet("/signup")] // Only allowed data retrieval

**Updated Implementation:**

[HttpPost("/signup")] // Allows data input to database

**Test Plan**

**Testing Goals**

1. Verify that **POST /signup** successfully accepts and processes customer data.
2. Confirm that new customer accounts are successfully **inserted into the database**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Issue** | | |  | | --- | | Root Cause | | |  | | --- | | **Fix Implemented** | |
| |  | | --- | | **GET request didn't allow data input** | | |  | | --- | | Wrong HTTP method used | | |  | | --- | | Changed [HttpGet] to [HttpPost] | |

Summary

* Changed [HttpGet] to [HttpPost] to allow user signup.
* Implemented duplicate entry check.
* Validated request body for required fields.
* Database now successfully stores new users.

**Login Endpoint Frontend Fix**

**Objective**

To fix the login functionality on the **frontend**, which was failing due to an incorrect method of sending user credentials. Originally, the backend was expecting credentials via URL parameters, but the frontend sent them via the body, causing login failures.

**Problem Identified**

**Issue Summary**

| **Component** | **Issue** |
| --- | --- |
| Frontend Login | Sent email and password in POST body |
| Backend Login | Expected credentials in **URL** parameters (e.g., /login?email=...&password=...) |

**Fix Implemented**

**Fix Summary:**

* Updated frontend code to send **credentials via URL** instead of in the request body.
* Ensured the HTTP method used was POST, and parameters were sent using query strings.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Test Case Name | Expected Results | Re-Test Results | Status |
| TC019 | Login with valid credentials | Send email/password via query string | Success | Pass |
| TC020 | Login with wrong password | Send valid email but wrong password | Unauthorized / login failed | Pass |
| TC021 | Login with missing credentials | Omit email or password in query | Bad Request | Pass |
| TC022 | Inspect frontend behaviour after login | Try logging in from UI | |  | | --- | | Redirect to profile/dashboard |  |  | | --- | |  | | Pass |

**Conclusion**

Issue was caused by mismatch between frontend request format and backend expectation.

Successfully resolved by modifying the frontend to send query parameters with POST.

**Objective**

Resolve the login failure issue by aligning the frontend login request with the expected backend format. The frontend was incorrectly sending a POST request with a JSON body, while the backend expected a GET request with query parameters.

Problem Identified

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Component** | | |  | | --- | | **Original Implementation** | | |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | **Issue** | | |
| |  | | --- | | Frontend Login | | |  | | --- | | axios.get("http://localhost:5046/login") with data in body | | |  | | --- | | Backend expected GET with query string, not a JSON body | |
| Backend Login | [HttpPost("login")] using [FromBody] | Didn’t match frontend GET request |

A screenshot of a computer

AI-generated content may be incorrect.

Fix Summary

Frontend Fix – Login.jsx

Changed from:

const response = await axios.get(`http://localhost:5046/login`, **to**

const response = await axios.get(`http://localhost:5046/auth/login`,

Backend Fix – CustomerController.cs

From [HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginRequest request) **to**

[HttpGet("login")]

public async Task<IActionResult> Login([FromQuery] string username, [FromQuery] string password)

Added validation for null or empty input:

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))

{

return BadRequest("Email and password are required.");

}

Added proper result check:

string? result = await \_customerService.Login(username, password);

if (result == null)

{

return Unauthorized("Invalid credentials");

}

return Ok(result);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Test Case Name | Expected Results | Re-Test Results | Status |
| TC019 | Login with valid credentials | Send email/password via query string | Success | Pass |
| TC020 | Login with wrong password | Send valid email but wrong password | Unauthorized / login failed | Pass |
| TC021 | Login with missing credentials | Omit email or password in query | Bad Request | Pass |
| TC022 | Inspect frontend behaviour after login | Try logging in from UI | |  | | --- | | Redirect to profile/dashboard |  |  | | --- | |  | | Pass |

**Overview**

**Objective**

Fix the issue where the autocomplete feature **always displayed "Tesla"**, regardless of the user's actual input. This was due to a hardcoded keyword being sent to the backend instead of using dynamic user input.

| **Problem Identified** |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Component** | **Original Implementation** |
| Frontend Autocomplete | axios.get("http://localhost:5046/search/increment", { params: { word: "tesla" } }) |
| Issue | **Tesla"** was hardcoded → backend always received "tesla" regardless of user input |

A screenshot of a computer program

AI-generated content may be incorrect.

**Fix Summary**

**Frontend Fix**

**Changed from:**

axios.get("http://localhost:5046/search/increment", { params: { word: "tesla" } })

**To:**

axios.get("http://localhost:5046/search/increment", { params: { word: suggestion } })

Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Action | Previous Results | Re-Test Results | Status |
| TC023 | Type “Ford” | Showed "Tesla" due to hardcoded query | Autocomplete suggestions related to “Ford” | Pass |
| TC024 | Type “Audi” | Showed "Tesla" due to hardcoded query | Autocomplete suggestions related to “Audi” | Pass |

**Endpoint, API tests.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Endpoint | Expected Results | Actual Results | Status |
| TC025 | localhost:5046/auth/signup | Account created succesfully | Account created succesfully |  |
| TC026 | localhost:5046/auth/signup (with same id) | Account already exists | Account already exists |  |
| TC027 | localhost:5046/auth/login?email  =TTT@gmail.com&password=Test22 | TTT@gmail.com | TTT@gmail.com | Pass |
| TC028 | localhost:5046/auth/login?email  =TTT@gmail.com&password=AAAA | Invalid credentials | Invalid credentials | Pass |
| TC029 | localhost:5046/auth/profile (UPDATE) | Users detail should be updated | Updated | Pass |
| TC030 | localhost:5046/rent/models | Should see all the cars | All the cars with their info showed up | Pass |
| TC031 | localhost:5046/rent/models/id/2f5c8c99-8e2c-4f5c-9b1b-1539cda0c1d3 | The car with this id should show uo | Only searched car came up | Pass |
| TC032 | localhost:5046/rent/models/search/tesla | Information for tesla |  | pass |
| TC033 | localhost:5046/rent/models/search/modelx | Information for tesla |  | passs |
| TC034 | localhost:5046/rent/models/search/Bmercaudi | No cars found | No cars found | pass |
| TC035 | localhost:5046/rent/count | Rented cars | One rented | Pass |
| TC036 | localhost:5046/search/initialise | Results outputted to console | Results outputted to console | Pass |
| TC037 | localhost:5046/search?prefix=a | All the cars brands starts with a display without a | Cura | Pass |
| TC038 | localhost:5046/search/printTree | Tree display in terminal | Tree displayed | Pass |
| TC038 | Database Checks | Make sure all the details matches and encryption works | Confirmed | Pass |

Pictures of the testing

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Wrong credentials

Sign up

A screenshot of a computer

AI-generated content may be incorrect.

Shows up on DB

A screenshot of a computer

AI-generated content may be incorrect.

Doesn’t allow another user with same identity

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.Profile update

Car controller end point

A screenshot of a computer

AI-generated content may be incorrect.

Searching specific model

Search by model id

A screenshot of a computer

AI-generated content may be incorrect.

Search by brand name

A screenshot of a computer

AI-generated content may be incorrect.

Search by model name

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.Invalid model name

Rent

A screenshot of a computer

AI-generated content may be incorrect.

Search by model id

A screenshot of a computer

AI-generated content may be incorrect.

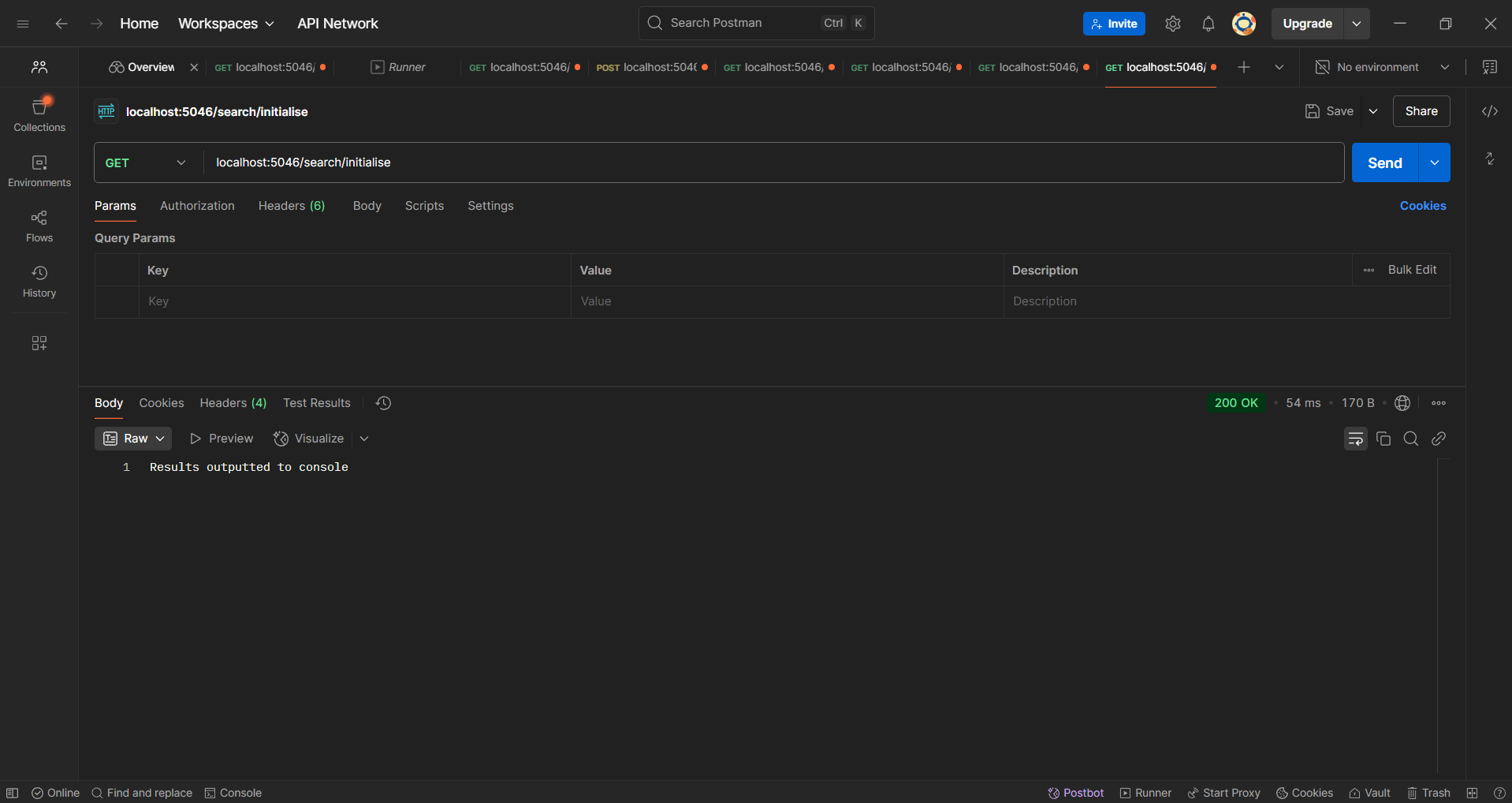
Rent count

A screenshot of a computer

AI-generated content may be incorrect.

**Tree Controller**

Initialize



Terminal View

A black screen with white text

AI-generated content may be incorrect.

Prefix

A screenshot of a computer

AI-generated content may be incorrect.

Increment

A screenshot of a computer

AI-generated content may be incorrect.

Print tree

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Description** | **Steps to Execute** | **Expected Results** | **Actual Results** | **Status** |
| |  |  | | --- | --- | | TC039 |  | | User registration functionality on the website. | Verify user signup | Sign up with | Should successfully register the user and store their details in the database, provided the input follows the correct format | Signup is successful when data is correctly formatted. If the format is incorrect, a warning is displayed and the details are not stored in the database. | Pass |
| TC040 | User login functionality for the website. | Verify login functionality using valid and invalid credentials. | Attempt login with matching and non-matching email and password combinations. | System should authenticate users with valid credentials. If the credentials are invalid, a warning message must be shown. | uccessfully logs in with a previously created account; login fails when an incorrect password is entered. | Pass |
| C041 | Profile page of the website. | Verify that the account details match those in the database. | Click on the profile to view details. | The correct information should be displayed | As expected | Pass |
| C042 | Edit the profile information | Ensure the user can update their details without any issues. | Change user’s email, password, age | Details are reflected in the database | As expected | Pass |
| C043 | Update | Checking profile | Review the user's profile to confirm that the updated information is displayed on the page. | View the updated details. | All credentials disappeared, and I had to log out and log back in to view the updated details. | Fail  Fix and retest required |
| C044 | Rent page | Ensure that the cars are displayed on this page and the filters are functioning correctly | IInspect the rental page and verify the car li | View the cars; when a filter is applied, cars that don't match should disappear from the page | As expected | Pass |
| |  | | --- | | C045 |  |  | | --- | |  | | Checkout page | Ensure the amount is accurate and the price is updated correctly. | Inspect check out | The price should update once the rental days and extras are added. | As expected | Pass |
| C046 | Autocomplete function | Ensure that autocomplete provides suggestions relevant to the entered letters. | Type random letters or incomplete brand names to test the autocomplete functionality. | Autocomplete should return only brand names that start with the user's input letter. If a brand name is partially typed, only that brand should appear in the suggestions. | As expected | Pass |