
Web Development Report

Tudor Haba
2420645

Client and Server-side Scripting

Client-side scripting is code that requires a web browser to run whereas server-side scripting is the code that runs on the server and then it returns the data ready to be rendered by the browser. Client-side programming languages like JavaScript are included in the HTML code to be processed by the browser. The advantage of client-side scripting is that it is efficient and runs faster than server-side scripting. Some uses of client-side scripting are reacting to client-side event such as user interactions and validating data directly on the browser without any server interactions. However, the drawback to client-side scripting is that it significantly reduces the security of the app, all of the code is exposed to the browser and easily viewable by the user through the use of the developer tools every browser allows [1][2].

Since server-side scripting executes the code before the HTML is sent to the browser the code stays hidden, adding a layer of extra security. Server-side scripting uses languages like PHP, Node JS, Perl and is able to interact with databases to retrieve data and perform other databases operations. One disadvantage to this would be that you would send a request to the server for every parse or interaction the page sends, which could slow down response time, making it slower when compared to client-side scripting [1][2].

```
<input
  id="price-numberbox"
  name="price-numberbox"
  type="number"
  onChange={(e) => setPrice(e.target.value)}
/>
```

An example of client-side scripting is the use of React's useState to update the price based on user input.

```
@PrePersist
@PreUpdate
private void validate() {
    if (isInappropriate(licensePlateNumber)){
        throw new IllegalArgumentException("License plate number contains inappropriate language.");
    }
    if (!correctLength(licensePlateNumber)){
        throw new IllegalArgumentException("License plate number has an invalid length.");
    }
    if (price < 0){
        throw new IllegalArgumentException("Price cannot be negative.");
    }
}
```

An example of server-side scripting is the use of Hibernate's tags to validate the entity before saving it to the database.

Node JS

Node.js is a cross platform run time environment that can be used to create full stack web applications [3]. It works by using the JavaScript language for the backend development of the web server as well as allowing the use of it for the frontend. Using node.js would make it easier for developers to switch between writing code for the backend to the frontend because both use the same language. Previously a full stack developer had to learn multiple languages for the backend and for the frontend which has now been simplified with the use of node.js. Node.js is open source and it is built on the V8 engine by Google that converts JavaScript code to machine code that is written in C and C++ for high performance execution. Node.js allows for asynchronous operations meaning it can handle multiple requests at the same time using a single thread. This makes it more scalable than other languages, which did not allow the start of a new request until the previous has finished or unless they had multi-threading [3]. While this is a benefit for node.js it can also be a big drawback since the use of a single thread can cause problems if too many requests are sent at once actually making the program run slower. As it uses the CPU to perform the requests, too many requests can strain the server's CPU. While it is a good tool for experts, node.js is hard for beginners to learn as there are complex topics such as callbacks and asynchronous operations which if used incorrectly can lead to callback hell. This is when there are too many call backs nested within each other and each callback relies on output from a previous callback, making the code complicated to debug and maintain [3].

Joint Assignment

User manual

To start the application first launch the RESTWebServiceApplication by clicking the run button in IntelliJ. Then cd into the Server folder within the ReactApp and start the server using 'npm start'. This will start the application on port 8081. To access the application, go to <http://localhost:8081> or follow the link from the terminal.

Application

For the assignment, the front end was implemented using React, and the server was built using NodeJS. The application is split into 2 views depending on the user's role. The users can see a list of all the license plates on the system, use the partial search function to look for license plates that might be of interest, add balance to their account in order to purchase license plates. They can also view a list of all the license plates they own. The other role implemented in the application, admin, is able to do all the things listed above the normal users can do as well as add new license plates to the system and view a purchase log of all the license plate purchases that have happened. Apart from the features listed there is a login page that allows users to login into the application and a

register page where the users can register a new account. On the register page there is the option to select whether the user is an admin, this is to demonstrate the functionality and to make it easier to set up an admin account, instead of setting it up from Bruno.

One of the most interesting aspects of the application is the use of transactions in the web service. The transaction is used when a user purchases a license plate. A transaction was used because purchasing a license plate will have to update the user account to assign the license plate to the user, it will also have to update the license plate table to add the account to the license plate and it will also have to create a new entry in the purchase log table with the details and date of the purchase. A transaction has allowed it to do all these operations with a single request from the application and it also offers the benefit that if one operation fails then everything will be rolled back to the previous state.

Validation is present in the application, both on the backend and frontend. The application uses frontend validation to check when a user adds balance to their account. It checks that the card number and cvv match the required length. The rest of the validation happens on the backend. The application validates a license plate before it stores it into the database, and it throws clear errors that are used by the frontend to display back to the user. Some of the other validation happens automatically with the use of Hibernate's column tags that can specify to only allow unique items to be stored to the database.

Testing

To run the tests first cd into the Client folder then do 'npm test'. Jest was used for testing along with mocked Axios requests to simulate interactions with the endpoints. The Get Account tests the user login and is broken down into 2 tests. One test, get account successful, simulating the correct credentials have been entered and the endpoints correctly returns the account. The other tests, Get Account failure, tests that nothing gets returned if the wrong credentials have been entered. The other test, Register Accounts, tests the post method to the database through the endpoints and tests that when successful the accounts has been correctly added, and nothing happens if any errors are present. Update Balance tests the patch operation and checks that the balance has been updated after a mocked call has been made.

```
PASS src/__tests__/clients/AccountClient.test.js
  Get Account
    ✓ Get account successful (2 ms)
    ✓ Get account failure (20 ms)
  Register Account
    ✓ Register account successful
    ✓ Register account failure (1 ms)
  Update Balance
    ✓ Update balance successful (1 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:  0 total
Time:        0.587 s
Ran all test suites.
```

Test results - passing all the implemented tests.

The other clients, LicensePlateClient and PurchaseClient, have also been fully tested. All of the tests are similar to the ones described in more detail above. They test that the correct data gets returned when a call has been made to the specific endpoint.

```
Test Suites: 3 passed, 3 total
Tests:       11 passed, 11 total
Snapshots:  0 total
Time:        0.491 s, estimated 1 s
```

Fully tested application.

References

- [1] J. Zaner, 'Taking Your Web Site to the Next Level', *Tech Directions*, vol. 66, no. 2, Prakken Publications, Inc., Ann Arbor, United States, pp. 16–19, Sep. 2006.
- [2] Anonymous, 'Scripting: Server-side vs. client-side', *Act. Serv. Dev. J.*, vol. 2, no. 9, pp. 15–16, Sep. 1998.
- [3] B. Basumatary and N. Agnihotri, 'Benefits and Challenges of Using NodeJS', *Int. J. Innov. Res. Comput. Sci. Technol.*, pp. 67–70, May 2022, doi: 10.55524/ijircst.2022.10.3.13.