CSC 648/848: Software Engineering

RecoveryNote

Everything you need for school in one place

Section 04 Team 06

Joel Giannelli - Team Lead
Hongjie Li - Front End
Tamir Rasheed - Scrum Master
Ivan Cebreros - Back End
Kuncheng Wu
Nyambayar Purevtseren

Milestone 4

Revision ID	Revision Date
Milestone 4 v1	**11/28/21**

<u>**1) Product summary**</u>

- <u>**Product Name:**</u>
  - RecoveryNote
- <u>**All Major Committed Functions:**</u>

Guest User:

1. Guest Users can create a unique account.
2. Guest Users can browse the store and see what supplies are available.
3. Guest Users can look up the supplies by selecting the departments it belongs to.
4. Guest Users can log in as account users.

Account User:

1. Account Users can log in with a unique account.
2. Account Users can log out of the account.
3. Account Users can browse the store.
4. Account Users can look up the supplies by selecting the departments it belongs to.
5. Account Users have a shopping cart.
6. Account Users can fill up a shopping cart with supplies they want to purchase.
7. Account Users can put in information needed to finalize purchases(payment method and address).
8. Account Users can choose to save information for future purchases.
9. Account Users can leave reviews on items that are sold multiple times.
10. Account Users can put up items to sell on the website.
11. Account users who put items to sell on the store shall put a title of the item, a description of the item, when the item was previously used (or if never used), and a price of the item.
12. Account Users who put items to sell on the store can track the item's status(selling or sold).
13. Account Users who save items on shopping carts can access the checkout page and place the order.

- <u>**Unique Product Features:**</u>
  - RecoveryNote provides students with a platform to resell their school supplies, notes, and books. It allows students(users) to create posts of their items and manage those posts. Users can also buy items from others. RecoveryNote is entirely focused on providing a platform for students to sell their old (or unused) textbooks, notes, and other supplies. It creates a space for all students to take advantage of what other students have to offer, all while saving a bit of money compared to buying a textbook brand new. It also provides students with a way to gain a return on the notes they take, as they can provide extra help for the next student who decides to purchase them.
  - Students can select their respective school and department for their school when searching.
  - Students can also upload images through the cloud on our site, providing ease of use when creating a post.

- **Product URL:**
  - http://ec2-54-183-15-93.us-west-1.compute.amazonaws.com/

2) QA testing- max 2.5 pages

All testing scripts can be found at the following location on our github.

Note that each folder in the testing folder exists as it's own node project with its own unique dependencies needed to execute the individual tests.

https://github.com/CSC-648-SFSU/csc648-fa21-04-Team06/tree/master/testing

2.1 Unit Test

All three of our user profiles shared the common thread of a student looking for a deal on school supplies. Both new and existing users would follow a similar framework in navigating our application.

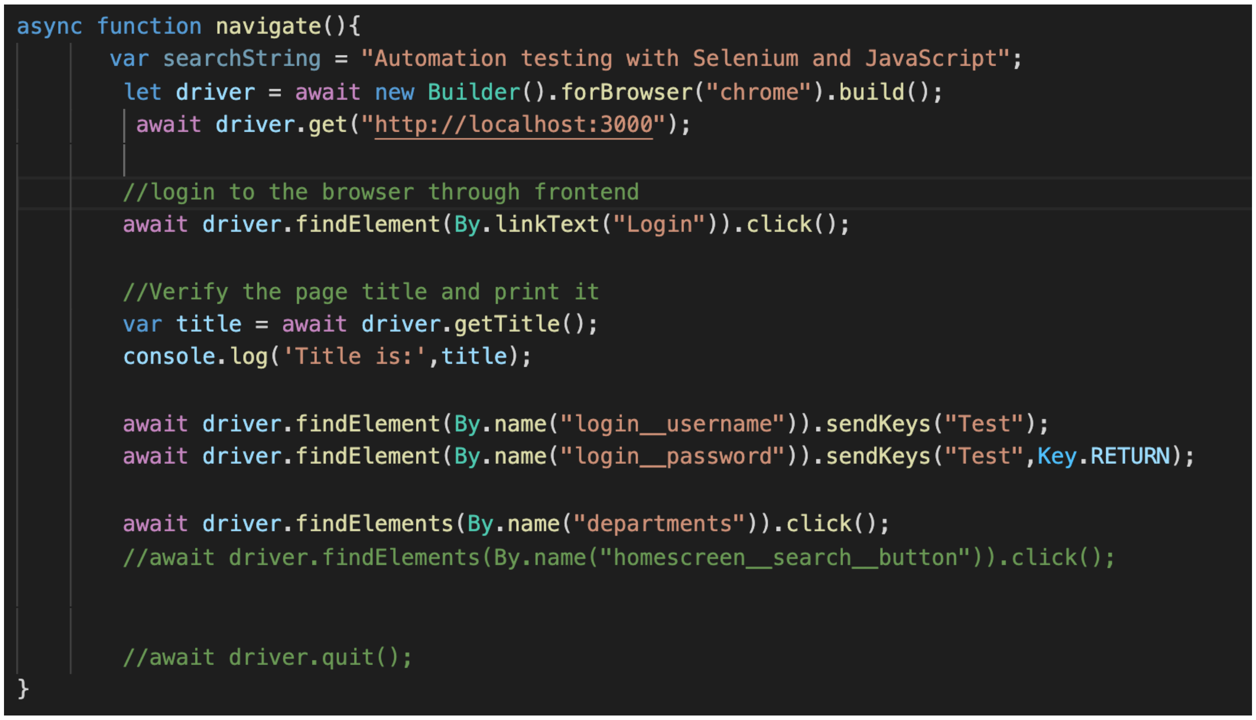
In order to test our front end we utilized the ***Selenium Webdriver*** in order to simulate user clicks and navigation on our site. Selenium only really tested the extent of effectiveness of our front end such as browsing, clicking entries and adding items to the cart. The testing of our back end and application functionality was tested in the integration testing which is demonstrated in the following section.

The unit tests for the following user profiles were combined into one large test that gauged overall UI functionality.

1) New student looking for textbooks
2) Existing University student looking to sell old supplies
3) Existing student/acct holder looking to buy used supplies.

Upon final tests, all navigation and unit tests were successful.

The function testing general front end UI interaction using Selenium is illustrated below. It is difficult to show selenium results as the code opens up the browser and executes all user interaction then closes the browser.
To perform tests to see results, *npm init* the /tests/unit_tests folder to install the packages and use the npm unit_test command/script to execute.

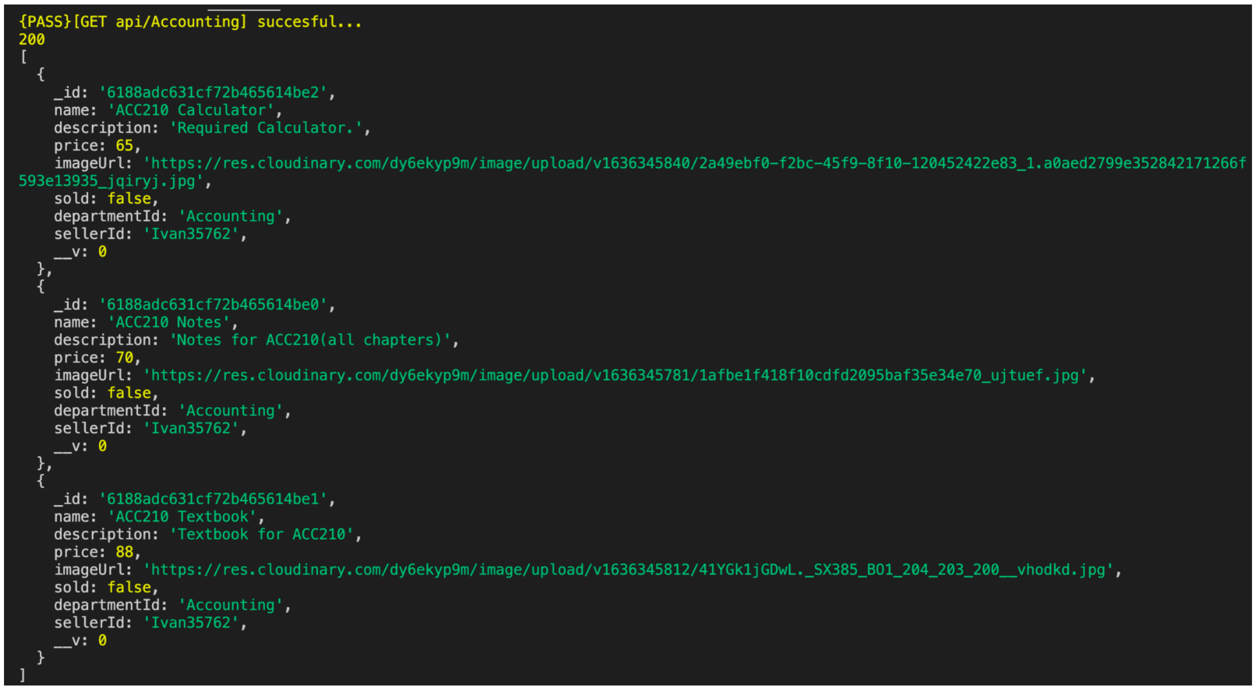


```
async function navigate(){
        var searchString = "Automation testing with Selenium and JavaScript";
         let driver = await new Builder().forBrowser("chrome").build();
         await driver.get("http://localhost:3000");

        //login to the browser through frontend
        await driver.findElement(By.linkText("Login")).click();

        //Verify the page title and print it
        var title = await driver.getTitle();
        console.log('Title is:',title);

        await driver.findElement(By.name("login__username")).sendKeys("Test");
        await driver.findElement(By.name("login__password")).sendKeys("Test",Key.RETURN);

        await driver.findElements(By.name("departments")).click();
        //await driver.findElements(By.name("homescreen__search__button")).click();


        //await driver.quit();
}
```

Integration test on next page.

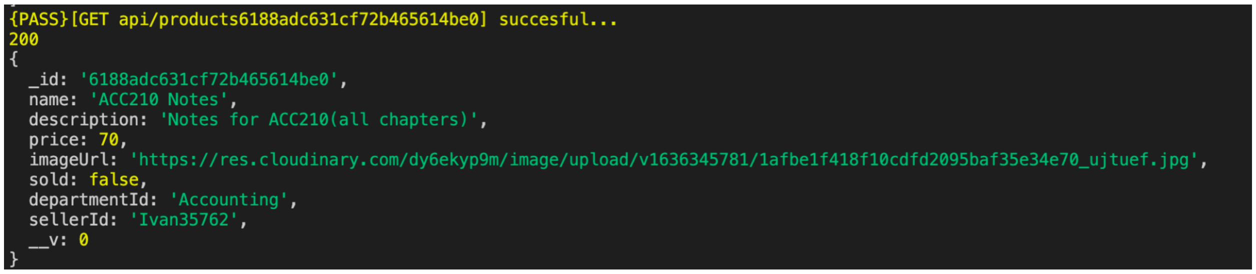
2.2 Integration Test Results **6/6 Integration Tests Passed**

Testing the retrieval of items from a particular department



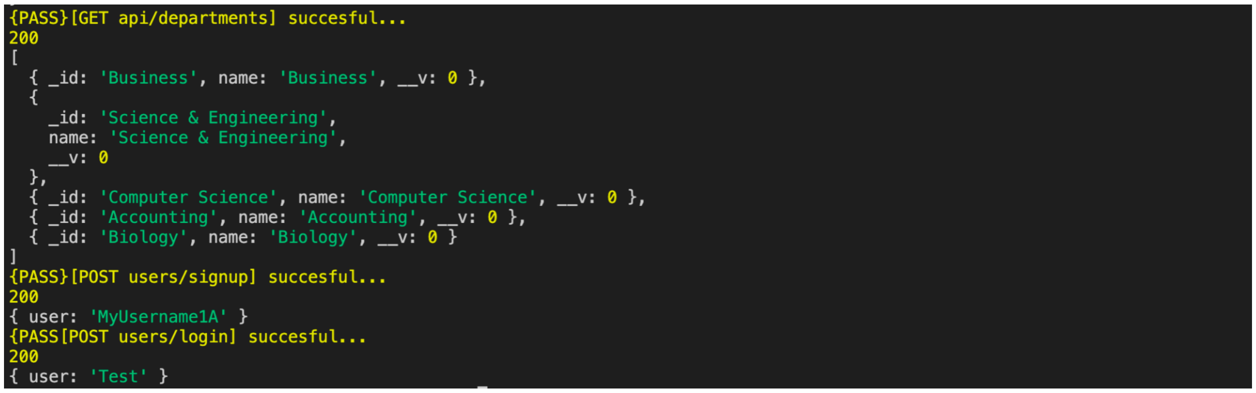
```
{PASS}[GET api/Accounting] succesful...
200
[
  {
    _id: '6188adc631cf72b465614be2',
    name: 'ACC210 Calculator',
    description: 'Required Calculator.',
    price: 65,
    imageUrl: 'https://res.cloudinary.com/dy6ekyp9m/image/upload/v1636345840/2a49ebf0-f2bc-45f9-8f10-120452422e83_1.a0aed2799e352842171266f
593e13935_jqiryj.jpg',
    sold: false,
    departmentId: 'Accounting',
    sellerId: 'Ivan35762',
    __v: 0
  },
  {
    _id: '6188adc631cf72b465614be0',
    name: 'ACC210 Notes',
    description: 'Notes for ACC210(all chapters)',
    price: 70,
    imageUrl: 'https://res.cloudinary.com/dy6ekyp9m/image/upload/v1636345781/1afbe1f418f10cdfd2095baf35e34e70_ujtuef.jpg',
    sold: false,
    departmentId: 'Accounting',
    sellerId: 'Ivan35762',
    __v: 0
  },
  {
    _id: '6188adc631cf72b465614be1',
    name: 'ACC210 Textbook',
    description: 'Textbook for ACC210',
    price: 88,
    imageUrl: 'https://res.cloudinary.com/dy6ekyp9m/image/upload/v1636345812/41YGk1jGDwL._SX385_BO1_204_203_200__vhodkd.jpg',
    sold: false,
    departmentId: 'Accounting',
    sellerId: 'Ivan35762',
    __v: 0
  }
]
```

Testing the retrieval of a specific product using PID



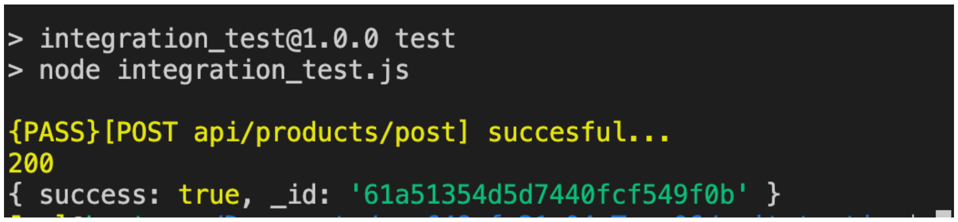
```
{PASS}[GET api/products6188adc631cf72b465614be0] succesful...
200
{
  _id: '6188adc631cf72b465614be0',
  name: 'ACC210 Notes',
  description: 'Notes for ACC210(all chapters)',
  price: 70,
  imageUrl: 'https://res.cloudinary.com/dy6ekyp9m/image/upload/v1636345781/1afbe1f418f10cdfd2095baf35e34e70_ujtuef.jpg',
  sold: false,
  departmentId: 'Accounting',
  sellerId: 'Ivan35762',
  __v: 0
}
```

Testing the retrieval of existing departments, signing up and logging in.



```
{PASS}[GET api/departments] succesful...
200
[
  { _id: 'Business', name: 'Business', __v: 0 },
  {
    _id: 'Science & Engineering',
    name: 'Science & Engineering',
    __v: 0
  },
  { _id: 'Computer Science', name: 'Computer Science', __v: 0 },
  { _id: 'Accounting', name: 'Accounting', __v: 0 },
  { _id: 'Biology', name: 'Biology', __v: 0 }
]
{PASS}[POST users/signup] succesful...
200
{ user: 'MyUsername1A' }
{PASS[POST users/login] succesful...
200
{ user: 'Test' }
```

Testing the posting of an actual listing



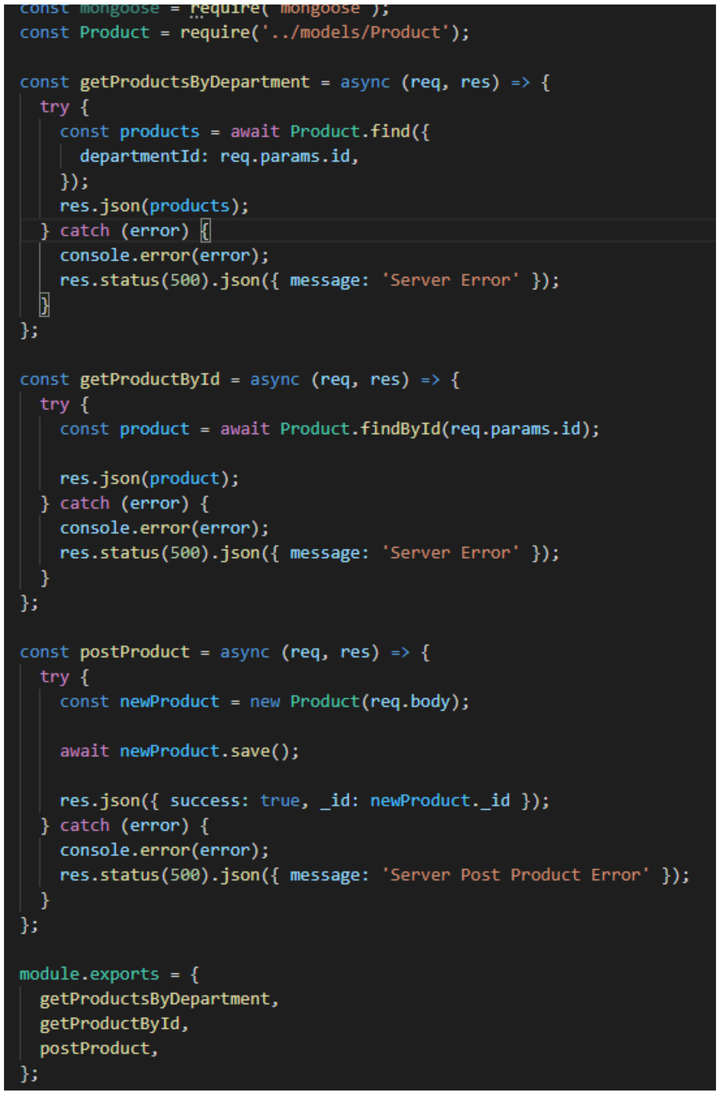
```
> integration_test@1.0.0 test
> node integration_test.js

{PASS}[POST api/products/post] succesful...
200
{ success: true, _id: '61a51354d5d7440fcf549f0b' }
```

<u>3) Code Review:</u>

A) For our coding style as a team, we were fortunate enough to all have a similar coding style already. As far as enforcing the style of coding we planned on having as a group, it was as simple as if one of the team members mistakenly did not follow the coding style rules for a part of the code, we would talk to that group member and it would be quickly fixed. However, this was a rare occurance. Our coding style included the following:

1. A space between the function name/parameter and parentheses
2. A space between parameters
3. Braces on the same line (space between was optional)
4. Indentation within the function
5. Make use of semicolons, even if not needed)

B) Here is an example of our code showing our coding style. The P1 features used for this code includes: Users being able to browse the store to see what's available and users being able to select a specific department to look at specific supplies.



```
const mongoose = require('mongoose');
const Product = require('../models/Product');

const getProductsByDepartment = async (req, res) => {
  try {
    const products = await Product.find({
      departmentId: req.params.id,
    });
    res.json(products);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server Error' });
  }
};

const getProductById = async (req, res) => {
  try {
    const product = await Product.findById(req.params.id);

    res.json(product);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server Error' });
  }
};

const postProduct = async (req, res) => {
  try {
    const newProduct = new Product(req.body);

    await newProduct.save();

    res.json({ success: true, _id: newProduct._id });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server Post Product Error' });
  }
};

module.exports = {
  getProductsByDepartment,
  getProductById,
  postProduct,
};
```

Within this portion of our code, it is seen that all of our coding style practices are being used here. To confirm this with this portion of the code and within the whole code of our program, These are the steps we'd take as a group to confirm the code:

1. Someone pushes their code
2. As a group, during our meetings, we peer review the code pushed since the last meeting and define where there are possible issues
3. After these issues are addressed, the issues are fixed before the next meeting and the finished product is pushed over to Github
4. Final reviews are done to the code to make sure everything is in order, and then the process starts again.

4) Self-check: Adherence to original Non-functional specs

1. Application shall be compatible with any web browsers and any operating systems.Done
2. The users' passwords shall be encrypted with salting and hashing. Done
3. All product and user data shall be stored in Amazon AWS server. Done
4. The code base shall be well maintained in a git repository and every code patch should get a review approval by at least one team member. Done
5. The application launching time is less than 1.001 sec.Done
6. The latency between pages should be 0.5 sec. Done