Testing Plan

By:

Camil Bouzidi (40099611)

Radley Carpio (40074888)

Bicher Chammaa (40096200)

Nimit Jaggi (40032159)

Matthew Kevork (40063824)

Cedric Martens (40086877)

William Morin-Laberge (40097269)

Adrien Tremblay (40108982)

Pierre-Olivier Trottier (40059235)

A report submitted in partial fulfillment of SOEN390.

Concordia University

March 17h, 2021

Table of Contents

1 QA Team

1.1 QA Lead

The QA lead manage processes and personnel in conducting quality assurance testing. They conduct rigorous testing and reports throughout the process of development. They also identify and report all issues to the development team.

1.2 Internal Testers

Internal testers are tasked with reviewing and analyzing system specifications. They execute test scripts and review their results. When a technical issue arises, they report and document it.

2 Testing Procedures

2.1 General Approach

2.1.1 Responsibilities of Testers

On this project, there is no formal divide between developers and testers. Developers are expected to write tests for their code, test their code before creating pull requests, as well as routinely familiarize themselves with the current state of the application to find and report bugs. Bugs should be reported as early as possible and communicated to the team via a GitHub issue using the proper bug report format (described in section 4.3). The bug reporter shall also make themselves available to the developer who eventually gets assigned the issue, to clarify and help resolve the bug if needed. Our testing coverage goal will be 50% coverage on the API controller and service methods.

2.1.2 Levels of Communication

Bugs and the results of tests can be communicated in the following ways, in increasing level of formality:

1. Conversation (video chat)
2. Message to individual
3. Group chat
4. Entry into the issue tracking system

Note that bugs must always be entered in the GitHub issues, but in some cases, such as a high priority bug, it may be appropriate to communicate the bug report using other additional methods.

2.2 Activities

2.2.1 Automated Tests

Automated tests will be incorporated into the CI/CD pipeline using GitHub Actions, and will also be run upon any new pull request being opened. The results of the test suite will be displayed to the developer. Pull requests which result in test failure should not be merged before the cause of the failure is found and resolved.

2.2.2 Focus Testing

Team members will also perform internal focus-testing regularly once a minimal viable product is achieved. This will allow rapid identification and reporting of bugs that can then be incorporated into the release schedules of subsequent sprints.

3 Generating Testing Requirements

Testing requirements are generated by this plan, as well as from the ERP Requirements document. Additionally, as new bugs are reported, new testing requirements will be generated to verify these bugs in subsequent builds.

3.1 Features to Be Tested

The following is a list of requirements to be tested:

- The system must allow defining products: sizes, colors, finishes, grades, etc.
    - To test product definitions, a unit test will be set up in which a product is created with different properties. This will determine if the unit of product creation is functioning properly. The test passes if a product is successfully defined with the desired properties. The test fails if the product is not defined, or the properties of the product do not match the expected properties.
- The system must allow creating, editing, and tracking material lists, which define what other parts are needed to assemble/create a product.
    - To test creating and editing, multiple different tests will be set up to test the user's ability to create and edit materials. The tests pass if the materials are successfully defined or edited with the desired properties. The tests fail if the materials created are not created, or the properties of the materials created or edited do not match the expected properties.
    - To test tracking material lists, tests will be written to verify that the materials created are accessible and trackable.
- The system must allow tracking inventory, which contains the numbers of things in the plants available and necessary to create parts and products.
    - Tests will verify the consistency of the materials, parts, and products inventory after simulating transactions on the system that would affect the inventory.
- The system must allow defining, ordering, and tracking raw material and final products.
    - Tests will simulate the production flow of ordering materials, manufacturing parts, and assembly of products while verifying the consistency of the data in the database of all relevant entities.
- The system must allow following all and any cost: Accounts Payable, Accounts Receivable, and Cost/Charge Number.
    - Tests will be written to simulate ordering and purchasing transactions on the system and verify that the correct financial calculations are made and written to the database.
- The system can connect to production machinery via different communication modes, including but not limited to Web services, COM ports, and CSV/XML format files.
    - Tests simulate POST requests on the API, as well as use microcontrollers to simulate production machinery COM ports.
- The system should allow only registered users to access the information, possible with different accesses for different roles.

    - Tests will simulate access attempts on guarded information, with different roles and credentials, to verify that only valid attempts gain access to the information.
- The system must generate logs and audit trails for the sake of debugging and accountability.
    - Tests will perform audit queries on the system and expect specific results to pass.
- The system can generate documents, like reports, in different formats.
    - This feature will be tested manually, before being tested automatically by verifying the existence of new report files.
- The system should follow "good" UI principles, like help, tooltips, etc.
    - This feature will be tested manually by internal focus testers.
- The system should take less than 5 seconds to load (on localhost), and no more than 10 seconds in normal network condition (on the cloud).
    - This feature will be tested manually by internal focus testers.
- Sensitive data must be encrypted.
    - To test the encryption of sensitive data, tests will be set up to encrypt some data and verify whether the data is correctly encrypted. In addition to testing encryption, decryption of the sensitive data will also be tested to verify that the encrypted data can be safely decrypted instead of lost. The tests pass if the data is successfully encrypted and decrypted without any loss of data. The tests fail if the data cannot be encrypted, decrypted, or if the data returned does not match the data provided.
- The system should support automatic logout, after a certain time of inactivity.
    - Tests will simulate the expiration of a token, upon which time any activity in the system will result in an exception being thrown, and redirection to the login screen.
- The system shall be mobile friendly web (work well on any common screen resolutions).
    - This feature will be tested manually by internal focus testers.
- The system sends e-mails to certain roles according to certain rules depending on events occurring within the system.
    - Tests will simulate the event-triggers for sending e-mails and will verify that the correct methods to send emails are called, with the correct contents and recipient.

3.2 Testing Levels

- Unit Testing: checks if software components are fulfilling their functionalities or not.
- Integration Testing: checks the data flow from one module to other modules.
- System Testing: evaluates both functional and non-functional needs for the testing.
- Acceptance Testing: checks that the requirements of a specification or contract are met as per its delivery.

3.3 Testing Types

- Regression Testing: Test suites will be run after changes or additions are made to the codebase, to ensure that no testable regression has occurred.
- Functional Testing: Tests will be written to test the system against the functional requirements outlined in section 3.1.

3.4 Testing Methods

- White-box testing: Most of the tests will be written with knowledge of the internal structure and implementation of the item being tested.
- Manual testing: In some cases, when testing usability and UI principles, manual testing will be performed by team members.

4 Bug Tracking Software

4.1 Tracking Software

Bugs will be tracked alongside other types of issues in the 'Issues' section of the GitHub repository.

4.2 Access

Access to the bug tracking list will be available to all contributors on the GitHub repository.

4.3 Bug Reporting

To report a bug, contributors will open a new issue on the GitHub repository, using the label 'bug' and with the following format:

Title: [BUG][Priority: Low/Med/High] <Descriptive Title>

Body:

Classification: What type of defect, as defined in section 5.

Summary: A clear and concise summary of the bug and undesirable symptoms observed.

Steps to Reproduce: Bullet point instructions on how to reproduce the bug. All bugs should be reproducible.

Expected Results: Short description of the expected behavior.

Actual Results: Short description of the actual, undesirable behavior.

Screenshots: If relevant, screenshots of stack traces, the application UI, and any other information best captured in images.

Extra Info: Any other relevant information to be included, such as configuration/platform/URLs.

5 Bug Classifications

5.1 Functional defects

Functional defects are errors that do not fulfill the functional requirements specified by the client.

5.2 Performance defects

Performance defects are errors that relate to the software's speed, the resources allocated, its stability, and its response time when loading content.

5.3 Usability defects

Defects that hinder the user's experience and make it difficult for the user to successfully navigate the software are called usability defects.

5.4 Compatibility defects

When the software behaves differently when used on a different browser, operating system, or device, it has a compatibility defect.

5.5 Security defects

A security defect is a defect that threatens the security of the software and can potentially allow a security attack on the software.

6 Bug Tracking

6.1 Classifying the bugs

Bugs are classified as they are discovered. The person that classifies the bug is the one that found it. This person can communicate with a peer or the team lead to classify the type of bug discovered. The bug is then reported in the issues section of the project's GitHub page along with its classification. Depending on the severity of the bug, it will either be added to the current sprint or the backlog.

6.2 Assigning the bugs

After reporting and classifying a newly discovered bug, the team lead will assign it to one of the developers based on their area of expertise and their workload. The person that discovered the bug can assign it to themselves if they know what the issue is and how to solve it.

6.3 After the bug is fixed

After the bug is fixed, the tests that failed before the bug fix are re-run. This is called re-testing or confirmation testing. If the tests fail again, the bug is not fixed and requires a new fix. New tests can also be added to check if the bug in question is fixed. Check if the bug is fixed and verify that there are not any new bugs that appear as a result of the new fix.

6.4 After the bug fix is verified

After verifying the bug fix, regression testing is applied to confirm that the new changes do not affect the previous build and that there are no new bugs that result from the change.

7 Equipment Budget and Costs

Each member of the team will be using their own computer for testing. The different software and libraries used are all Open-source or have a free license. One main software that will be used for bug tracking is GitHub.

8 Testing Frameworks

There are several different frameworks for testing both the frontend and the backend of a software application. These different options are explored to find an appropriate framework that corresponds to the testing needs of the team. A few options are described below:

a) Mocha

Mocha is a popular JavaScript testing framework that runs on NodeJS. It is useful to test both the frontend and the backend of an application. It also boasts extensive documentation that helps developers get familiar with it.

b) Jest

Jest is a JavaScript testing framework that is well establish and maintained by Facebook. Jest is easy to pick up with no prior knowledge and requires very minimal configuration before getting started. It is perfect for people that are not familiar with testing. This framework is compatible with both frontend and backend frameworks, meaning that you only need to learn how to use one framework for the entire project. It is a highly recommended framework for React based applications. The syntax and documentation are easy to use, and the testing is highly performant and fast.

After careful consideration, Jest was the chosen framework for testing the Epic Resource Planner app because it is recommended to use with React. The choice was also based on user friendliness and ability to pick up with little prior knowledge. Since the ERP is a big project and there is no dedicated testing team, it is necessary that each team member be able to test their own code efficiently. Jest also has a code coverage tool that will be very useful in determining the quality of the testing and extent of the test coverage in the app.

9 Jest Test Executions

After installing the Jest framework, tests were run to evaluate the code written. Below are examples of frontend and backend tests that were executed:

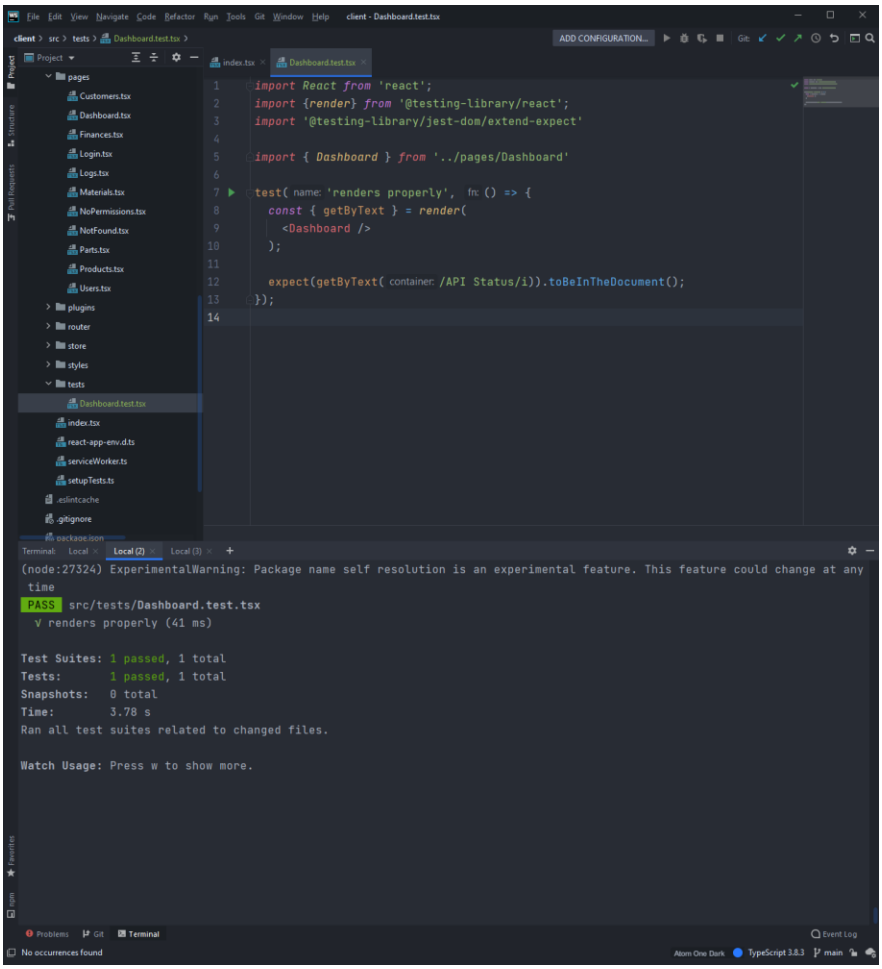


Figure 1 React test using the Jest framework

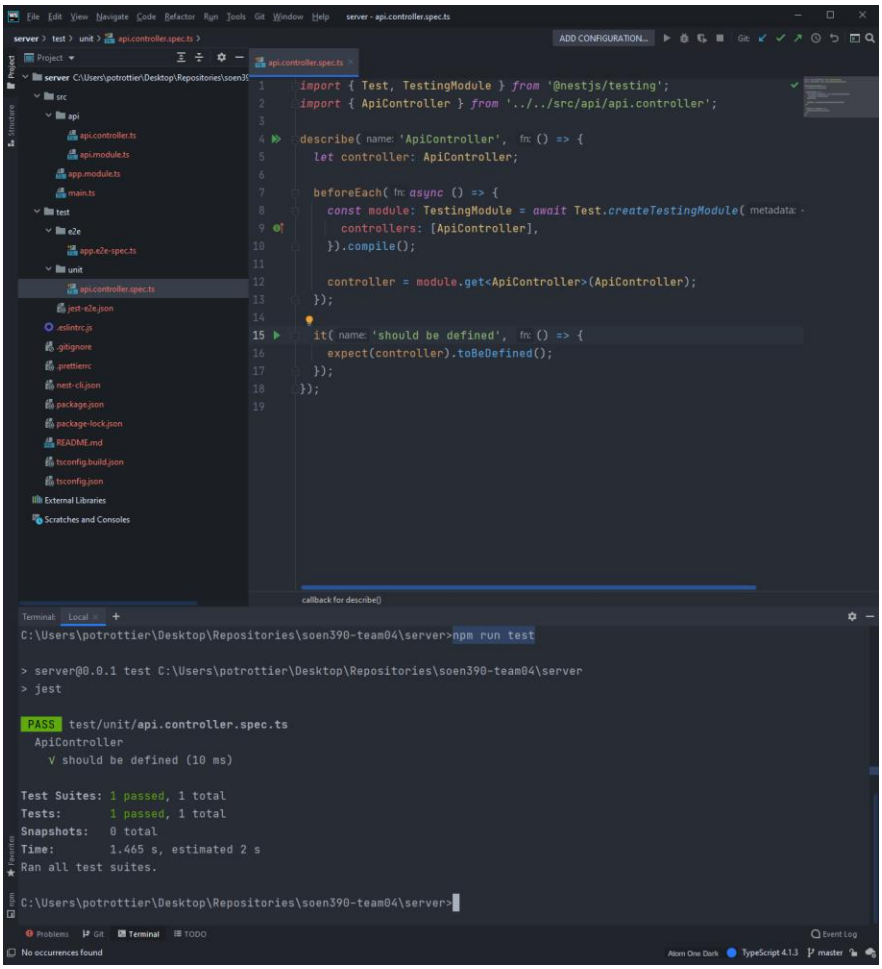
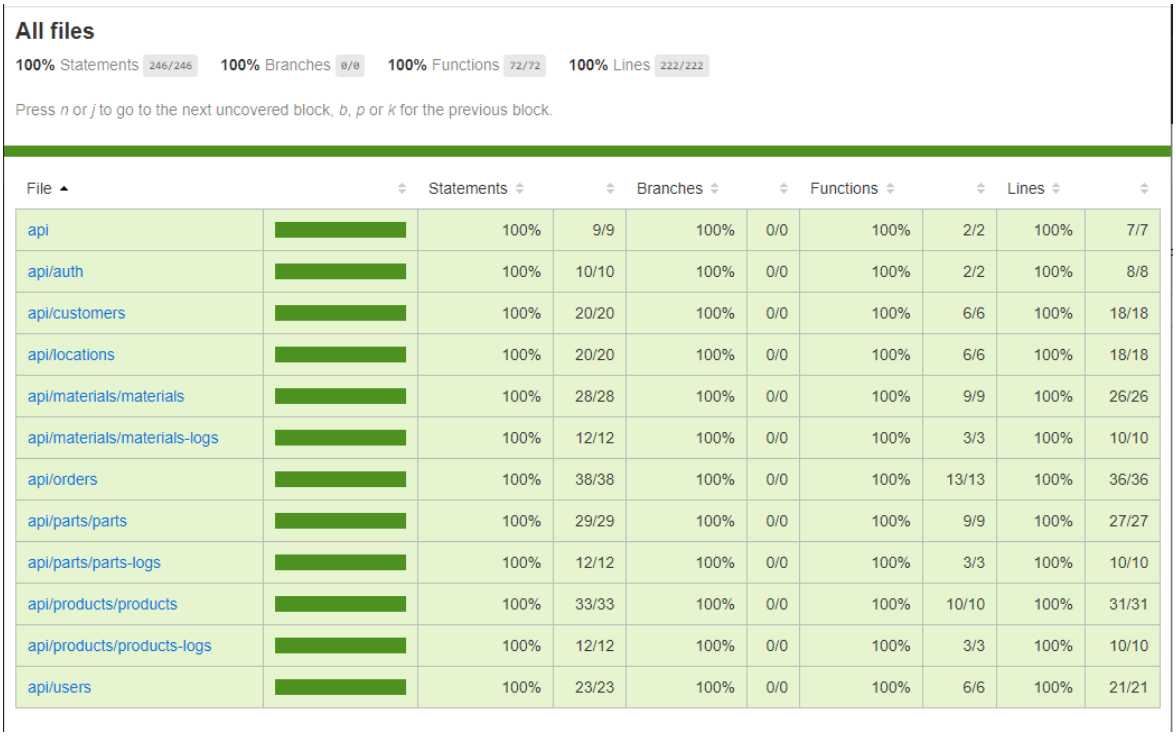


Figure 2 NestJS test using the Jest framework.

Sprint 3 Unit Tests

This section includes the unit tests performed up to and including sprint 3. The tests are focused on the server side of the application and they test the controller classes and service classes. Below is a breakdown of all the test folders with their pass status:



All files

100% Statements 246/246 **100%** Branches 0/0 **100%** Functions 72/72 **100%** Lines 222/222

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
api	100%	9/9	100%	0/0	100%	2/2	100%	7/7
api/auth	100%	10/10	100%	0/0	100%	2/2	100%	8/8
api/customers	100%	20/20	100%	0/0	100%	6/6	100%	18/18
api/locations	100%	20/20	100%	0/0	100%	6/6	100%	18/18
api/materials/materials	100%	28/28	100%	0/0	100%	9/9	100%	26/26
api/materials/materials-logs	100%	12/12	100%	0/0	100%	3/3	100%	10/10
api/orders	100%	38/38	100%	0/0	100%	13/13	100%	36/36
api/parts/parts	100%	29/29	100%	0/0	100%	9/9	100%	27/27
api/parts/parts-logs	100%	12/12	100%	0/0	100%	3/3	100%	10/10
api/products/products	100%	33/33	100%	0/0	100%	10/10	100%	31/31
api/products/products-logs	100%	12/12	100%	0/0	100%	3/3	100%	10/10
api/users	100%	23/23	100%	0/0	100%	6/6	100%	21/21

Figure 3 Breakdown of code coverage for test folders

In the above screenshot, the test coverage shown is of 100%. All tests pass and they cover all statements, branches, functions, and lines.

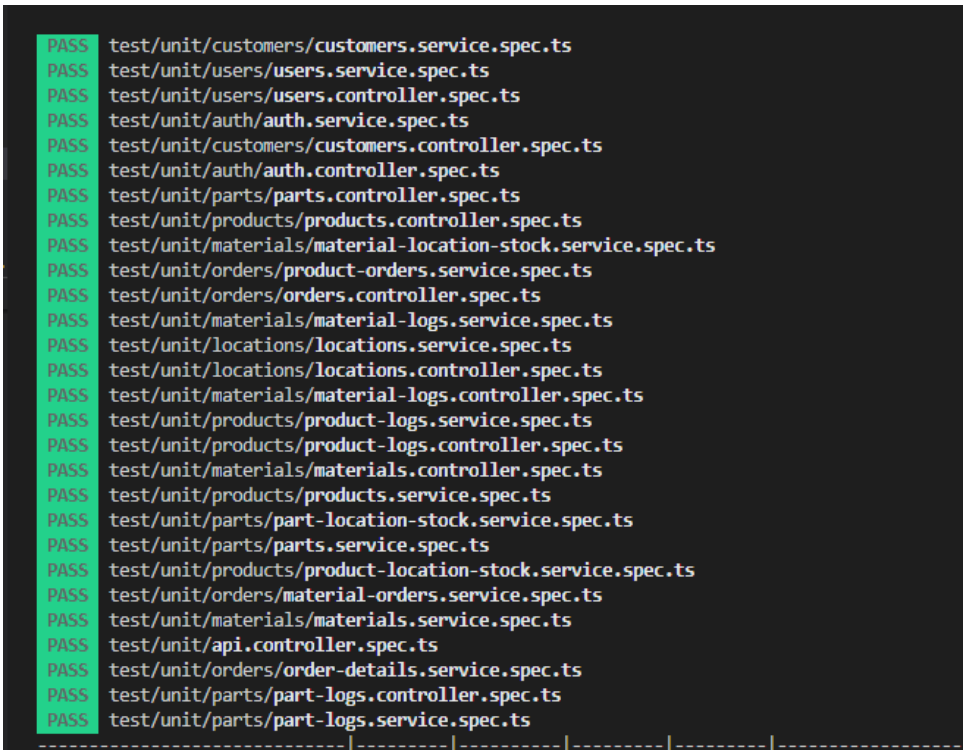


Figure 4 Unit test file paths

The screenshot above provides more details on the path of each test file and the pass state of each individual file. All test files are located at ***soen390-team04/server/test/unit/***. Each testcase has a description before it that helps better understand what is being tested.

Attached in the same folder is the html report for the code coverage. The report contains more information on the coverage and the tested code can be accessed from there by selecting the proper folder from the index.html file.