

SOEN390 - Software Engineering Team Design Project
Team 6 - Deliverable 2

Testing Plan and Report

Winter 2024

Done by:

Hoang Minh Khoi Pham 40162551
Michaël Gugliandolo 40213419
Jessey Thach 40210440
Mahanaim Rubin Yo 40178119
Vanessa DiPietrantonio 40189938
Ahmad Elmahallawy 40193418
Clara Gagnon 40208598
Khanh Huy Nguyen 40125396
Jean-Nicolas Sabatini-Ouellet 40207926
Mohamad Mounir Yassin 40198854

Professor Junqiu Yang
Department of Computer Science and Software Engineering
Gina Cody School of Engineering and Computer Science

Concordia University

Table of Content

1. TESTING TOOL	3
1.1 Platform	3
1.1.1 Github	3
1.1.2 Sonarqube	3
1.2 Testing Module	3
2. APPROACH	3
2.1 Test Planning and Design	4
2.2 Unit Test	4
2.3 Integration Test	4
2.4 End-to-End (E2E)	4
3. METRICS	5
3.1 Reliability	5
3.2 Maintainability	6
3.3 Security	6
3.4 Coverage	7
3.5 Duplication	7
3.6 Size	8
3.7 Complexity	9
3.8 Issues	10
4. COVERAGE	11
5. ACCEPTANCE TEST	17
6. TEST RESULT	21

This test approach document describes the appropriate strategies, process, workflows and methodologies used to plan, organize, execute and manage testing of Condo Management System.

1. TESTING TOOL

As a part of the developing cycle, testing plays a key role in maintainability and reliability for existing software, including the Condo Management System. In 2 parts, backend and frontend, tests will be conducted under types of unit tests, integration test and system test.

1.1 Platform

2 platforms that we chose to implement for this project: Github platform and Sonarqube

1.1.1 Github

We implemented 2 workflows as GitHub actions for CI in our repository, one for dev branch and one for main branch. The CI GitHub action is responsible for building and deploying the frontend and backend, and for monitoring the status of the deployment on the server, not to mention the compatibility of the node_modules implemented throughout the project.

1.1.2 Sonarqube

We implemented Sonarqube to validate the code quality and maintainability. Not only it's a powerful static code analysis tool that automatically reviews codes for bugs, vulnerabilities, code smells and quality issues, but also it can indicate the latent issues early on, resulting in faster resolution and reducing risks in development.

1.2 Testing Module

Jest is a testing framework for unit and integration tests with the benefits of being simple, speedy and built-in features (mocks and snapshots). In the frontend, we implemented Jest with a configuration for Typescript, on the other hand, the backend is implemented with JavaScript. Jest, with its functions, can provide a quick test report with insightful data, telling the % coverage of statements, branches, functions and lines.

2. APPROACH

During the development process, the systematic approach is adopted to use Jest for testing. The process includes several important steps to ensure our tests' efficiency and the imminent resolution of issues may encounter:

2.1 Test Planning and Design

Supposedly, before any implementation, test cases and scenarios of each functional requirement, user story or acceptance criterion will be conveyed. All possible cases of functionality (positive, negative, edge, error) will have to be covered by test cases.

2.2 Unit Test

Unit testing evaluates individual components or "units" of software independently. It is a part of white box testing, an approach that examines the internal modules of the code. These units are the smallest testable pieces in the software. Unit testing is an essential component of software development since it ensures that each unit is executed correctly and detects any flaws early on. Following the TDD approach, each component, function or module will be put as top priority to write unit tests. After each implementation or code change, we execute the unit test to ensure the maintainability and quality of the updated codes. Notwithstanding, it results in high chances of catching regression earlier in the process and ensuring the new code still meets the expected behavior. A sample of a JEST unit testing on the Frontend side is shown below.

```
test("initially renders default request description", () => {  
  render(<Requests />);  
  const defaultDescription = screen.getByText(/I hope you're well/i);  
  expect(defaultDescription).toBeInTheDocument();  
});
```

Figure 1: Unit Testing Sample with JEST

2.3 Integration Test

Integration testing aims to evaluate the collective performance of various system components by examining their interactions rather than scrutinizing the entire system as a single entity. The focus is on verifying if these components communicate effectively. For instance, when component A transmits message X to component B, integration testing assesses whether their interactions result in the expected behaviors and functionalities. For instance, the landing page for the user profile is composed of several components. Testing these components separately would be unit testing, but testing them together and how they interact is integration testing. Verification between frontend and backend components, API endpoints and data flow will be ensured by integration tests. We will be writing integration tests to ensure the quality of our code.

2.4 End-to-End (E2E)

Condo Management website's workflow from frontend UI and backend services will be tested by E2E with real user interaction, stimulated by Cypress UI. The reason we decided to go with Cypress is because it provides an intuitive UI which makes it easier to learn and use. Furthermore, Cypress provides a powerful debugging tool, developers can easily inspect the DOM element if a test assertion fails and find out what went wrong. Supposedly, E2E is less

frequently run compared to unit and integration tests, only if there are some major deployments such as pushing from dev to main. Broken user flows and critical bugs are more likely to be seen in this test as it provides confidence in overall functionality.

3. METRICS

With SonarQube it is easy to keep track of our quality metrics. The metrics are separated in two distinct sections. The first section is overview metrics, which combine other metrics to give an overall perspective of the code quality. As such, we have selected reliability, maintainability and issue metrics. The second section is singular metrics, which are security, test coverage, code duplication, code size, code complexity and issues.

3.1 Reliability

SonarQube assesses the reliability of code by identifying potential bugs, code smells, and other issues that could lead to unexpected behavior or system failures. By analyzing reliability metrics, such as the number of bugs detected and their severity, SonarQube helps ensure that the codebase is robust and dependable. Furthermore, the use of tests ensures that the functionalities work consistently according to the requirements. Finally, as part of our core development structure, each function also goes through manual testing in our dev environment. This acts as a final safety net to ensure that the code does what it is supposed to do consistently.

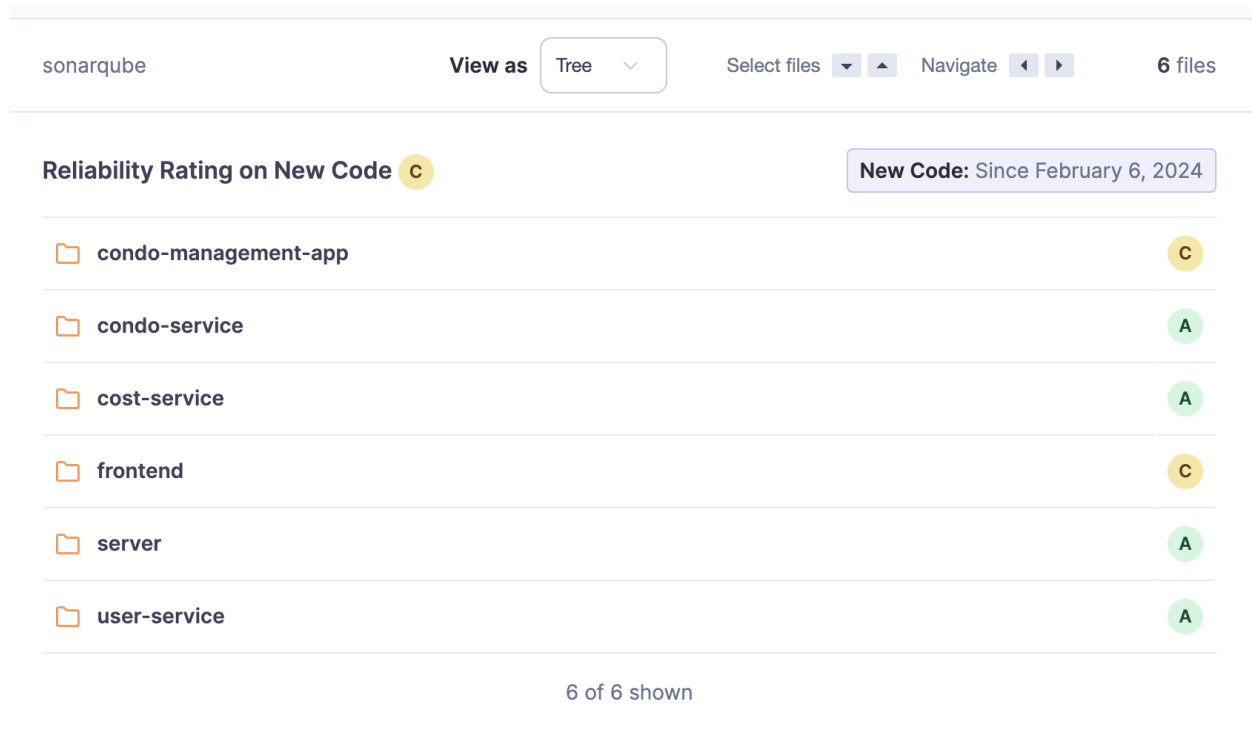


Figure 2: Reliability Report on Sprint 2

3.2 Maintainability

SonarQube assesses code maintainability based on readability, complexity, and adherence to coding standards and best practices. Improving maintainability measures such as code duplication, cyclomatic complexity, and code coverage simplifies future maintenance, refactoring, and code review procedures.

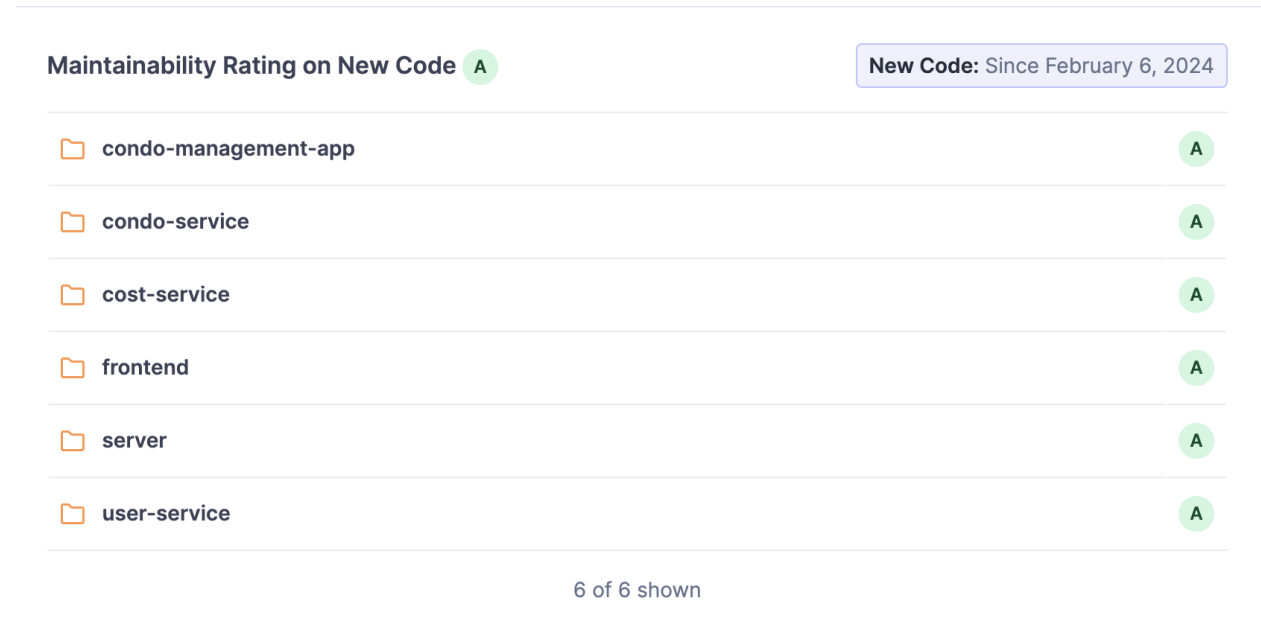


Figure 3: Maintainability report for sprint 2

3.3 Security

SonarQube contains security-focused rules and tests that help uncover vulnerabilities, security hotspots, and possible security hazards in code. In addition, SonarQube reduces security risks and strengthens the application's overall security posture by assessing security metrics such as the number and severity of security vulnerabilities.

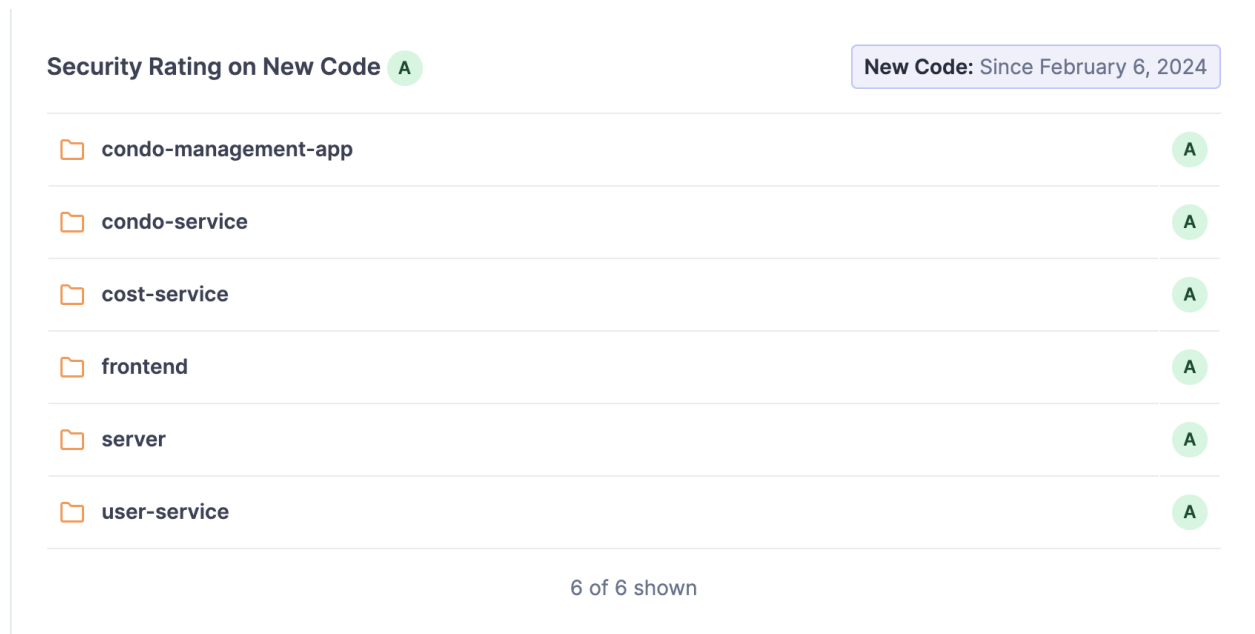


Figure 4: Security Report on Sprint 2

3.4 Coverage







SonarQube assesses code coverage by calculating the proportion of code covered by automated tests, such as unit tests, integration tests, and end-to-end tests. boosting code coverage measures assures that the codebase has been properly tested, lowering the possibility of undetected bugs and boosting overall product quality.

3.5 Duplication

Duplication is a metric that needs to be monitored correctly since multiple people work on the same project. For example, two developers work on two different features that have similarities. If there are no checks in place, the code can get bloated, and hard to read and maintain. Thus, SonarQube is used to help us reduce duplication by finding repeating code patterns or similar code parts throughout the source. Addressing code duplication improves maintainability, readability, and consistency, resulting in more efficient development and simpler issue fixes.

Duplicated Lines (%) 16.8% [See history](#)

New Code: Since February 6, 2024

	Duplicated Lines (%)	Duplicated Lines
 condo-management-app	0.0%	0
 condo-service	86.5%	83
 cost-service	86.5%	83
 frontend	12.3%	1,126
 server	20.2%	4,144
 user-service	11.4%	237

6 of 6 shown

Figure 5: Duplication Report for sprint 2

3.6 Size

SonarQube gives information on the size of the codebase, including the number of files, lines of code, and code churn over time. Monitoring code size metrics allows us to track project growth, identify areas of code bloat or excessive complexity, and improve resource allocation and project planning.

Lines of Code 27,333 [See history](#)

New Code: Since February 6, 2024

TypeScript 22k
JavaScript 3.2k
CSS 1.9k
HTML 197
Docker 57

condo-management-app	1,661
condo-service	77
cost-service	77
frontend	7,604
server	16,327
user-service	1,587

6 of 6 shown

Figure 6: Line Of Code Report for Sprint 2

3.7 Complexity

SonarQube evaluates code complexity by measures such as cyclomatic complexity, nesting depth, and method length. Analyzing complexity metrics aids in identifying unnecessarily complicated code segments, potential failure spots, and places for rewriting or optimization to enhance code quality and maintenance.

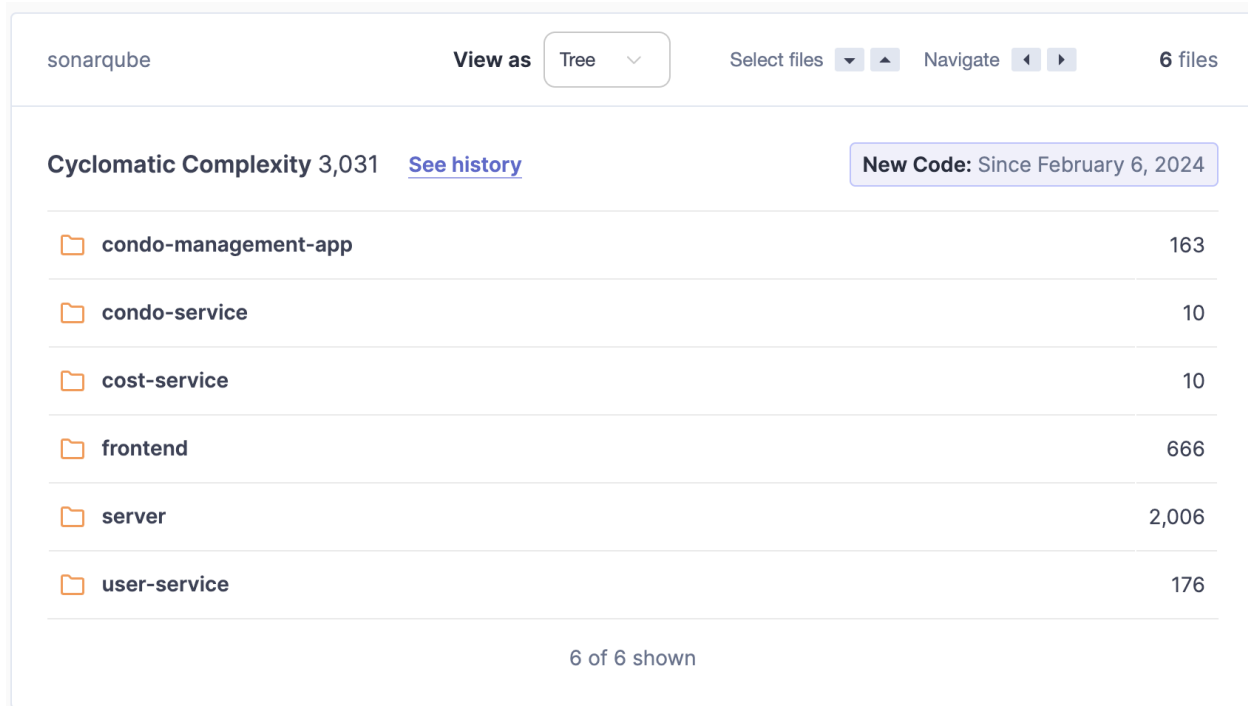


Figure 7: Cyclomatic Complexity Report for sprint 2

3.8 Issues

SonarQube keeps track of issues found in the code, including but not limited to: code smells, potential bugs, bad practices, unnecessary hard coding etc...

This process is called linting and is a good tool to ensure consistent good practices across developers. As a consequence, issue tracking can increase the maintainability, readability, complexity and other software qualities. The number of issues should be kept as low as possible

New Issues 405

New Code: Since February 6, 2024

condo-management-app	122
condo-service	1
cost-service	1
frontend	87
server	186
user-service	8

6 of 6 shown

Figure 8: Issue Report for sprint 2

4. COVERAGE

Software testing involves various metrics to assess the effectiveness of test cases in exercising the code. Four commonly used metrics are statement coverage, branch coverage, function coverage, and line coverage.

Statement coverage

measures the percentage of executable statements that have been executed at least once by the test cases. This metric provides insight into how well individual statements in the code have been exercised during testing.

Branch coverage

focuses on decision points in the code, such as if statements or loops. It assesses the percentage of decision branches that have been executed at least once, giving a more granular view of how well the code paths have been tested.

Function coverage

evaluates the percentage of functions or methods that have been executed by the test cases. This metric helps ensure that the different functional components of the software are adequately tested.

Line coverage

measures the percentage of lines of code that have been executed at least once. It provides a comprehensive overview of how well the entire codebase has been tested in terms of individual lines of code.

As asked in the requirements. We were able to pass 80% code coverage in both sprint 1 and 2.

On the frontend side, the statement coverage overall was 88.3%, the branch coverage was 81.67%, function coverage was 80.89% and lines were at 87.69% (as shown in the figure below). Therefore we succeeded at achieving over 80% for everything. We had 35 test suites (22 additional test cases from sprint 1) and we had 150 test cases (101 additional test cases from sprint 1) which all passed without any failure. These numbers are significant compared to sprint 1 and we managed to increase the branch coverage by around 1% which was hard to achieve. Although the other test metrics fell by 2-3%, we were still above 80%.

On the backend side, the statement coverage overall was 89.11%, the branch coverage was 81.98%, function coverage was 93.38% and lines were at 87.88%. We were able to achieve over 80% coverage for all of these criterias. In total, there were 19 test suites and 87 tests. All of the tests passed without exception. Compared to sprint 1, sprint 2's statement coverage decreased by around 6% which is due to the abundant lines of code added in this sprint for additional functionalities. Our branch coverage has improved by 1%, while function and line coverage decreased by 2% and 7 % respectively. That being said, all our test metrics exceeds the 80% mark despite the additional functionalities added in sprint 2.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.3	81.67	80.89	87.69	
src	97.29	80	83.33	97.05	
App.tsx	100	100	100	100	
api.ts	93.33	80	80	91.66	21
urls.ts	100	100	100	100	
src/Components	96.66	85.71	80	100	
NavBar.tsx	96.66	85.71	80	100	45,82
src/Components/Authentication	93.33	83.95	100	93.47	
EmployeeRegistration.tsx	100	71.92	100	100	33-216
LogIn.tsx	85.71	86.2	100	86.11	57-68,80
LogOut.tsx	100	100	100	100	
ResetPassword.tsx	100	96.42	100	100	125
SignUp.tsx	91.07	87.67	100	91.3	118-123
src/Components/Common	100	71.42	100	100	
AuthUtil.tsx	100	100	100	100	
InitialValues.ts	100	100	100	100	
List.tsx	100	66.66	100	100	17-20
LoadingScreen.tsx	100	100	100	100	
ValidationSchema.ts	100	100	100	100	
src/Components/CompanyDashboard	100	100	100	100	
IconCard.tsx	100	100	100	100	
src/Components/Condo	100	100	100	100	
Condo.tsx	100	100	100	100	
src/Components/CondoProfile	90.24	91.17	90.9	88.73	
CondoCreation.tsx	82.05	87.5	100	80.55	25-35,62-63
CondoInfoField.tsx	100	100	100	100	
CondoInfoForm.tsx	97.05	100	85.71	96.15	50
src/Components/Property	78.94	75	61.11	79.41	
CreateProperty.tsx	74.6	75	56.25	75	57,68-76,88,108-109,113-114,118-123,197
Property.tsx	100	100	100	100	
src/Components/PropertyProfile	81.03	90.47	65.51	79.79	
CondoFilesModal.tsx	73.43	81.81	58.82	72.72	54-63,89-115,170-199
PropertyInfoField.tsx	100	100	100	100	
PropertyInfoForm.tsx	87.5	100	66.66	84.37	49-50,54-55,110
src/Components/Requests	97.77	92.3	100	97.43	
KeyGeneration.tsx	97.77	92.3	100	97.43	96
src/Components/UserProfile	88	70.73	85.71	85.71	
UserInfoFields.tsx	100	100	100	100	
UserInformation.tsx	78.46	52	75	74	44,52-53,63-65,72-73,95-104
UserKeyRegister.tsx	95	100	75	93.33	38
UserProperties.tsx	100	100	100	100	
UserRequests.tsx	100	100	100	100	
src/Pages	82.18	67.85	76.92	81.69	
CompanyDashboardLandingPage.tsx	93.75	66.66	100	93.75	20
CondoCreationLandingPage.tsx	100	100	100	100	
CondoProfileLandingPage.tsx	100	100	100	100	
CreatePropertyLandingPage.tsx	100	100	100	100	
EmployeeRegistrationLandingPage.tsx	100	100	100	100	
EmployeeRequestsLandingPage.tsx	100	100	100	100	
Hero.tsx	100	100	100	100	
LogInLandingPage.tsx	100	100	100	100	
PropertyListLandingPage.tsx	96.66	75	100	96	50
PropertyProfileLandingPage.tsx	58.46	57.14	40	52.94	52-53,60-87,95-127,131-132,157-161
RegistrationLandingPage.tsx	100	100	100	100	
UserProfileLandingPage.tsx	84.61	100	100	84.61	19-21
Test Suites: 35 passed, 35 total					
Tests: 150 passed, 150 total					

Figure 9: Sample Frontend Percentage Coverage on the Terminal

One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.

For the future, we plan to maintain the overall code coverage of at least 80%. This will reduce the number of undetected bugs and will give us more confidence in our application. Code coverage enforcement can be done by configuring JEST to fail every time coverage drops below 80% which will prevent the PR from being merged.

Sprint 3 Testing Plan for Code Coverage

Our plan for Sprint 3 is to achieve over 90% coverage in statement as well as line coverage. For Branch and Function coverage, since these are the hardest, we aim to achieve 85%. We will continue to test every component we work on and enforce each team member to include tests in the Pull Requests to ensure that we will always be above 80% coverage in all metrics. We will continue to use JEST as a testing library as it is effective and detailed.

I

Frontend Reports:

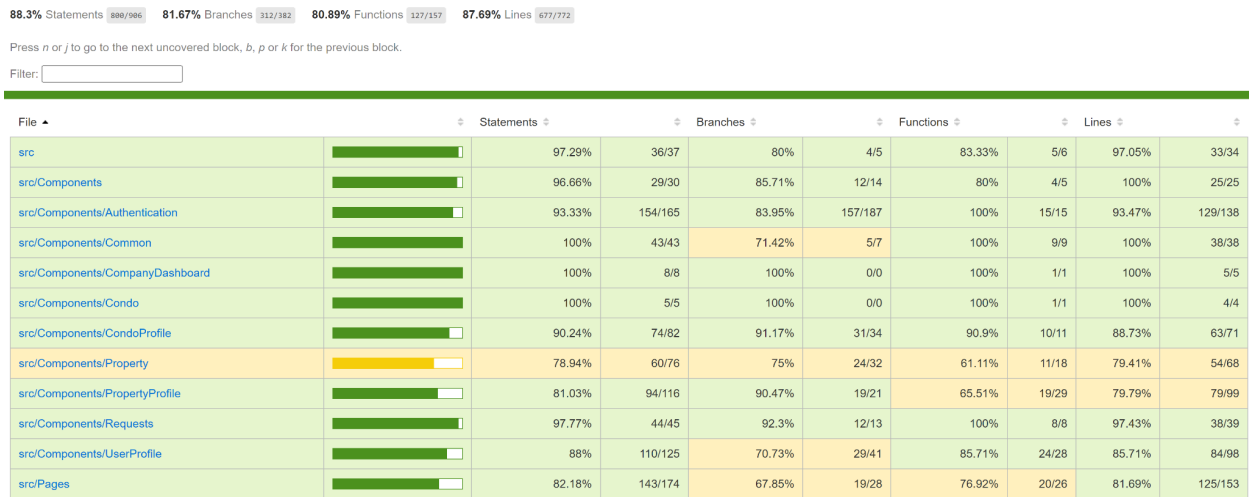


Figure 10: Main Page Report with JEST

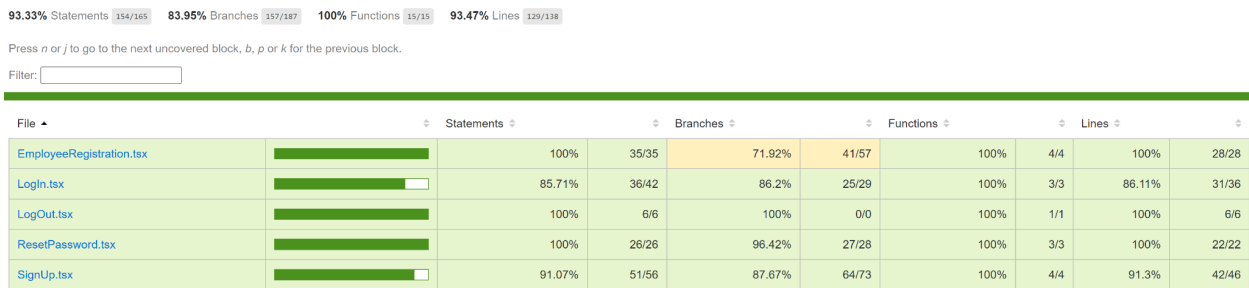


Figure 11: Detailed Statistics of the Authentication Folder and its Components

100% Statements 43/43 71.42% Branches 5/7 100% Functions 9/9 100% Lines 38/38

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

Filter:

File		Statements	Branches	Functions	Lines
AuthUtil.tsx	<div><div></div></div>	100%	7/7	100%	1/1
InitialValues.ts	<div><div></div></div>	100%	5/5	100%	0/0
List.tsx	<div><div></div></div>	100%	20/20	66.66%	4/6
LoadingScreen.tsx	<div><div></div></div>	100%	5/5	100%	0/0
ValidationSchema.ts	<div><div></div></div>	100%	6/6	100%	0/0

Figure 12: Detailed Statistics of the Common Folder with Reusable files

File		Statements	Branches	Functions	Lines
UserInfoFields.tsx	<div><div></div></div>	100%	13/13	100%	12/12
UserInformation.tsx	<div><div></div></div>	78.46%	51/65	52%	13/25
UserKeyRegister.tsx	<div><div></div></div>	95%	19/20	100%	0/0
UserProperties.tsx	<div><div></div></div>	100%	18/18	100%	4/4
UserRequests.tsx	<div><div></div></div>	100%	9/9	100%	0/0

Figure 13: Detailed Statistics of the UserProfile

File		Statements	Branches	Functions	Lines
CompanyDashboardLandingPage.tsx	<div><div></div></div>	93.75%	15/16	66.66%	2/3
CondoCreationLandingPage.tsx	<div><div></div></div>	100%	5/5	100%	0/0
CondoProfileLandingPage.tsx	<div><div></div></div>	100%	17/17	100%	2/2
CreatePropertyLandingPage.tsx	<div><div></div></div>	100%	5/5	100%	0/0
EmployeeRegistrationLandingPage.tsx	<div><div></div></div>	100%	4/4	100%	0/0
EmployeeRequestsLandingPage.tsx	<div><div></div></div>	100%	5/5	100%	0/0
Hero.tsx	<div><div></div></div>	100%	4/4	100%	0/0
LoginLandingPage.tsx	<div><div></div></div>	100%	5/5	100%	0/0
PropertyListLandingPage.tsx	<div><div></div></div>	96.66%	29/30	75%	6/8
PropertyProfileLandingPage.tsx	<div><div></div></div>	58.46%	38/65	57.14%	8/14
RegistrationLandingPage.tsx	<div><div></div></div>	100%	5/5	100%	0/0
UserProfileLandingPage.tsx	<div><div></div></div>	84.61%	11/13	100%	1/1

Figure 14: Detailed Statistics of the Pages Folder with Landing Pages

97.29% Statements 36/37 80% Branches 4/5 83.33% Functions 5/6 97.05% Lines 33/34

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

Filter:

File		Statements	Branches	Functions	Lines
App.tsx	<div><div></div></div>	100%	18/18	100%	0/0
api.ts	<div><div></div></div>	93.33%	14/15	80%	4/5
urls.ts	<div><div></div></div>	100%	4/4	100%	0/0

Figure 15: Detailed Statistics of the Src Folder(we tested the app itself)

90.24% Statements 74/82 91.17% Branches 31/34 90.9% Functions 16/11 88.73% Lines 53/71

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

Filter:

File		Statements	Branches	Functions	Lines	
CondoCreation.tsx	<div><div></div></div>	82.05%	32/39	87.5%	21/24	100%
CondoInfoField.tsx	<div><div></div></div>	100%	9/9	100%	4/4	100%
CondoInfoForm.tsx	<div><div></div></div>	97.05%	33/34	100%	6/6	85.71%

Figure 16: Detailed Statistics of the CondoProfile

File		Statements	Branches	Functions	Lines	
CreateProperty.tsx	<div><div></div></div>	74.6%	47/63	75%	24/32	56.25%
Property.tsx	<div><div></div></div>	100%	13/13	100%	0/0	100%

Figure 17: Detailed Statistics of the Property

File		Statements	Branches	Functions	Lines	
CondoFilesModal.tsx	<div><div></div></div>	73.43%	47/64	81.81%	9/11	58.82%
PropertyInfoField.tsx	<div><div></div></div>	100%	12/12	100%	2/2	100%
PropertyInfoForm.tsx	<div><div></div></div>	87.5%	35/40	100%	8/8	66.66%

Figure 18: Detailed Statistics of the PropertyProfile

The rest of the folders with no details only have 1 file, so it is already represented in the “All Files” section.

Backend Reports:

All files						
89.11% Statements 638/707 81.98% Branches 95/115 93.38% Functions 133/143 87.88% Lines 562/640						
Press n or j to go to the next uncovered block, b, p or k for the previous block.						
Filter: <input type="text"/>						
File		Statements	Branches	Functions	Lines	
company/base	<div><div></div></div>	90.21%	83/92	90.66%	5/6	83.33%
companyEmployee/base	<div><div></div></div>	95.45%	42/44	85.45%	9/11	100%
condoUnit/base	<div><div></div></div>	85.71%	78/91	80.88%	17/18	88.88%
file/base	<div><div></div></div>	82.14%	53/56	86.31%	17/19	87.76%
locker/base	<div><div></div></div>	86.74%	38/43	87.27%	11/11	100%
parkingSpot/base	<div><div></div></div>	86.74%	40/43	87.27%	11/11	100%
property/base	<div><div></div></div>	88.78%	95/107	96.27%	18/11	87.27%
registrationKey/base	<div><div></div></div>	90.69%	39/43	100%	3/3	100%
role/base	<div><div></div></div>	86.74%	39/43	100%	3/3	100%
user/base	<div><div></div></div>	80.26%	49/62	16.66%	5/6	88.88%
userCondo/base	<div><div></div></div>	86.74%	38/43	92.2%	9/11	100%

Figure 19: Main Page Report of the Backend with JEST

File		Statements	Branches	Functions	Lines	
company/base	<div><div></div></div>	90.21%	83/92	90.66%	5/6	83.33%
companyEmployee/base	<div><div></div></div>	95.45%	42/44	85.45%	9/11	100%

Figure 20: Detailed Statistics of company

File	Statements	Branches	Functions	Lines
condoUnit/base	<div><div></div></div> 85.71%	78/91 80.88%	17/18 88.88%	16/18 85.39%
userCondo/base	<div><div></div></div> 86.74%	38/43 92.2%	9/11 100%	6/6 100%

Figure 21: Detailed Statistics of condo

File	Statements	Branches	Functions	Lines
user/base	<div><div></div></div> 80.26%	49/92 16.66%	5/6 88.88%	18/18 100%
userCondo/base	<div><div></div></div> 86.74%	38/43 92.2%	9/11 100%	6/6 100%

Figure 22: Detailed Statistics of User

The rest of the folders with no details only have 1 file, so it is already represented in the “All Files” section.

5.ACCEPTANCE TEST

Sprint 3 Acceptance Tests

Test ID	AT-1
Requirement	As a condo owner, I want to have a dashboard of properties (general information, personal profile, condo information, financial status, remaining balance of condo fee payments, status of submitted request)
Acceptance Criteria	User is logged into their account.
Test Case	Expected Results
TC-1 User clicks on “My Profile” from the dashboard page.	The user navigates to their personal profile.
TC-2 User clicks on “My Condos” from the dashboard page.	The user navigates to the list of condos under their ownership.
TC-3 User clicks on “My Finances” from the dashboard page.	The user navigates to the financial status and payments section of their account.

TC-4 User clicks on “My Requests” from the dashboard page.	The user navigates to the list of their submitted requests.
---	---

Test ID	AT-2
Requirement Tested	As a condo owner, I want to be able to submit requests.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to compose a new request.
Test Case:	Expected Results
TC-1 Condo owner selects a request type and submits a request.	Request is successfully submitted, and the details are stored in the system. Assigned personnel receive notification of the new request for it to be addressed promptly.
TC-2 Condo owner does not select a request type and submits a request.	The request is not successfully submitted and the user is notified to select a request type from the dropdown for a valid submission.

Test ID	AT-3
Requirement Tested	As a condo management employee, I want to be able to see the assigned requests.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to display all assigned requests.
Test Case:	Expected Results
TC-1 Employee accesses the assigned requests page	The system displays a list of all requests assigned to them by type with relevant details available.
TC-2 Employee updates the status of a request	The request status is successfully updated in the system and notifications are sent to the corresponding condo owner.

Test ID	AT-4
---------	------

Requirement Tested	Users should have a notification page.
Acceptance Criteria	Users can access a notification page from the navigation bar. The notification page should display a list of recent activities related to submitted or assigned requests.
Test Case:	Expected Results
TC-1 Condo owner navigates to the notification page.	The system displays a list of recent activities related to their submitted requests with updated status.
TC-2 Condo employee navigates to the notification page.	The system displays a list of recent activities related to assigned requests.
TC-3 Manager navigates to the notification page.	The system displays a list of recent activities related to assigned requests.

Test ID	AT-5
Requirement Tested	Condo management companies should be able to set up different roles for employees
Acceptance Criteria	The user is logged into their account. The system provides a user interface to set up role management for employees.
Test Case:	Expected Results
TC-1 Company admin assigns the role "Manager" for an employee.	The role is successfully assigned to the corresponding employee, and permissions are configured to allow access to relevant features and data.
TC-2 Company admin assigns the role "Financial Manager" for an employee.	The role is successfully assigned to the corresponding employee, and permissions are configured to allow access to relevant features and data.
TC-3 Company admin assigns the role "Operator" for an employee.	The role is successfully assigned to the corresponding employee, and permissions are configured to allow access to relevant features and data.

Test ID	AT-6
Requirement Tested	The system should allow condo management companies to be able to delete employee profiles from the system to remove individuals who are no longer employed within the company.
Acceptance Criteria	User is logged into their account. The system provides a list of employee profiles registered in the system.
Test Case:	Expected Results
TC-1 Company searches for and selects profile they wish to delete	The system locates and displays the desired employee profile for deletion.
TC-2 Company initiates the deletion action and confirms the action when prompted.	The system displays a confirmation prompt to verify the deletion action. Upon confirmation, the selected employee profile is permanently deleted from the system.
TC-3 Company verifies the deletion by confirming that the employee profile is no longer accessible in the system	The selected employee's profile is successfully removed from the system and it is no longer listed in the employee registry list. All associated data and permissions are also deleted.

Test ID	AT-7
Requirement Tested	The system should allow companies to delete a property profile in the contents of their property list.
Acceptance Criteria	The user is logged into their account. The system displays the user interface for the list of properties under the company's ownership.
Test Case:	Expected Results
TC-1 Company selects a property from the properties list and initiates the deletion action.	The system displays a confirmation prompt to confirm the deletion. Upon confirmation, the property entry is successfully removed from the system.

Test ID	AT-8
Requirement Tested	The system should allow condo users to view condo profiles of the condos they own.
Acceptance Criteria	The user is logged into their account. The system displays the condo owner's dashboard.
Test Case:	Expected Results
TC-1 Condo owner navigates to the "My Condos" section from the dashboard.	The system displays the list of condos under the condo owner's ownership.
TC-2 Condo owner selects a specific condo from the condos list to view its profile.	The system navigates to and displays the selected condo profile and displays its details, such as size, occupant name, and condo fee.

6. TEST RESULT

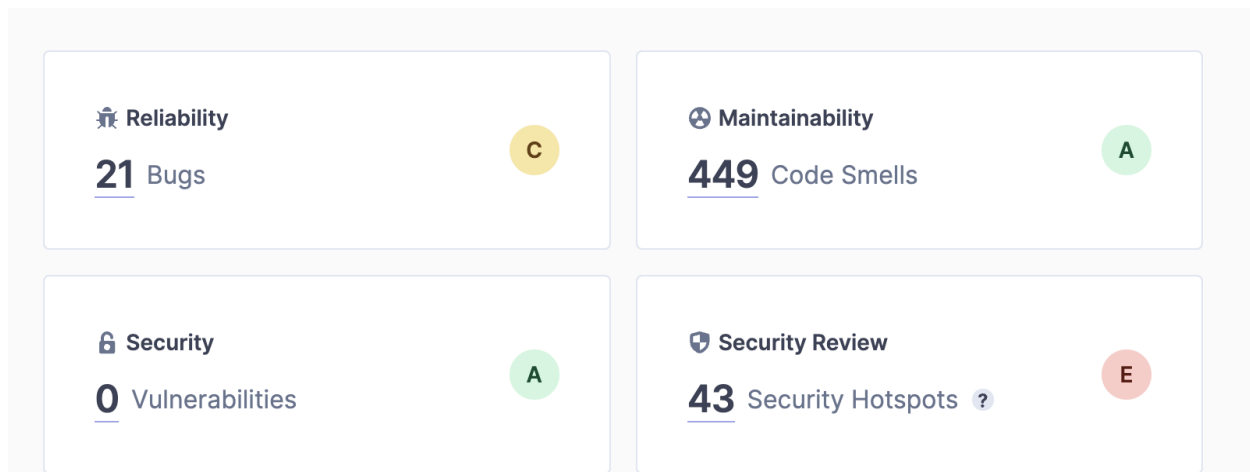


Figure 23: SonarQube Analytics Result for overall Code