SOEN390 - Software Engineering Team Design Project
Team 6 - Deliverable 4

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

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.

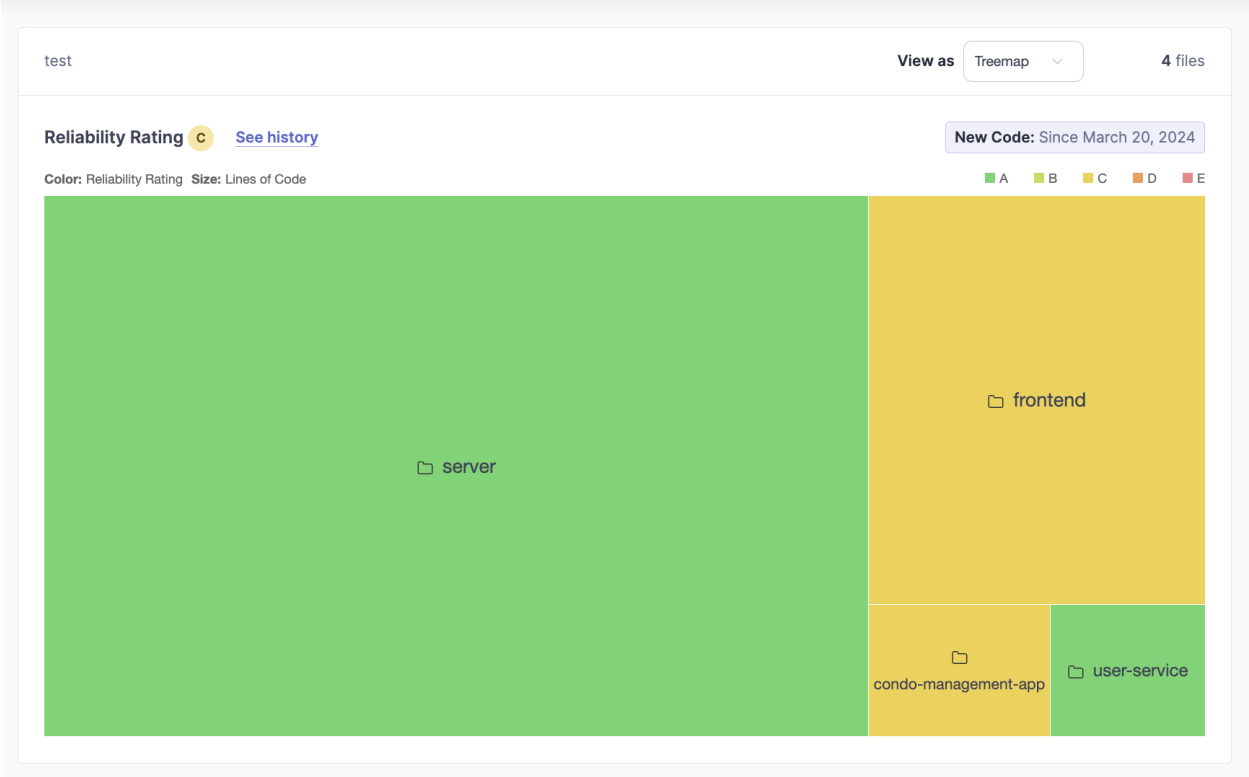


Figure 2: Reliability Report on Sprint 4

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 4

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 4

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.

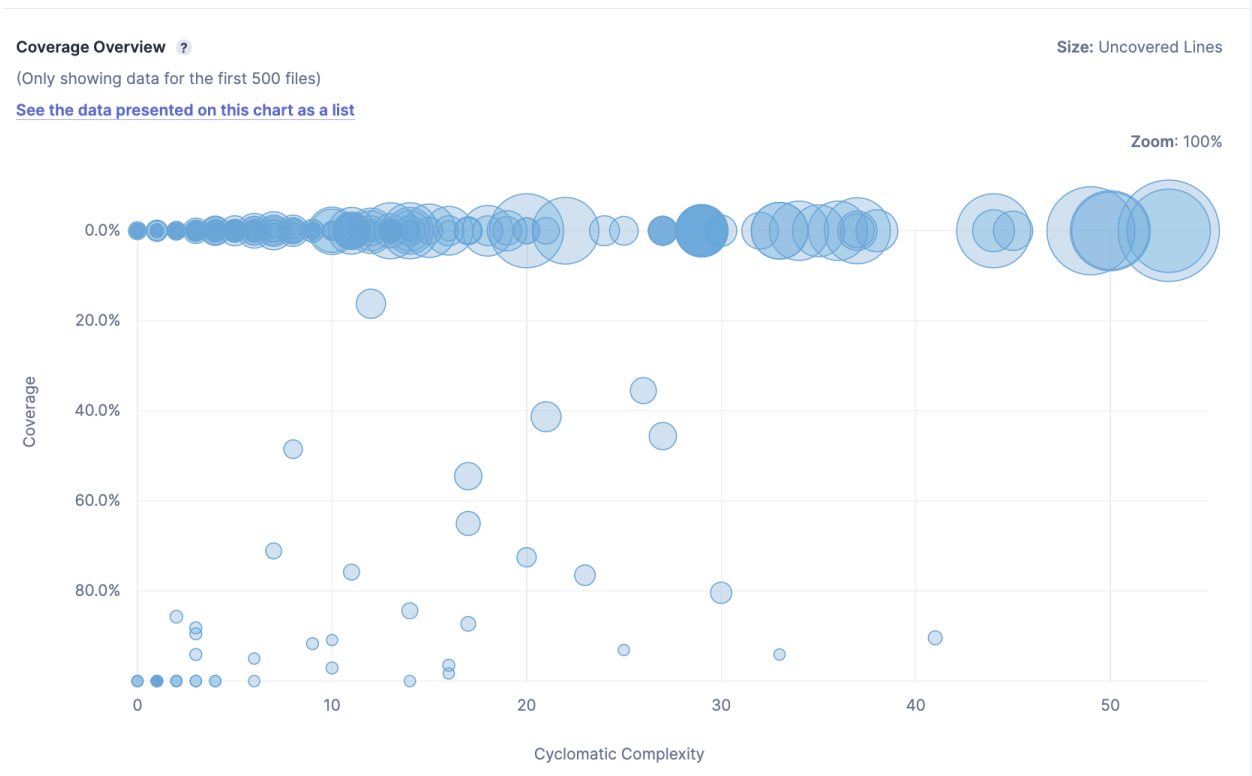


Figure 5: Coverage Report on Sprint 4

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.

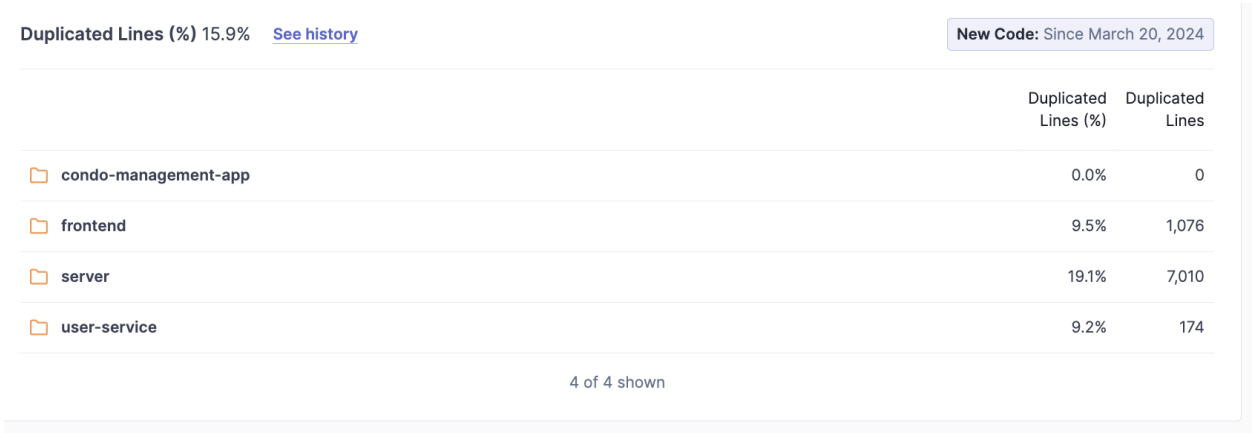


Figure 6: Duplication Report for sprint 4

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.

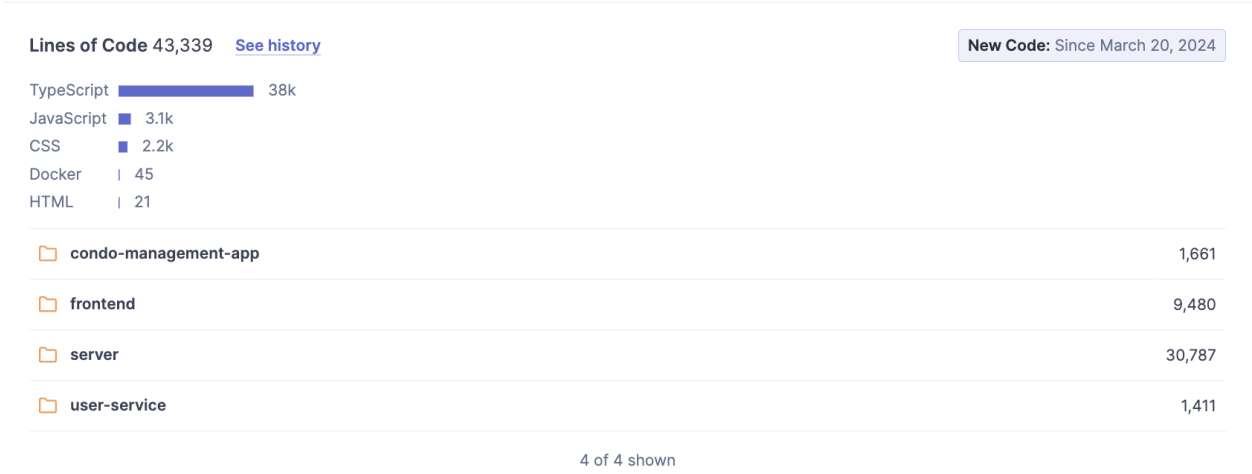


Figure 7: Line Of Code Report for Sprint 4

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 8: Cyclomatic Complexity Report for sprint 4

Figure 9: Cognitive Complexity Report for sprint 4

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

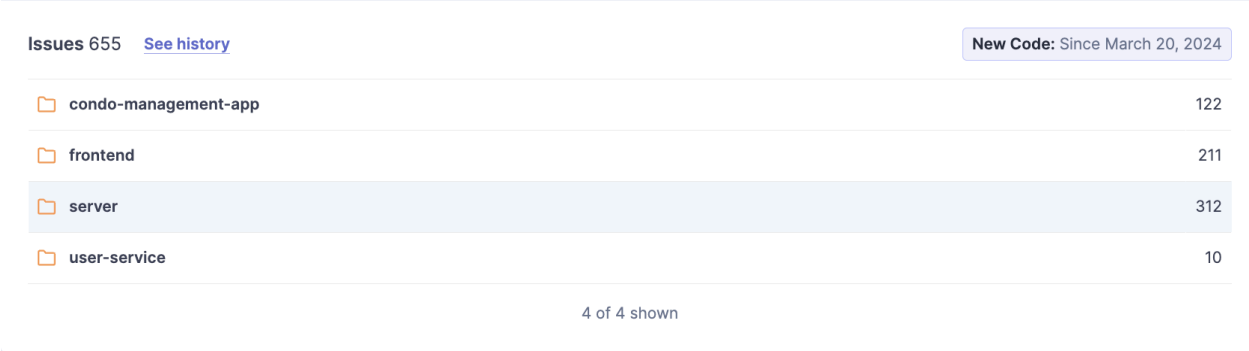


Figure 10: Issue Report for sprint 4

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, 2 and 3.
On the frontend side, the statement coverage overall was 81.14%, the branch coverage was 82.82%, function coverage was 84.62% and lines were at 80.85% (as shown in the figure below. Therefore we succeeded at achieving over 80% for everything. We had 59 test suites (15 additional test cases from sprint 3) and we had 209 test cases(27 additional test cases from sprint 3) which all passed without any failure. These numbers are significant compared to sprint 3. However, we can see that percentages fell due to how complicated the system started to get, but we are still above 80% in all metrics.

On the backend side, the statement coverage overall was 97.48%, the branch coverage was 92.07%, function coverage was 92.69% and lines were at 94.64%. We were able to achieve over 80% coverage for all of these criterias. In total, there were 31 test suites compared to the 28 test suites from sprint 4 and 259 test suites this sprint compared to 251 testsuites last sprint. We have added a significant amount of tests compared to previous sprints to account for adding the financial system on our website. All of the test suites passed without exception. Compared to sprint 3, sprint 4's statement coverage increased by around 2% which is due to the rigorous testing methods executed in this sprint for the additional functionalities. Our branch coverage has improved by 7%, while function and line coverage increased by 3% and 2 % respectively. That being said, all our test metrics exceed the 80% mark despite the additional functionalities added in sprint 4.

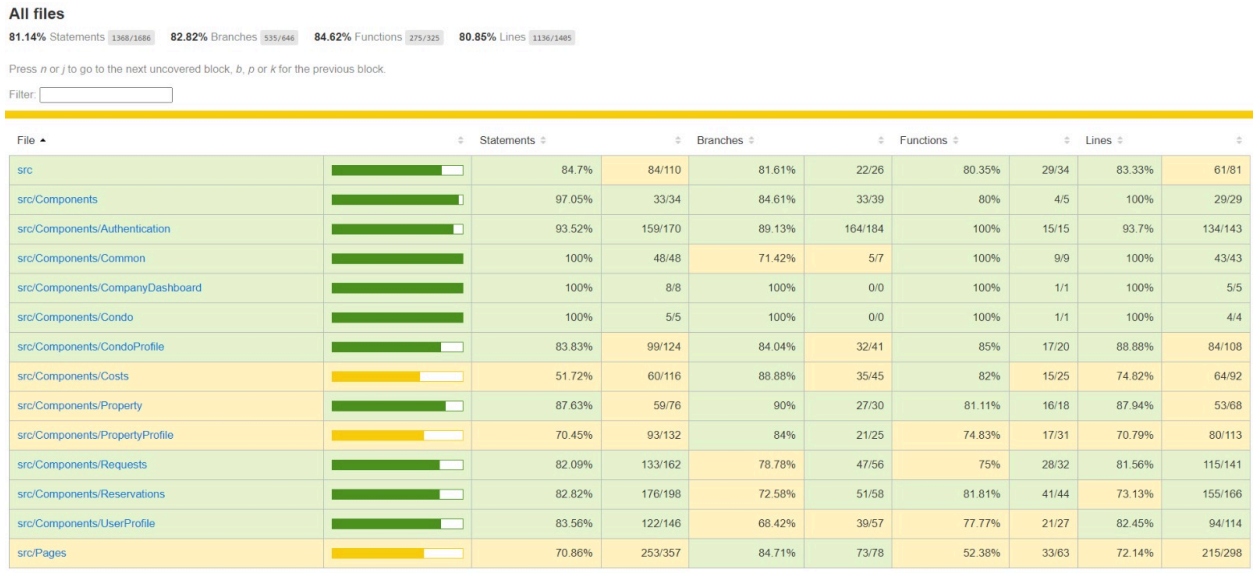
One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.
For the future, we plan to maintain the overall code coverage 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 5 Testing Plan for Code Coverage
Our plan for Sprint 5 is to maintain over 80% 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:



All files

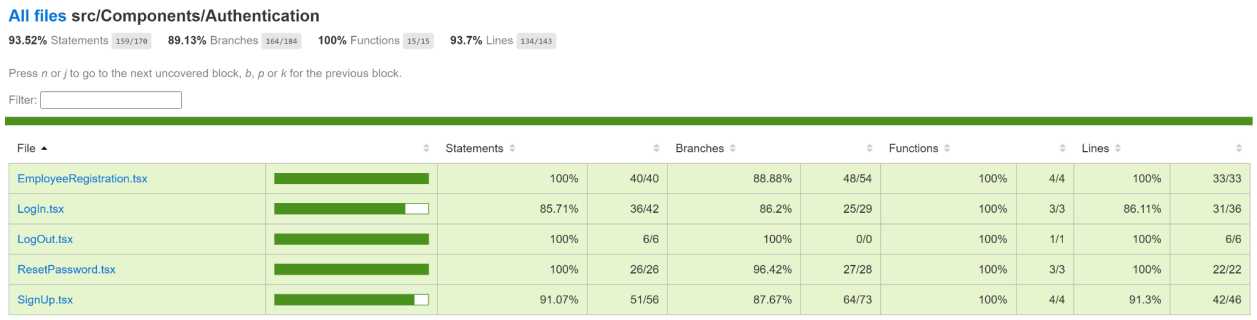
81.14% Statements 1368/1686 **82.82%** Branches 535/646 **84.62%** Functions 275/325 **80.85%** Lines 1136/1405

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	
src	84.7%	84/110	81.61%	22/26	80.35%	29/34	83.33%	61/81
src/Components	97.05%	33/34	84.61%	33/39	80%	4/5	100%	29/29
src/Components/Authentication	93.52%	159/170	89.13%	164/184	100%	15/15	93.7%	134/143
src/Components/Common	100%	48/48	71.42%	5/7	100%	9/9	100%	43/43
src/Components/CompanyDashboard	100%	8/8	100%	0/0	100%	1/1	100%	5/5
src/Components/Condo	100%	5/5	100%	0/0	100%	1/1	100%	4/4
src/Components/CondoProfile	83.83%	99/124	84.04%	32/41	85%	17/20	88.88%	84/108
src/Components/Costs	51.72%	60/116	88.88%	35/45	82%	15/25	74.82%	64/92
src/Components/Property	87.63%	59/76	90%	27/30	81.11%	16/18	87.94%	53/68
src/Components/PropertyProfile	70.45%	93/132	84%	21/25	74.83%	17/31	70.79%	80/113
src/Components/Requests	82.09%	133/162	78.78%	47/56	75%	28/32	81.56%	115/141
src/Components/Reservations	82.82%	176/198	72.58%	51/58	81.81%	41/44	73.13%	155/166
src/Components/UserProfile	83.56%	122/146	68.42%	39/57	77.77%	21/27	82.45%	94/114
src/Pages	70.86%	253/357	84.71%	73/78	52.38%	33/63	72.14%	215/298

Figure 11: Main Page Report with JEST



All files src/Components/Authentication

93.52% Statements 159/170 **89.13%** Branches 164/184 **100%** Functions 15/15 **93.7%** Lines 134/143

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	
EmployeeRegistration.tsx	100%	40/40	88.88%	48/54	100%	4/4	100%	33/33
LogIn.tsx	85.71%	36/42	86.2%	25/29	100%	3/3	86.11%	31/36
LogOut.tsx	100%	6/6	100%	0/0	100%	1/1	100%	6/6
ResetPassword.tsx	100%	26/26	96.42%	27/28	100%	3/3	100%	22/22
SignUp.tsx	91.07%	51/56	87.67%	64/73	100%	4/4	91.3%	42/46

Figure 12: Detailed Statistics of the Authentication Folder and its Components

All files src/Components/Common

100% Statements 45/45 **71.42%** Branches 5/7 **100%** Functions 9/9 **100%** Lines 40/40

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	100%	7/7	100%	1/1	100%	1/1	100%	6/6
InitialValues.ts	100%	6/6	100%	0/0	100%	0/0	100%	6/6
List.tsx	100%	20/20	66.66%	4/6	100%	7/7	100%	16/16
LoadingScreen.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
ValidationSchema.ts	100%	7/7	100%	0/0	100%	0/0	100%	7/7

Figure 13: Detailed Statistics of the Common Folder with Reusable files

All files src/Components/UserProfile

84.82% Statements 123/145 **71.92%** Branches 41/57 **88.88%** Functions 24/27 **83.18%** Lines 94/113

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	
UserInfoFields.tsx	100%	14/14	100%	14/14	100%	7/7	100%	14/14
UserInformation.tsx	79.68%	51/64	58.33%	14/24	72.72%	8/11	76%	38/50
UserKeyRegister.tsx	77.5%	31/40	60%	9/15	100%	4/4	75.86%	22/29
UserProperties.tsx	100%	18/18	100%	4/4	100%	4/4	100%	13/13
UserRequests.tsx	100%	9/9	100%	0/0	100%	1/1	100%	7/7

Figure 14: Detailed Statistics of the UserProfile

90.24% Statements 74/82 **91.17%** Branches 31/34 **90.9%** Functions 10/11 **88.73%** Lines 63/71

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	82.05%	32/39	87.5%	21/24	100%	3/3	80.55%	29/36
CondoInfoField.tsx	100%	9/9	100%	4/4	100%	1/1	100%	9/9
CondoInfoForm.tsx	97.05%	33/34	100%	6/6	85.71%	6/7	96.15%	25/26

Figure 15: Detailed Statistics of the CondoProfile

The rest of the folders with no details have very few files, so it is already represented in the “All Files” section.

Backend Reports:

All files

97.48% Statements 1201/1232 **92.07%** Branches 186/202 **92.69%** Functions 203/219 **94.64%** Lines 1132/1196

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	
commonFacility/base	91.52%	54/59	87.5%	7/8	100%	10/10	91.22%	52/57
company/base	92.85%	130/140	100%	9/9	93.33%	28/30	94.92%	131/138
companyEmployee/base	98.33%	59/60	100%	12/12	100%	10/10	94.82%	55/58
condoUnit/base	97.57%	101/107	100%	19/19	90.92%	20/22	98.14%	101/105
cost/base	93.02%	40/43	100%	7/7	100%	6/6	100%	41/41
file/base	96.42%	54/56	100%	19/19	100%	6/6	96.29%	52/54
forum/base	91.52%	54/59	100%	8/8	100%	10/10	91.22%	52/57
health/base	100%	10/10	100%	1/1	100%	3/3	100%	10/10
locker/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41
notification/base	93.04%	40/43	81.81%	9/11	100%	6/6	100%	41/41
parkingSpot/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
post/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
property/base	90.1%	118/139	100%	13/13	93.33%	28/30	93.72%	130/137
registrationKey/base	95.45%	42/44	100%	3/3	100%	6/6	95.23%	40/42
request/base	87.58%	50/58	93.83%	22/24	100%	10/10	100%	56/56
reservation/base	95.34%	41/43	95.45%	10/11	100%	6/6	95.12%	39/41
role/base	95.34%	41/43	100%	3/3	100%	6/6	95.12%	39/41
user/base	80.12%	125/156	100%	10/10	88.23%	30/34	79.87%	123/154
userCondo/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41

Figure 16: Main Page Report of the Backend with JEST

File	Statements		Branches		Functions		Lines	
company/base	90.21%	83/92	90.66%	5/6	83.33%	16/18	90%	81/90
companyEmployee/base	95.45%	42/44	85.45%	9/11	100%	6/6	95.23%	40/42

Figure 17: Detailed Statistics of company

File	Statements		Branches		Functions		Lines	
condoUnit/base	85.71%	78/91	80.88%	17/18	88.88%	16/18	85.39%	76/89
userCondo/base	86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 18: Detailed Statistics of condo

File	Statements		Branches		Functions		Lines	
user/base	80.26%	49/92	16.66%	5/6	88.88%	18/18	100%	90/90
userCondo/base	86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 19: 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 4 Acceptance Tests

Test ID	AT-1
Requirement	As a condo owner, I want to be able to reserve common facilities in a calendar-like interface.
Acceptance Criteria	User is logged into their account.
Test Case	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-2
Requirement Tested	As a condo management company, I want to be able to set up a common facility which requires reservations.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to reserve a common facility.
Test Case:	Expected Results
TC-1 Company selects a date and time for a common facility that is available.	The reservation is successfully submitted, and the details are stored in the system.
TC-2 Company selects a date and time for a common facility that is unavailable.	The reservation is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-3

Requirement Tested	As a rental user, I want to be able to see the availability of common facilities.
Acceptance Criteria	The user is logged into their account.
Test Case:	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-4
Requirement Tested	As a rental user, I want to be able to modify or cancel my existing reservation for a common facility.
Acceptance Criteria	Users can access the list of their reservations from the Common Facilities Calendar Page. The Reservations List page should display a list of all reservations related to the user’s submissions.
Test Case:	Expected Results
TC-1 User deletes an existing reservation.	The system displays a confirmation prompt to confirm the deletion. Upon confirmation, the reservation entry is successfully removed from the system.
TC-2 User modifies an existing reservation with valid availability.	The system displays a confirmation prompt to confirm the modification. Upon confirmation, the reservation entry is successfully modified in the system.
TC-3 User deletes an existing reservation with invalid availability.	The modification is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-5
Requirement Tested	As a condo management company, I want to be able to disable the availability of a

	common facility.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to set the status of common facilities.
Test Case:	Expected Results
TC-1 Company assigns the status “Mark as Closed” to a common facility.	The status of the common facility is successfully updated in the system as closed, and permissions are configured to disallow users access to this facility for reservations.
TC-1 Company assigns the status “Mark as Open” to a common facility.	The status of the common facility is successfully updated in the system as open, and permissions are configured to allow users access to this facility for reservations.

Test ID	AT-6
Requirement Tested	As a condo management company, I want to be able to enter the condo fee per square foot, per parking spot for a condo unit.
Acceptance Criteria	The user is logged into their account. The system displays the user interface for setting condo fees for a condo unit.
Test Case:	Expected Results
TC-1 Company enters the fee per square foot.	The fee per square foot entry is successfully entered in the system.
TC-1 Company enters the fee per parking spot.	The fee per parking spot entry is successfully entered in the system.

Test ID	AT-7
Requirement Tested	As a condo management company, I want to be able to enter the cost for each operation.
Acceptance Criteria	User is logged into their account. The system provides a user interface to add cost for operations.

Test Case:	Expected Results
TC-1 Company selects an operation and assigns a cost to it.	The cost of the operation is successfully submitted and stored in the system.

Test ID	AT-8
Requirement Tested	As a condo owner, I want to be able to see the calculated condo fee for each unit.
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.

Test ID	AT-9
Requirement Tested	As a condo management company, I want to be able to see a record of the operational budget, which includes the total condo fees collected from condo owners.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Operational Budget Report" section from the dashboard.	The system displays the report for the condo fees collected from condo owners.

Test ID	AT-10

Requirement Tested	As a condo management company, I want to be able to see an annual report.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Annual Report" section from the dashboard.	The system displays the annual report, including details of the revenue, expenses and profits of the company.

6. TEST RESULT

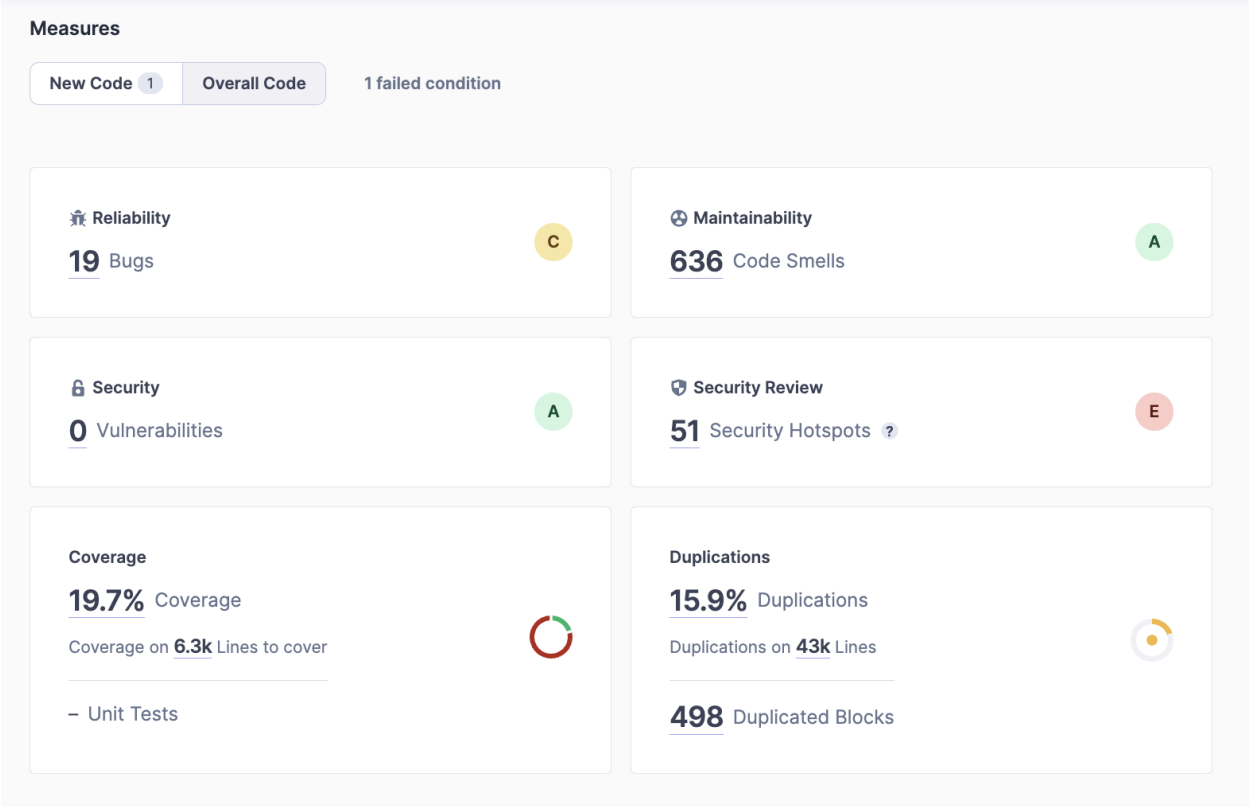


Figure 20: SonarQube Analytics Result for overall Code