Test Plan

NADKO DENTALS

FONTYS

2021

|  |  |  |
| --- | --- | --- |
| Version | Author | Changes |
| 1.0 | Tsanko Nedelchev | Initial release |
| 1.1 | Tsanko Nedelchev | Added e2e testing strategy ,specified the different user acceptance tests to be run, specified what commands to run for manual testing. |

Contents

[Test Strategy 2](#_Toc75112791)

[End-to-End testing 2](#_Toc75112792)

[User Acceptance tests 3](#_Toc75112793)

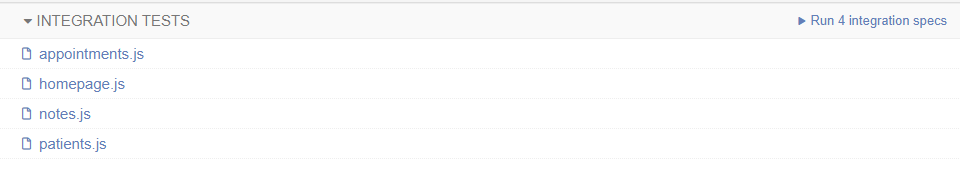
[Unit testing 4](#_Toc75112794)

[Quality assurance testing 4](#_Toc75112795)

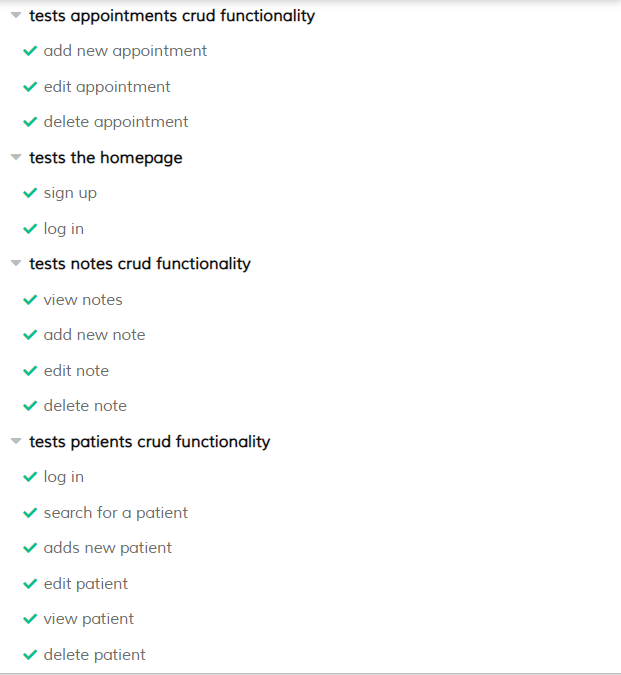
# Test Strategy

## End-to-End testing

End-to-end test implementation via Cypress per user story divided into 4 integration specs



Each having its own separate integration tests.



The test can be launched manually by running the command npm test in the root directory of the React.js project.

## User Acceptance tests

Testing done by an independent person who plays the part of the user in that test case. The purpose is to observe the user-friendliness of the application and to receive feedback on the different features.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test Name | Test Purpose | Test Data/Directions | Expected result |
| 1. | Sign Up | Test that a user is created in the database with their assigned role. | Username: testDentist Email: [test@email.com](mailto:test@email.com)  Password: password  Role: dentist | A user with the given credentials is created into the database. |
| 2. | Log In | Test that a previously created user can log in with their credentials. | Username: testDentist Password: password | The user can access their profile page. |
| 3. | Add patient | Test if the dentist can add a patient to their list of patients | First Name: test  Last Name: test  Email: [test@gmail.com](mailto:test@gmail.com)  Phone: +3112837621 | The new patient is added to the user’s list of patients. |
| 4. | Edit patient | Test if the dentist can edit the patient’s information. | First Name: testPatient | The new patient has their name changed from “test” to “testPatient” |
| 5. | Search for a patient | Test if the search functionality works | Search: testPatient | “testPatient” appears in the search results |
| 6. | View patient | Test if the dentist can view the patient’s personal pages where they can see their notes about the patient | Click on the “View” button of the selected patient | The dentist can see the patient’s data and their own notes about that patient. |
| 7. | Delete patient | Test if the dentist can remove a patient from their list of patients | Click on the “Remove” button of the selected patient | The removed patient is no longer displayed in the list of patients. |
| 8. | Add note | Test if the dentist can add a note about a patient | Note content: “Note about a patient” | The dentist can see the newly-added note in the patient’s info page. |
| 9. | Edit note | Test if the dentist can edit a note they created beforehand | Note content: “Note about a patient (edited)” | The dentist can see the updated note content with updated date and time. |
| 10. | Delete note | Test if the dentist can delete a note they previously created | Click on the trash can icon on the note that needs to be deleted. | The removed note is no longer visible. |
| 11. | Create an appointment | Test if the dentist can create a new appointment for their patient | Patient: choose from a dropdown list of patients Appointment Time: select a date and time for the appointment. | The new appointment is displayed in the list of appointments. |
| 12. | Edit an appointment | Test if a dentist can edit the appointment they previously created | Change either the time of the appointment or the patient who visits | The edited appointment has its data changed to the edited data. |
| 13. | Remove an appointment | Test if the dentist can remove and appointment from their list of appointments | Click on the “Remove” button of the selected appointment. | The removed appointment is no longer shown in the list of appointments. |

## Unit testing

Unit testing will be done automatically by the CI/CD pipeline every time a developer makes a commit to the master branch repository.

It can be done manually by running the command gradle test inside the root directory of the project.

## Quality assurance testing

Quality is monitored by SonarQube and an analysis is triggered by the CI/CD pipeline with each successful commit to the master branch repository.

It can be done manually by running the command gradle sonarqube inside the root directory of the project.