TALLINN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF SOFTWARE SCIENCE

Procurement Of Health Information System For 25 Primary Health Centers In Nigeria

Lab 2 in subject "Software Quality and Standards" (IDY0204)

Authors, student group:

Monika Shrestha
Kehinde Esther Ogundeyi
Chioma Jessica Nkem-Eze

Team ID: Digital Health
Supervisor: Jaak Tepandi
Presented: 24th Oct 2021

TALLINN 2021

Table of Content

4. Test-Driven Development

4.1. Requirements

To document our development process, we chose to develop all requirements necessary to complete our system's appointment module. We decided to select the appointment module due to its ease of development on an autonomous basis. The requirements are as follows:

Use Case ID	FR-004
Use Case Name	Creating new appointment
Primary Actor	Doctor
Preconditions	The patient record exists in the database
Postconditions	A new appointment is created and added to patient information
Main Success Scenario	• During appointment creation, the doctor cannot set an appointment end time that is before the appointment start time • The doctor successfully creates a new appointment for the patient
Alternative scenario	If it is not possible to create a new appointment, the doctor will receive an error message with information about the type of error (e.g. selected date is not correct)

Use Case ID	FR-005
Use Case Name	Modify appointment
Primary Actor	Doctor
Preconditions	The appointment exists in the HIS
Postconditions	The appointment’s time is changed
Main Success Scenario	The doctor modifies the appointment.

Use Case ID	FR-010
Use Case Name	See appointment view for doctor
Primary Actor	Doctor
Preconditions	The doctor has access to the schedule
Postconditions	The doctor can open schedule and see upcoming and past appointments
Main Success Scenario	• The doctor opens the appointment calendar. • The doctor sees the upcoming and past appointment schedule.
Alternative scenario	In case there are no appointments, an empty calendar is displayed.

4.2. Tools

For our system, we chose an open-source project that aligns with the requirements that we specified in Lab 1. Hospital Run is the name of this open-source project. It is a free hospital information system for developing world countries. Having stated that, for the remainder of this lab, we will utilize the same language (React TypeScript) and unit testing tool (Jest and React Testing Library) as in the project.

We chose GitHub for version control due to its prominence compared to other version control tools and its free public repositories.

4.3. Test-driven development cycle

We have a series of test cases for each requirement that validate our main success scenario for our cycles. We add each test and then add the necessary functionality that would make these tests pass. We build on the previous work done in each successive implementation and then refactor to keep the program simple. We will define these tests below and illustrate the TDD process by providing links to the commits.

4.3.1. Use Case FR004 - Creating a new appointment

Test 01	It should render an appointment component
Test code + initial boilerplate: This includes setup for the test, the test itself, and the creation of the NewAppointment component.	Commit Link
Implementation code. A label is added here to pass the layout test.	Commit Link

```
FAIL src/__tests__/scheduling/appointments/new/NewAppointment.test.tsx (21.77s)
  New Appointment
    layout
      × should render an Appointment Detail Component (1046ms)

  ● New Appointment › layout › should render an Appointment Detail Component

    TestingLibraryElementError: Unable to find a label with the text of: new appointment form
```

```
PASS src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx
  Edit Appointment
    ✓ should load an appointment when component loads (34ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.943s, estimated 10s
Ran all test suites matching /src\/__tests__\/scheduling\/appointments\/edit\/EditAppointment.test.tsx/i.
```

Test 02	It should have an error if a patient record does not exist
Test code: New test added here	Commit Link
Implementation code. The component is built with links to external functions	Commit Link

```
FAIL src/__tests__/scheduling/appointments/new/NewAppointment.test.tsx (9.414s)
  New Appointment
    layout
      ✓ should render an Appointment Detail Component (27ms)
    new appointment creation
      × should have error if patient record does not exist (11ms)

  ● New Appointment › new appointment creation › should have error if patient record does not exist

    TestingLibraryElementError: Unable to find an element with the placeholder text of: /scheduling\.appointment\.pa
tient/i
```

```
PASS src/__tests__/scheduling/appointments/new/NewAppointment.test.tsx (10.618s)
  New Appointment
    layout
      ✓ should render an Appointment Detail Component (115ms)
    new appointment creation
      ✓ should have error if patient record does not exist (719ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        12.86s
Ran all test suites matching /src\/__tests__\/scheduling\/appointments\/new\/NewAppointment.test.tsx/i.
```

Test 03	It should have an error when the end time is earlier than the start time
Test code: New test added here	Commit Link
Implementation code. A validation checker was added to check this scenario. Code is refactored	Commit Link

```
FAIL src/__tests__/scheduling/appointments/new/NewAppointment.test.tsx (16.141s)
  New Appointment
    layout
      ✓ should render an Appointment Detail Component (122ms)
    new appointment creation
      ✓ should have error if patient record does not exist (922ms)
      × should have error when end time earlier than start time (3902ms)

  ● New Appointment › new appointment creation › should have error when end time earlier than start time
```

```
PASS src/__tests__/scheduling/appointments/new/NewAppointment.test.tsx (13.969s)
  New Appointment
    layout
      ✓ should render an Appointment Detail Component (123ms)
    new appointment creation
      ✓ should have error if patient record does not exist (776ms)
      ✓ should have error when end time earlier than start time (3382ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        16.231s, estimated 17s
Ran all test suites matching /src\/__tests__\/scheduling\/appointments\/new\/NewAppointment.test.tsx/i.
```

4.3.2. Use Case FR005 - Modify appointment

Test 01	It should load an appointment when the component loads
Test code + initial boilerplate: This includes setup for the test, the test itself, and the creation of the EditAppointment component.	Commit Link

Implementation code. An appointment is gotten from a useAppointment hook	Commit Link

```
FAIL src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx (33.299s)
  Edit Appointment
    × should load an appointment when component loads (1032ms)

  ● Edit Appointment › should load an appointment when component loads
```

```
PASS src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx
  Edit Appointment
    ✓ should load an appointment when component loads (34ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.943s, estimated 10s
Ran all test suites matching /src\/__tests__\/scheduling\/appointments\/edit\/EditAppointment.test.tsx/i.
```

Test 02	It should render an edit appointment form
Test code: New test added here	Commit Link
Implementation Code. The AppointmentDetailForm is added.	Commit Link

```
FAIL src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx (6.381s)
  Edit Appointment
    ✓ should load an appointment when component loads (44ms)
    × should render an edit appointment form (1011ms)

  ● Edit Appointment › should render an edit appointment form
```

```
PASS src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx (7.99s)
  Edit Appointment
    ✓ should load an appointment when component loads (515ms)
    ✓ should render an edit appointment form (353ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        9.391s, estimated 10s
Ran all test suites matching /src\/__tests__\/scheduling\/appointments\/edit\/EditAppointment.test.tsx/i.
```

Test 03	It should edit an appointment when the save button is clicked
Test code: New test added here	Commit Link
Implementation Code. Save functionality added	Commit Link

```
FAIL src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx (11.499s)
  Edit Appointment
    ✓ should load an appointment when component loads (601ms)
    ✓ should render an edit appointment form (273ms)
    × should editAppointment when save button is clicked (1251ms)

  ● Edit Appointment › should editAppointment when save button is clicked
```

```
PASS src/__tests__/scheduling/appointments/edit/EditAppointment.test.tsx (8.869s)
  Edit Appointment
    ✓ should load an appointment when component loads (605ms)
    ✓ should render an edit appointment form (448ms)
    ✓ should editAppointment when save button is clicked (494ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        10.215s, estimated 12s
```

4.3.3. Use Case FR010 - See appointment view for doctor

Test 01	It should render a calendar view with the proper events
Test code + initial boilerplate: This includes setup for the test, the test itself, and the creation of the ViewAppointments component.	Commit Link
Implementation of the calendar view	Commit Link

```
FAIL src/__tests__/scheduling/appointments/ViewAppointments.test.tsx (5.76s)
  ViewAppointments
    × should render a calendar with the proper events (1024ms)

  ● ViewAppointments › should render a calendar with the proper events
```

```
PASS src/__tests__/scheduling/appointments/ViewAppointments.test.tsx (18.555s)
  ViewAppointments
    ✓ should render a calendar with the proper events (785ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        24.379s
Ran all test suites matching /src\/__tests__\/scheduling\/appointments\/ViewAppointments.test.tsx/i.
```

5. Code Coverage

5.1. Program presentation

Following the activities in Section 4, we will analyze the code coverage of the appointment module. This module is made up of several interconnected files, which include some major components and helper functions. For this lab, we will pick two of these components to analyze. These two components form a cohesive flow that is simple enough to provide code coverage for the lab, but complex enough to fulfil the complexity criteria. The components are:

1. **appointments/Appointments.tsx**: This is the appointment module's primary entry point and is responsible for the appointment view being displayed on the dashboard. This is accomplished by importing all major feature components and then defining private routes for each of them. Each of the routes has an `isAuthenticated condition to check if the signed in doctor has permission to get or update appointments. We include a link to the file on GitHub for reference purposes.
2. **appointments/view/ViewAppointment.tsx**: This file is one of the major components defined in the Appointments.tsx file. It is responsible for all the logic pertaining to viewing a single appointment, as well as performing actions on them. We include a link to the file on GitHub for reference purposes.

5.2. Coverage tests

We use the branch coverage criterion for testing. The outline of the different branches within the component we have chosen, as well the tests for them are prepared below:

5.2.1. Coverage for Appointments.tsx

For this file, we are interested in designing tests for the Appointments component, as well as the `useSelector` function call (permissions). We test these by writing tests for the different routes. This is because they exist within the Appointment component and use the permissions variable.

There are four private routes in total. Each route should render its respective component if the user has certain permission, the alternative would be that it does not throw an error, but renders the dashboard instead. We write tests for each of the routes to test these two scenarios.

```
const Appointments = () => {
  const permissions = useSelector((state: RootState) => state.user.permissions)
  return (
    <Switch>
      <PrivateRoute
        isAuthenticated={permissions.includes(Permissions.ReadAppointments)}
        exact
        path="/appointments"
        component={ViewAppointments}
      />
      <PrivateRoute
        isAuthenticated={permissions.includes(Permissions.WriteAppointments)}
        exact
        path="/appointments/new"
        component={NewAppointment}
      />
      <PrivateRoute
        isAuthenticated={
          permissions.includes(Permissions.WriteAppointments) &&
          permissions.includes(Permissions.ReadAppointments)
        }
        exact
        path="/appointments/edit/:id"
        component={EditAppointment}
      />
      <PrivateRoute
        isAuthenticated={permissions.includes(Permissions.ReadAppointments)}
        exact
        path="/appointments/:id"
        component={ViewAppointment}
      />
    </Switch>
  )
}

export default Appointments
```

/appointments - For this path, we have:

1. The appointment view should render when "*/appointments*" is accessed
2. The dashboard should be rendered when the user does not have read appointment privileges

/appointments/new - For this path, we have:

1. The new appointment view should render when "*/appointments/new*" is accessed
2. The dashboard should be rendered when the user does not have read appointment privileges

/appointments/edit/:id - For this path we have:

1. The edit appointment view should render when "*/appointments/edit/:id*" is accessed
2. The dashboard should be rendered when the user does not have read appointment privileges

3. The dashboard should be rendered when the user does not have write appointment privileges

/appointments/:id - For this path, we have:

1. The appointment view whose id is passed should render when "*/appointments/id*" is accessed
2. The dashboard should be rendered when the user does not have read appointment privileges

5.2.2. Coverage for ViewAppointment.tsx

For this file, there are several implementations that we are interested in testing to ensure complete coverage. They are:

The *"getButtons"* function; This function handles what should happen to the button toolbar if certain permissions are not available. For this, we write three tests to test each permission, as well as if the permissions are not available.

```
const getButtons = useCallback(() => {
  const buttons: React.ReactNode[] = []
  if (appointment && permissions.includes(Permissions.WriteAppointments)) {
    buttons.push(
      <Button
        key="editAppointmentButton"
        color="success"
        icon="edit"
        outlined
        onClick={() => {
          history.push(`/appointments/edit/${appointment.id}`)
        }}
      >
        {t('actions.edit')}
      </Button>,
    )
  }

  if (permissions.includes(Permissions.DeleteAppointment)) {
    buttons.push(
      <Button
        key="deleteAppointmentButton"
        color="danger"
        icon="appointment-remove"
        onClick={onAppointmentDeleteButtonClick}
      >
        {t('scheduling.appointments.deleteAppointment')}
      </Button>,
    )
  }

  return buttons
}, [appointment, history, permissions, t])

useEffect(() => {
  setButtonToolBar(getButtons())

  return () => {
    setButtonToolBar([])
  }
}, [getButtons, setButtonToolBar])
```

The tests are:

1. It should add an "Edit Appointment" button to the button toolbar if the doctor has WriteAppointment permissions
2. It should add a "Delete Appointment" button to the button toolbar if the doctor has DeleteAppointment permissions
3. The button toolbar should be empty if the doctor has only ReadAppointments permission

The "*ViewAppointment*" component; This component conditionally renders a loading spinner if the system is still trying to get the patient and appointment details. When these details come in, it renders an AppointmentDetailsForm and Modal. We write two tests to ensure that each of the scenarios is met in the different conditions.

```
return (
  <>
    {patient && appointment ? (
      <div>
        <AppointmentDetailForm appointment={appointment} isEditable={false} patient={patient} />
        <Modal
          body={t('scheduling.appointment.deleteConfirmationMessage')}
          buttonsAlignment="right"
          show={showDeleteConfirmation}
          closeButton={{
            children: t('actions.delete'),
            color: 'danger',
            onClick: onDeleteConfirmationButtonClick,
          }}
          title={t('actions.confirmDelete')}
          toggle={() => setShowDeleteConfirmation(false)}
        />
      </div>
    ) : (
      <Spinner type="BarLoader" loading />
    )}
  </>
)
```

The tests are:

1. It should render a loading spinner if details are not met
2. It should render the AppointmentDetailForm with the correct data

The "*onDeleteConfirmationButtonClick*" function; This function is responsible for deleting the appointment from the database, as well as closing the delete confirmation modal. delete confirmation modal shows up when the doctor clicks on the delete action. As its name implies, it asks for confirmation to delete an appointment. We write one test for this function to ensure that it indeed deletes the appointment.

```
const onDeleteConfirmationButtonClick = () => {
  if (!appointment) {
    return
  }

  deleteMutate({ appointmentId: appointment.id }).then(() => {
    history.push('/appointments')
    Toast('success', t('states.success'), t('scheduling.appointment.successfullyDeleted'))
  })
  setShowDeleteConfirmation(false)
}
```

5.2.3. Coverage decisions

All of the functions that we picked exist within the components. There are several other functions that are used within the components but are defined externally. They are not within the scope of this program.

5.3. Test and evaluation

The initial open-source project analyzes code coverage using a tool called coveralls. It includes a command for this purpose in the 'package.json' file. This was used to conduct our code coverage analysis.

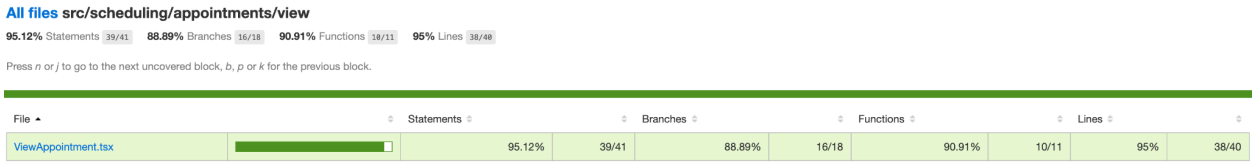
To examine the results locally, first, complete the installation process indicated in Section 6.1, and then run the code coverage analysis script command (npm run coveralls). This command creates a coverage subdirectory in the project's root directory containing the coverage analysis results.

In a browser, navigate to the index.html file in the root directory of the coverage folder. This page contains a report on the coverage of all system components. To find the components utilized in this lab, conduct a search for the appointment files.

5.4. Presentation

Appointments.tsx		100%	4/4	100%	2/2	100%	2/2	100%	3/3

According to our coverage report, we have branch coverage of 100 per cent on the Appointment.tsx component.



All files src/scheduling/appointments/view

95.12% Statements 39/41 **88.89%** Branches 16/18 **90.91%** Functions 10/11 **95%** Lines 38/40

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

File	Statements		Branches		Functions		Lines	
ViewAppointment.tsx	95.12%	39/41	88.89%	16/18	90.91%	10/11	95%	38/40

```
const onDeleteConfirmationButtonClick = () => {
  if (!appointment) {
    return
  }

  deleteMutate({ appointmentId: appointment.id }).then(() => {
    history.push('/appointments')
    Toast('success', t('states.success'), t('scheduling.appointment.successfullyDeleted'))
  })
  setShowDeleteConfirmation(false)
}

const getButtons = useCallback(() => {
  const buttons: React.ReactNode[] = []
  if (appointment && permissions.includes(Permissions.WriteAppointments)) {
    buttons.push(
      <Button
        key="editAppointmentButton"
        color="success"
        icon="edit"
        outlined
        onClick={() => {
          history.push(`/appointments/edit/${appointment.id}`)
        }}
      >
        {t('actions.edit')}
      </Button>,
    )
  }
```

According to our coverage report, we have an 88.89 percent branch coverage on the ViewAppointment.tsx component. As illustrated in the second graphic, the reason why we do not have full coverage is that we omitted checks for two scenarios. The first is for the case where the "onDeleteConfirmationButtonClick" method returns null when there is no appointment. The second is for the case when the "editAppointmentButton" is clicked. We do not have tests for the onClick function. Having stated this, the coverage we have received is favourable.

6. System For Further Testing

6.1. Reference to system location

The system used in this lab is based on the open-source software Hospital Run. To view the application in its entirety in a browser, click on the Hospital Run demo link. There are no preliminaries required for access.

6.1.1. Software Details

Version	v2.0.0-alpha.7
Producer	OpenJS Foundation
Authors	Open-source contributors
References	Hospital Run README, Hospital Run Website
License	Released under the MIT license
Duration	Not specified

6.1.2. Installation source reference

To carry out the previous chapters' activities, we cloned the open-source project to another repository. This section will walk you through the installation process for this version.

1. Make sure you have Git installed.
2. Make sure you have installed Node.js. We recommend that you use the most recent "Active LTS" version of Node.js.
3. Make sure you have npm installed.
4. Clone this repo with `*git clone https://github.com/fegaeze/HRLab02*`, go to the cloned folder and run `*npm install*`.
5. Start the server by running npm start in the repo folder. If npm start doesn't work for you, run this command:
   `*npm i --save-dev @types/node@15*` or `*npm i --save-dev @types/node@14*` and try again.
6. Go to *http://localhost:3000/* in a browser
7. For running tests:
   a. `*npm run test:ci*` will run tests for the entire file
   b. `*npm run test:ci <filename>*` e.g.
      `*npm run test:ci src/__tests__/HospitalRun.test.tsx*` will run that specific test file

c. *`npm run coveralls`* will run coverage analysis for the entire file, and generate a coverage subdirectory.

6.1.3. Additional components or activities needed for installation

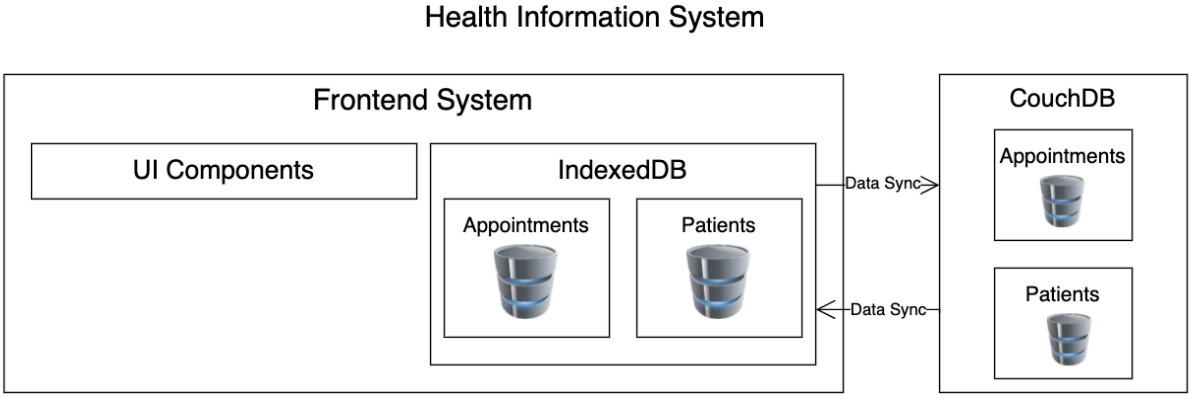
All components needed for installation have been outlined in section 6.1.2

6.1.4. Reasons for software selection

HospitalRun is a free hospital information system that meets the requirements of our project. The most prominent feature is the offline-first mode. Additionally, it is constructed using Typescript React, a very popular language that is acquainted with the team in case of future maintenance.

6.2. Architecture

The figure in lab one examined the context of the system. This diagram expands on that by outlining the health information system's high-level technical components.



An enhanced architecture of the software