Anthony Krivonos (ak4483)

Kenneth Chuen (kc3334)

Kevin Wong (hw2735)

William Pflueger (fwp2108)

November 17th, 2020

Adv. SWE Assignment T3

Prof. Gail Kaiser

Team: Starmen

Part 1 - Github

https://github.com/anthonykrivonos/4156-Starmen

Part 2 - User Stories and Acceptance Testing

User Stories

Login as an HCP and view my profile:

As an HCP, I want to be able to login with my Google account and view and edit my current upMed account information. I should be able to login via my Google account and then take me to the HCP portal, where I should be able to view the information that is currently in my profile and edit any fields, such as phone numbers or my hours. My conditions of satisfaction is to login, view my profile, edit my profile and save my edits.

Access patient's own health records:

As a patient, I want my health records to be kept securely with easy access so that I can take reference to them during my consultation. I should be able to input some health data such as if I am a smoker or if I drink and body weight. My conditions of satisfaction are able to review all of my records.

View, add and edit patient's health records:

As a HCP, I want to view, add, and modify my patients' healthcare records of vastly diverse types (lab results, radiology results, consultation notes). My conditions of satisfaction are able to conveniently view, add, update and edit health records if I am assigned to my patient.

Keep a schedule of my appointments:

As a HCP, I need to schedule appointments with all my patients so that I can meet my patients in an orderly manner and use my time efficiently. My conditions of satisfaction are able to use a calendar to schedule appointments and notify my patients about their appointments.

Acceptance Testing

Acceptance Testing is as follows: we simulate a Patient and HCP, and have them complete all related user stories.

Patient

Sign up as a Patient

- Navigate to website, and click “Get Started” (or “Sign In”)
- Select “Log in as a Patient”
- Choose Google account to sign up with
- Test with valid input:
    - Valid inputs:
        - Fill in extra credential in the proper data fields and click “Create Account”
            -
        - Having all data fields valid allows “Create Account” to be pressed
        - Having incomplete data fields grays out “Create Account” and prevents account creation
    - Expected result:
        - Patient is presented with a view of the calendar with upcoming appointments, a button to schedule new appointments, and sidebar options.
- Test with invalid input:
    - Invalid inputs:
        - Empty out one or more necessary credential details in the proper data fields and click “Create Account”
    - Expected result:
        - Having incomplete data fields grays out “Create Account” and prevents account creation
        - Any invalid data field is highlighted with a red border, and prevents account creation
- ENDPOINTS TESTED
    - /patient/signUp

Login as a Patient

- Test with valid input:
    - Valid inputs:
        - Navigate to website, and click “Get Started” (or “Sign In”)
        - Select “Log in as a Patient”
        - Choose Google account to log in with
    - Expected result:

        - Patient is presented with a view of the calendar with upcoming appointments, a button to schedule new appointments, and sidebar options.
- No possible invalid inputs (user story only involves clicking)
- ENDPOINTS TESTED
    - /patient/logIn
    - /patient/getByToken

Access patient's own health records

- Test with valid input:
    - Valid inputs:
        - Click Log in / sign up as a Patient
        - Select "Medical Info" on the sidebar.
    - Expected result:
        - Patients can view notes from recent appointments, past health records, and contact information for their doctors.
- No Possible invalid inputs (user story only involves clicking)
- ENDPOINTS TESTED
    - /patient/getByToken
    - /patient/getRecords

Edit patient's own profile

- Log in / sign up as a Patient
- Select "Edit Profile" on the sidebar.
- Patients can view their basic information (name, email, phone number) as well as personal health information (height, weight, smoker/drinker history).
- Patient can edit any of these fields:
- Test with valid input:
    - Valid inputs:
        - Having all data fields correctly inputted
    - Expected Results:
        - Allows "Update Account" button to be pressed
- Test with invalid input:
    - Invalid inputs:
        - Incomplete data fields
        - invalid data (e.g. string in weight field)
    - Expected Results:
        - Grays out "Update Account" and prevents account updates
        - Invalid data field is highlighted with a red border, and prevents account updates
        - Leaving this page without pressing "Update Account" will not change any profile values.
- ENDPOINTS TESTED
    - /patient/getByToken
    - /patient/editProfile

Keep a schedule of patient's appointments

- Valid inputs:
    - Click Log in / sign up as a Patient
    - Select "Appointments" on the sidebar.
- Expected result:
    - Patients can view his past and present appointments and its details

    - No possible invalid inputs (user story only involves clicking)
- ENDPOINTS TESTED
    - /patient/getByToken
    - /appointment/getByToken
    - /appointment/getCalendar

Create and delete an appointment

- Valid inputs:
    - Click Log in / sign up as a Patient
    - Select “Appointments” on the sidebar.
- Expected result:
    - Patients can view his past and present appointments and its details
    - No possible invalid inputs (user story only involves clicking)
- ENDPOINTS TESTED
    - /patient/getByToken
    - /hcp/getAll
    - /appointment/getByToken
    - /appointment/createAppointment
    - /appointment/deleteAppointment

Healthcare Professional

Sign up as a HCP

- Navigate to website, and click “Get Started” (or “Sign In”)
- Select “Log in as a HCP”
- Choose Google account to sign up with
- Fill in extra credential in the proper data fields and click “Create Account”
- Test with valid input:
    - Valid inputs:
        - Having all data fields input with valid data
    - Expected result
        - “Create Account” button is allowed to be pressed
- Test with invalid input:
    - Invalid inputs:
        - Incomplete data fields
        - invalid data
    - Expected result
        - Having incomplete data fields grays out “Create Account” and prevents account creation
        - Any invalid data field is highlighted with a red border, and prevents account creation
- ENDPOINTS TESTED
    - /hcp/signUp

Login as a healthcare professional

- Test with valid input:
    - Valid inputs:
        - Navigate to website, and click “Get Started” (or “Sign In”)
        - Select “Log in as a HCP”

        - Choose Google account to log in with
    - Expected result:
        - Patient is presented with a view of the calendar with upcoming appointments, a button to schedule new appointments, and sidebar options.
- No possible invalid inputs (user story only involves clicking)
- ENDPOINTS TESTED
    - /hcp/logIn
    - /hcp/getByToken

Edit HCP's own profile

- Log in / sign up as a HCP
- Select "Edit Profile" on the sidebar.
- HCP can view their basic information (name, email, phone number) as well as his office hours.
- HCP can edit any of these fields:
    - Having all data fields valid allows "Update Account" to be pressed
    - Having incomplete data fields grays out "Update Account" and prevents account updates
    - Any invalid data field is highlighted with a red border, and prevents account updates
- Leaving this page without pressing "Update Account" will not change any profile values.
- Test with valid input:
    - Valid inputs:
        - Having all data fields correctly inputted
    - Expected Results:
        - Allows "Update Account" button to be pressed
- Test with invalid input:
    - Invalid inputs:
        - Incomplete data fields
        - invalid data (e.g. string in weight field)
    - Expected Results:
        - Grays out "Update Account" and prevents account updates
        - Invalid data field is highlighted with a red border, and prevents account updates
        - Leaving this page without pressing "Update Account" will not change any profile values.
- ENDPOINTS TESTED
    - /hcp/getByToken
    - /hcp/editProfile

Access and edit a patient's health records

- Log in / sign up as an HCP
- Select "Medical Info" on the sidebar
- Select the relevant Patient
- HCP can view Patient Details (DOB, sex, height, weight, email, phone, drinker/smoker history), and select "Add Health Record"
- Test with valid input:
    - Valid inputs:
        - HCP will fill in relevant fields, and select "Add Health Record" again
    - Expected Results:
        - The patient's health record is successfully added
- Test with invalid input:
    - Invalid inputs:
        - Incomplete data fields
        - invalid data (e.g. string in weight field)

        - Expected Results:
            - Grays out “Update record” and prevents health record updates
- ENDPOINTS TESTED
    - /patient/getRecords
    - /hcp/getByToken
    - /hcp/setRecords

Keep a schedule of my appointments

- Log in / sign up as an HCP
- HCP is presented with a view of the calendar with upcoming appointments, a button to schedule new appointments, and sidebar options.
- ENDPOINTS TESTED
    - /hcp/getByToken
    - /appointment/getByToken
    - /appointment/getCalendar
    - /hcp/notify

Create and delete an appointment

- Valid inputs:
    - Click Log in / sign up as a HCP
    - Select “Appointments” on the sidebar.
- Expected result:
    - Patients can view his past and present appointments and its details
- No possible invalid inputs (user story only involves clicking)
- ENDPOINTS TESTED
    - /patient/getByToken
    - /hcp/getAll
    - /appointment/getByToken
    - /appointment/createAppointment
    - /appointment/deleteAppointment

Part 3 - Unit Testing

Front-end:

- The folder that contains the test cases is located here, and has sub-folders that mimic /src.
- The folder that contains the reports (unit.txt and component-unit.txt) created by running ‘yarn jest’ (our testing tool) is located here. This is the same folder that contains our style/bug/lint reports.

Back-end:

- The folder with the test cases are located here, each module as a separate test file.
- The outputs for the unit tests are located in the same folder as the other reports here. The output of the tests will output to tests.txt here.

Backend unit testing is as follows:

- /api
  - /appointment_test
    - createAppointment_test (test creating an appointment)
    - getCalendar_test (test getting the appointment given the id)
    - getByToken_test (test getting the appointment associated with the logged in account)
    - delete_appointment_test (test delete an appointment)
  - /hcp_test
    - signup_test (test sign up function using dummy hcp profile)
    - login_test (test the login function)
    - set_health_event_test (test create a health event)
    - notify_test (test to create a notification to patient)
    - remove_test (test to remove a HCP account)
  - /patient_test
    - Signup_test (test sign up function using dummy patient profile)
    - login_test (test the login function)
    - getbytoken_test (get a patient profile by logged in token)
    - getRecords_test (get a patient's health record by logged in token)
    - remove_test (test to remove a patient account)
    - editProfile_test (test edit the patient's profile)
    - gethcps_test (test to assess all the healthcare professionals taking care of the patient)
    - get_all_test (test obtain all of the patient's profile)
- /models
  - /appointment_model_test
  - /day_model_test

Part 4 - Style/Bug Checker

Front-end:

Since these outputs come from redirected command line outputs into STDIN, the absence of messages indicates that no errors or warnings have been found. Note that SonarQube has been disabled for React components since non-class-based components have a really high complexity rating that cannot be avoided.

Style checking using Prettier (doesn't provide any output besides a list of files violating styling conventions):

- View the report before making code changes here.
- View the report after making code changes here.

Linting using TSLint:

- View the report before making code changes here.
- View the report after making code changes here.

Bug checking using ESLint's SonarQube extension:

- View the report before making code changes here.
- View the report after making code changes here.

Back-end:

For style checking we switched from using pylint to pycodestyle to check if all the files in the upmed-api folder comply to pep8 code style guide. We switched from pylint to pycodestyle so that we can use pylint's error checking function capability. We consulted with our IA, on what is the best method for generating the reports automatically. The reports are generated on deploy from running the report.sh file in the upmed-api folder. Throughout the process I took advantage of autopep8 in PyCharm. Reports are stored in the reports folder in upmed-api

Style Checking before using pep8:

- View the report before making code changes here.
- View the report after making code changes here.

Bug Checking with pylint's error checking feature:

- View the report before making code changes here
- View the report after making code changes here
    - Note: in this report there are many issues related to imports and pylint unable to find the imports. This is an artifact of Pycharm and pylint unable to see the python file directory as a python project and each folder is not registered as a module that is importable. Below is an example of this.
-

```
import unittest
import requests
import time
from unittest.mock import MagicMock, Mock

from sys import path
from os.path import join, dirname
path.append(join(dirname(__file__), '../../..'))

from src import Database, Patient, Appointment, HCP, Day, Hours, Status, Auth, hcp, Twilio, hcp_signup, hcp_get_all,
from tst.constants import dummy_hcp, dummy_appointment, dummy_patient  # noqa
"""
HCP Endpoint Tests
"""
db = Database()
set_func = Mock()
set_func.to_dict = MagicMock(return_value=1)
set_func.set = MagicMock(return_value=1)
set_func.get = MagicMock(return_value=1)
set_func.stream = MagicMock(return_value=2)
set_func.delete = MagicMock(return_value=3)
db.document = MagicMock(return_value=set_func)
db.stream = MagicMock(return_value=2)

hcpdb = db.getHCP()
pat = db.getPatients()
appointmentsdb = db.getAppointments()
auth = Auth()
twilio = Twilio()

unittest.TestLoader.sortTestMethodsUsing = None
hcp_token = ''


class HCPTestCase(unittest.TestCase):
    def test_signup_test(self):
        week = []
        for i in range(0, 7):
            week.append(Day(startTime=540, endTime=1020))
        schedule = Hours(
```