0.1 Website

Selenum WebDriver will be used to run test suites written in Python, which also populate the database of the attendance management system and automate the data entry process, in Microsoft Edge Browser.

```python
def login():
    driver.find_element_by_id("email").send_keys("hcyyk1@nottingham.admin")
    driver.find_element_by_id("password").send_keys("123456")
    driver.find_element_by_id("login").click()
```

This function tests the login function on the landing page. The script enters email and password before clicking the login button to submit the user credentials.

```python
def student():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Student')))
    driver.find_element_by_id("Student").click()

    for x in range(10):
        driver.find_element_by_id("fName").send_keys(get_first_name())
        driver.find_element_by_id("lName").send_keys(get_last_name())

        programme = ['Computer Science', 'Business', 'Engineering']
        driver.find_element_by_id("programme").send_keys(random.choice(programme))

        driver.find_element_by_id("email").send_keys(random_char(7) + "@nottingham.edu.my")
        driver.find_element_by_id("password").send_keys("123456")

        driver.find_element_by_id("submit").click()
```

This function registers 10 student accounts. It waits for the student tab to become clickable before clicking it. The script will enter the first name, last name, choose the student's programme randomly, enter email and password, and submit all details, for 10 times.

```python
def lecturer():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Lecturer')))
    driver.find_element_by_id("Lecturer").click()

    for x in range(10):
        driver.find_element_by_id("LfName").send_keys(get_first_name())
        driver.find_element_by_id("LlName").send_keys(get_last_name())

        select_pos = Select(driver.find_element_by_id('position'))
        position = ['Assistant Professor', 'Associate Professor', 'Professor']
        select_pos.select_by_value(random.choice(position))

        select_fac = Select(driver.find_element_by_id('faculty'))
        faculty = ['Computer Science', 'Business', 'Engineering']
        select_fac.select_by_value(random.choice(faculty))

        module_set = ['Set1', 'Set2']
        driver.find_element_by_id("set").send_keys(random.choice(module_set))

        driver.find_element_by_id("Lemail").send_keys(random_char(7) + "@nottingham.edu.my")
        driver.find_element_by_id("Lpassword").send_keys("123456")

        driver.find_element_by_id("Lsubmit").click()
```

This function registers 10 lecturer accounts. It waits for the lecturer tab to become clickable before clicking it. The script will enter the first name, last name, choose the lecturer's position randomly choose the lecturer's faculty randomly, choose the lecturer's taught module set randomly, enter email and password, and submit all details, for 10 times.

```python
def module():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Module')))
    driver.find_element_by_id("Module").click()
    module_codes1 = ["COMP1001", "COMP1002", "COMP1003", "COMP1004"]
    module_codes2 = ["BSC1001", "BSC1002", "BSC1003", "BSC1004"]
    module_codes3 = ["ENG1001", "ENG1002", "ENG1003", "ENG1004"]
    module_names1 = ["Computer Fundamentals", "Software Engineering", "Discrete Maths", "Software Maintenance"]
    module_names2 = ["Business Communication", "Leadership", "Business Principles", "Business Writing"]
    module_names3 = ["Thermodynamics", "Natural Mechanics", "Electricity", "Computer Skills"]
    module_selector("Computer Science", module_names1, module_codes1)
    module_selector("Business", module_names2, module_codes2)
    module_selector("Engineering", module_names3, module_codes3)
```

This function adds modules to the database. It waits for the module tab to become clickable before clicking it. The code snippet above contains all the module codes and names that will be entered on the website. The *module_selector* function will send the module details to the database, as shown below.

```python
def module_selector(plan, modules, module_code):
    for n in range(4):
        driver.find_element_by_id("mID").send_keys(module_code[n])

        driver.find_element_by_id("mName").send_keys(modules[n])

        driver.find_element_by_id("mPlan").send_keys(plan)

        select_type = Select(driver.find_element_by_id('moduleType'))
        module_type = ['Core', 'Elective']
        select_type.select_by_value(module_type[0])

        module_set = ['Set1', 'Set2']
        driver.find_element_by_id("mSet").send_keys(random.choice(module_set))

        driver.find_element_by_id("Add").click()
```

The function above sends the module id, module name, the programme of the module (plan), chooses the module type for the students (in this case all modules are 'Core', choose s the module set for the lecturers, and submit all the details.

```python
def timetable():
    date = 20210401
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Timetable')))
    driver.find_element_by_id("Timetable").click()
    module_codes1 = ["COMP1001", "COMP1002", "COMP1003", "COMP1004"]
    module_codes2 = ["BSC1001", "BSC1002", "BSC1003", "BSC1004"]
    module_codes3 = ["ENG1001", "ENG1002", "ENG1003", "ENG1004"]
    for x in range(5):
        timetable_selector(module_codes1, date)
        timetable_selector(module_codes2, date)
        timetable_selector(module_codes3, date)
        date += 1
```

This function updates the timetable on the database. It waits for the timetable tab to become clickable before clicking it. The code snippet above contains all the module codes that represents. The *module_selector* function will send the module details to the database, as shown below, for 5 days: April 1st 2021 to April 5th 2021.

```python
def timetable_selector(codes, date):
    for x in range(4):
        date_entry = str(date)
        driver.find_element_by_id("date").send_keys(date_entry)

        driver.find_element_by_id("module_id").send_keys(codes[x])

        session_type = ["Lecture", "Tutorial", "Lab"]
        driver.find_element_by_id("sessionType").send_keys(random.choice(
    session_type))

        driver.find_element_by_id("status").send_keys("Closed")

        if x < 2:
            time = ["0900", "1100", "1300"]
            driver.find_element_by_id("time_begin").send_keys(time[x])
            driver.find_element_by_id("time_end").send_keys(time[x + 1])
        else:
            time = ["1400", "1600", "1800"]
            driver.find_element_by_id("time_begin").send_keys(time[x - 2])
            driver.find_element_by_id("time_end").send_keys(time[x + 1 - 2])

        driver.find_element_by_id("Tsubmit").click()
        x += 1
```

This function enters the date, module id, chooses session type randomly, session status, time of each sessions, 4 in a day (0900-1100, 1100-1300, 1400-1600, 1600-1800) and submit the details.

```python
def logout():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'logout'
    )))
    driver.find_element_by_id("logout").click()
```

This function logs out the user from the website, after clicking the 'Logout' buton.

0.2 Application

Application testing was conducted in two different ways, testing via emulators and manual device testing using real devices. For functionalities such as the UI functionality and fetching from the Firebase Database, testing was done using emulators provided by Android Studio. For the UI, a test suite was created where each possible user action and its corresponding application response was tested. For database functionality, manual testing was conducted whereby data was inputted into the database (via the website), and the applications ability to fetch data and populate the UI was tested.

Regarding the testing of the attendance taking functionality, real life testing was conducted using a minimum of 2 Android devices. Attendance taking relies on Android background services and Bluetooth Low Energy connections between devices. Real life testing was conducted because Bluetooth is not available on emulators so physical devices must be used.

Regarding Android background services, testing was done by monitoring notifications posted by the app to the user. If the specific notification for the service was posted, the app's background services were functioning normally. A test suite was created for the scheduled background services where we monitored notifications to determine whether tests passed or failed. We tested whether the services started within the correct time frame and whether subsequent services were started.

A test suite was also created for app's Bluetooth functionality. There were some issues that occurred during testing. To monitor connections using Bluetooth Low Energy, Toast messages were used to monitor the connection state, devices scanned, and Bluetooth services discovered. However, Toast messages were an unreliable method for monitoring tests as not all Toast messages were displayed despite the app functioning properly. To solve this issue, tests were instead monitored using changes to the database. All Toast messages used for testing were converted to updates/writes to the database. By monitoring the database, we could see whether the Bluetooth functionality was working as expected.