Doğukan Ertunga Kurnaz
21702331
Section 1

<u>Answers:</u>

Part 1.

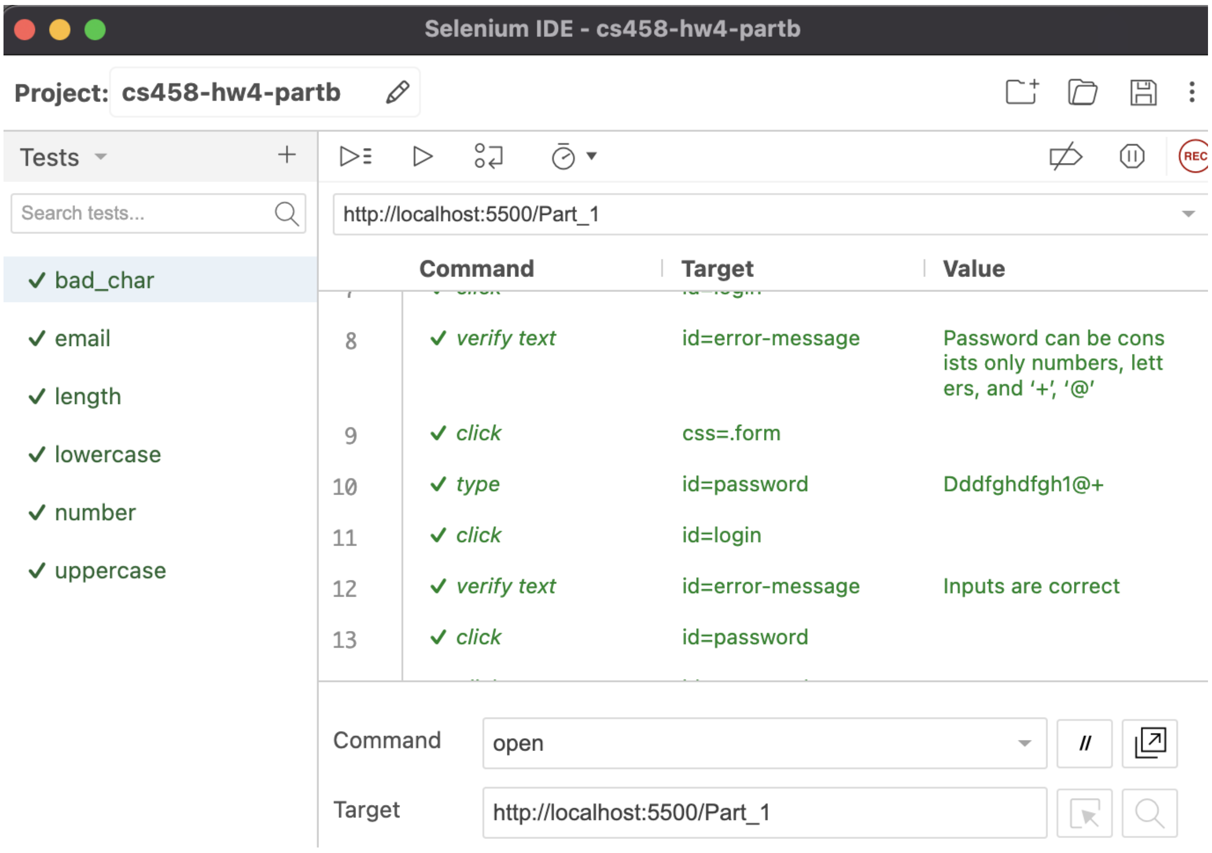
a. In this part, I used HTML, JS, and CSS to build the required website. This revision of the website has no API requests, and its files stay in the Part_1 folder.

b. It was easy to use for my recorded test cases in Selenium IDE to me. Especially the recording button saved me a lot of time because I did not have to write all the div names and values manually.

To evaluate the functionality of my website I used those inputs:

Email	Password	isEmailValid	isPasswordValid
a@a.	DDDD@@1sdgds	No	Yes
test@test.com	1234	Yes	No
test@test.com	D@DYY1234	Yes	No
test@test.com	d@dyy1234	Yes	No
test@test.com	mysecretP@ss	Yes	No
test@test.com	!!!!!!!!!!!!!!Dd1	Yes	No
test@test.com	+@dsfwsfD1	Yes	Yes

To recreate the test suit in your machine, you can simply start a live server which server index.html in the Part 1 folder. Then import the Selenium file into your IDE and run all the tests. For PoC, I added a screenshot below:


Selenium IDE - cs458-hw4-partb

Project: cs458-hw4-partb

Tests

- ✓ bad_char
- ✓ email
- ✓ length
- ✓ lowercase
- ✓ number
- ✓ uppercase

http://localhost:5500/Part_1

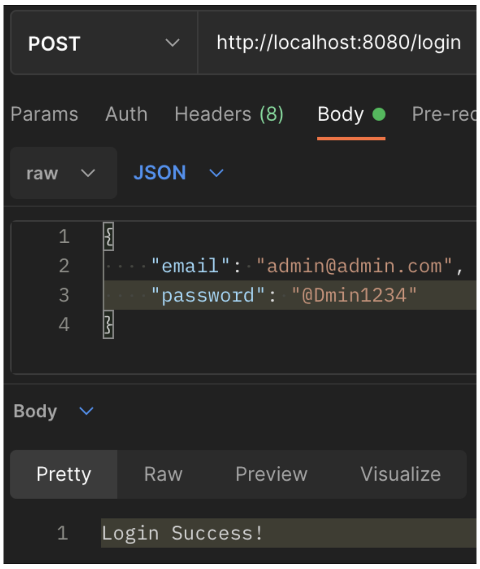
	Command	Target	Value
8	✓ verify text	id=error-message	Password can be consists only numbers, letters, and '+', '@'
9	✓ click	css=.form	
10	✓ type	id=password	Dddfghdfgh1@+
11	✓ click	id=login	
12	✓ verify text	id=error-message	Inputs are correct
13	✓ click	id=password	

Command: open

Target: http://localhost:5500/Part_1

Part 2.

In Part 2, using Wiremock standalone was a bit tricky. Firstly, I add a new feature for posting requests to my mocked API and if the response was successful, the website informs us. After completing the website part, the most challenging stuff starts. I mocked a login API using Wiremock CLI. To not specify its rules every time I created a pre-defined mapping and it can load up automatically in the startup of wiremock. Since I put my data in the request body in JSON format, I somehow know that the first thing should be parsing the body. To achieve this I used `bodyPatterns` and `matchesJsonPath` methods of Wiremock to extract and compare the request data with my rule. If the comparison is successful, `"Login Success!"` message will be sent to the client. I used Postman to test my mocked API.

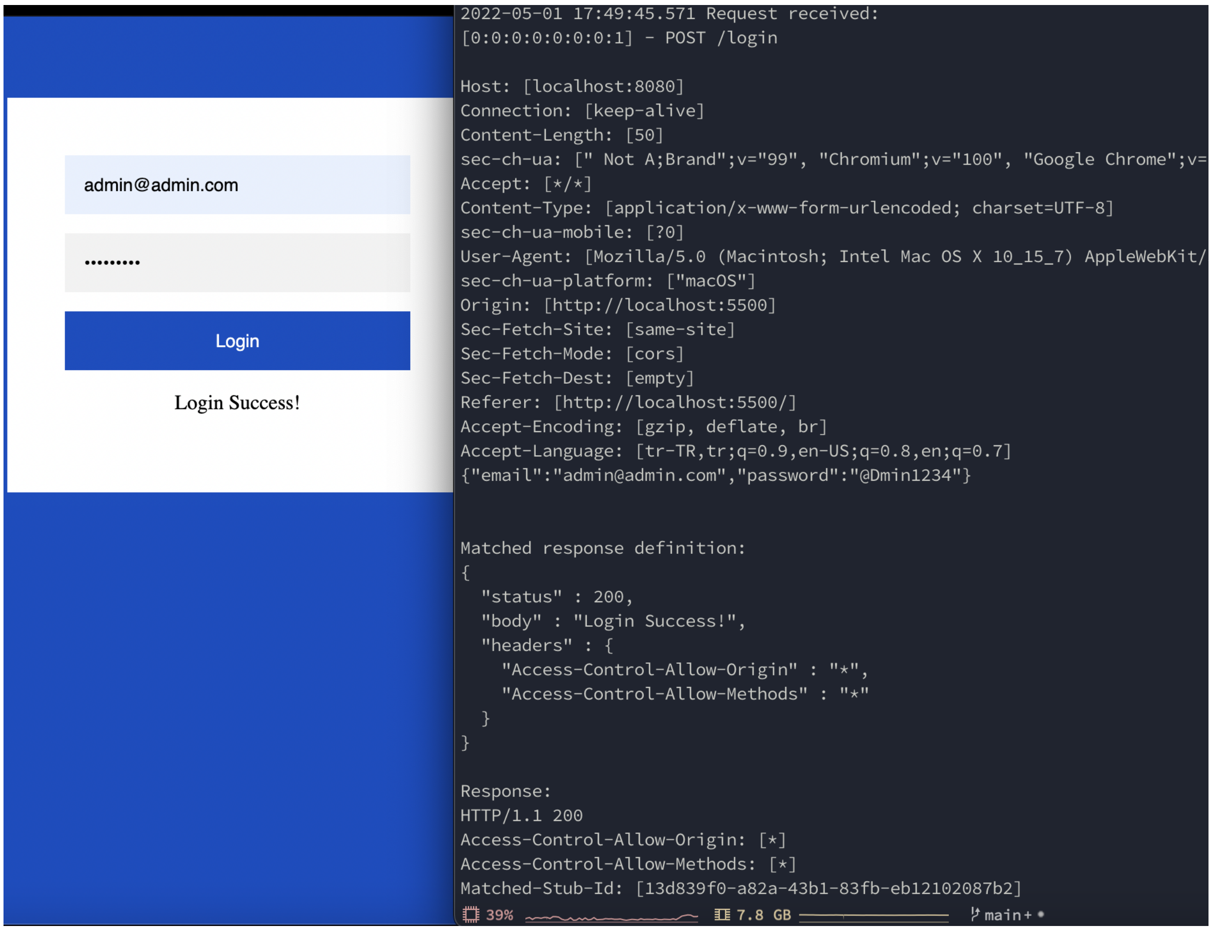


http://localhost:8080/login

```
{
····"email":·"admin@admin.com",
····"password":·"@Dmin1234"
}
```

```
Login Success!
```

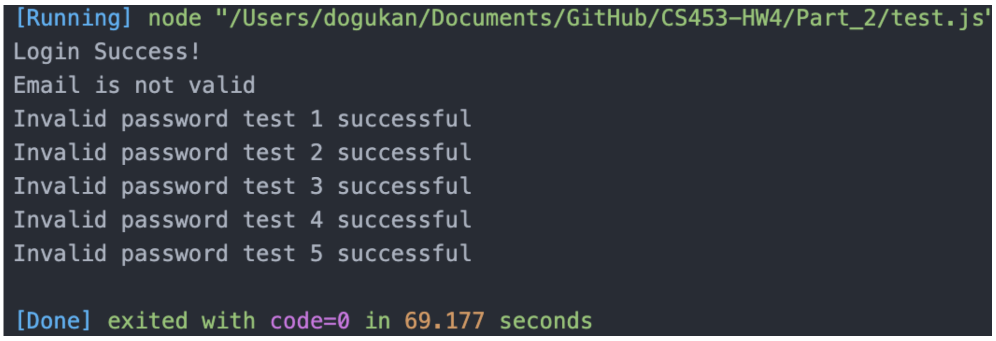
This mapping file can be found in Part 2's folder. Also, you can find the required Selenium Chrome driver and standalone version of Wiremock in the folder as well. After connecting my original website to my API, an example request and response can be shown in the screenshot below:



```
2022-05-01 17:49:45.571 Request received:
[0:0:0:0:0:0:0:1] - POST /login

Host: [localhost:8080]
Connection: [keep-alive]
Content-Length: [50]
sec-ch-ua: [" Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v=
Accept: [*/*]
Content-Type: [application/x-www-form-urlencoded; charset=UTF-8]
sec-ch-ua-mobile: [?0]
User-Agent: [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/
sec-ch-ua-platform: ["macOS"]
Origin: [http://localhost:5500]
Sec-Fetch-Site: [same-site]
Sec-Fetch-Mode: [cors]
Sec-Fetch-Dest: [empty]
Referer: [http://localhost:5500/]
Accept-Encoding: [gzip, deflate, br]
Accept-Language: [tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7]
{"email":"admin@admin.com","password":"@Dmin1234"}


Matched response definition:
{
  "status" : 200,
  "body" : "Login Success!",
  "headers" : {
    "Access-Control-Allow-Origin" : "*",
    "Access-Control-Allow-Methods" : "*"
  }
}

Response:
HTTP/1.1 200
Access-Control-Allow-Origin: [*]
Access-Control-Allow-Methods: [*]
Matched-Stub-Id: [13d839f0-a82a-43b1-83fb-eb12102087b2]
```

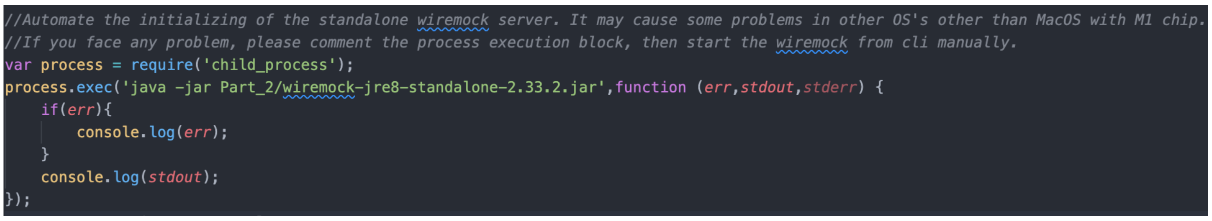
After being sure everything is working, I started to write tests with Selenium Programming API in JavaScript. First I downloaded the required chromedriver and selenium-webdriver.

They can be installed by yarn or they can be installed manually. For chromedriver I installed it manually, for selenium-webdriver I used yarn. After this part, I started to write Selenium tests, for the cases to test my API. I couldn't get why we need the Mocha framework in this part since our website already uses API calls. That's why I didn't use Mocha. Its requirements should be more clear. Writing test cases did not take that much time thanks to nice documentation on the Selenium Help page. After running the test, you should get a console output like that:



```
[Running] node "/Users/dogukan/Documents/GitHub/CS453-HW4/Part_2/test.js'
Login Success!
Email is not valid
Invalid password test 1 successful
Invalid password test 2 successful
Invalid password test 3 successful
Invalid password test 4 successful
Invalid password test 5 successful

[Done] exited with code=0 in 69.177 seconds
```

To achieve the full automation with one click, I decided to integrate the initial startup command of wiremock into the code as well. Before this step, tests were automated but creating the environment was a manual process because we were starting the wiremock manually by using CLI. I added the following lines to my js code to automate this process:



```
//Automate the initializing of the standalone wiremock server. It may cause some problems in other OS's other than MacOS with M1 chip.
//If you face any problem, please comment the process execution block, then start the wiremock from cli manually.
var process = require('child_process');
process.exec('java -jar Part_2/wiremock-jre8-standalone-2.33.2.jar',function (err,stdout,stderr) {
    if(err){
        console.log(err);
    }
    console.log(stdout);
});
```

With this implementation, *test.js* can start wiremock with the correct mapping and it can drive a chromium browser with Selenium's testing capabilities.