Politecnico di Milano
AA 2017-2018

Software Engineering Project

RAVLENDAR+

ATD

AcceptanceTest Document

Version 1.0

Marco Prosdocimi - 898395
Enrico Ruggiano - 900040
Giacomo Vercesi - 899928

Contents

Table of Contents

depth 2

Introduction

This document present all the informations about the Acceptance Test of Travlendar+.

analyzed project

AldeghiKrasniqiMazzoleni project.

link to project folder: https://github.com/filipkrasniqi/AldeghiKrasniqiMazzoleni.

Commit info

- commit message BEERS WITH THE BOyS
- commit hash: 399e84146fb5c53cd323c0f44ff95edb67ab4a61.
- Author: Filip Krasniqi <filip.krasniqi@mail.polimi.it>.
- Date: Sun Jan 7 23:59:28 2018 +0100.

All other commit posted after the last date of submission were ignored. All above information were obtained with "git log" command.

Installation and Setup

The installation procedure was accomplished in the following ways:

Client Side

The client side installation procedure was done by using the provided apk inside `DeliveryFolder/Implementation`, which was not accounted for in the ITD document. The apk was installed on a variety of devices using Android Virtual Device Manager.

Server Side

For consistency's sake the we decided to utilize docker to deploy the server component. Docker allowed us to have consistent results while we were testing the server on our computers. The war file used was the one provided in `DeliveryFolder/Implementation`, The installation procedure that we followed is as follows:

```
cp <Repo Folder>/DeliveryFolder/Implementation/web.war .
sudo ./build.sh
sudo docker-compose up
```

the rest of the testing was done using the local server. One detail that was not explicitly specifiied in the ITD document was that the mysql server's ip was hardcoded to `localhost`, something that revealed itself to be cumbersome in our docker setup and it would cause major issues if it were used in the setup as shown in the RASD (3.4.3).

Acceptance Test Cases

Documents inconsistencies

Comparing the RASD, DD and ITD document with the code we found those inconsistencies:

- There are not motivation on why Travlendar+ has to present to the user a disclamer for the use of personal information. Even external information provided from a facebook or external login are not specified what will need for. There are not specified requirements or algorithm which will not work if those informations are not inserted.(see RASD 2.4-Disclamer Approvance)
- There is not a clear model of the data structure on the RASD and the DD. How the data are managed by the database is not clearly written in any document. For example are not specified the tables used and the organization of the information on the different database, the local REALM and the remote MYSQL (see RASD and DD).
- Is not a 'Thin Client' a client which has implemented a local database synchronized with a server. Also the app takes 61,94 MB on a Android 6.0 device, which is not so 'Thin'(see DD 2.7 and ITD 4-client_side).

- Is not specified why it is required a forced synchronization of the client data to server data even if the client can't work properly without internet connection (and so without server ednpoint connection). (see ITD 4-client_side).
- Is not specified why a PHP server implementation was replaced by a JEE server with glassfish. (see RASD 3.1.3).
- Is not clear the algotithm design and with which vehicle can calculate the best path. Also the City Bike, Inrix, Car2Go API are not implemented and their neglect is not mentioned in ITD document. (see RASD 2.1)

Analysis of Functionalities

For our analysis we used the following environment.

- We started in local the Server using the file Web.war
- We used mitmproxy, a proxy server that allows us to see the api calls and the json files exchanged between the server and the client.
- We tried some actions provided by the application and register the information exchanged by the client and the server.
- We tried to reproduce the Post calls with jmeter, we have stored a log of our test in the file AcceptanceTest.jmx.

In the following section we report our results of test of the functionalities with some presumed bugs and jmeter test reference.

Registration

jmeter test reference: Registration

The registration password check does not work properly. The application asks a password that "Contain at least 8 characters, one lower case character, one upper case character, one number and one special character" but if we set this type of password the server refuses the POST. The relative jmeter test fail and return error 400.

The most probable cause of this is that the server-side regex responsible for this is not the correct one.

Actual regular expression:

```
((?=.*\\\\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[.:-_,;*+\\[\\]@!\"&/()=?#$%\\\\\]).{8})
```

Login

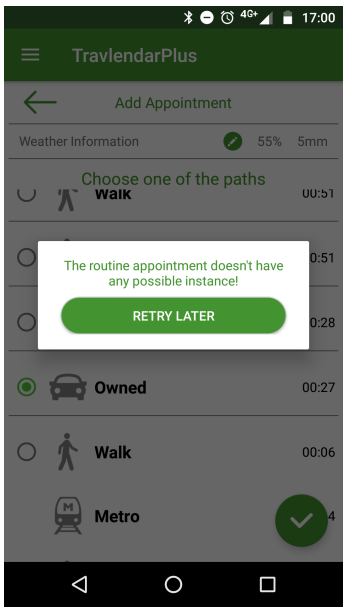
jmeter test reference: Login

We can't perform a login test without a registration. The test returns an error 400.

Submit Standard Event

jmeter test reference: Event

we noticed a strange bug in the app. when we submit the event the server answers with the possible paths list but when we try to select a path some times we see this screen:



The post as can see in jmeter test report the message: `"error": "The routine appointment doesn't have any possible instance!"`

Submit Flexible Event

The flexible lunch seems not working properly.

The Server answer at the post is: `"error": "The minReservationTime is greater than the allowed timeslot"`

We find this bug in the code that check the reservation time:

The code perform this: `if(minReservationTime > start.getTimestamp() - end.getTimestamp())`

Instead of this: `if(minReservationTime > end.getTimestamp() - start.getTimestamp())`

Submit Preference

jmeter test reference: Thread group Preference

The submit of the preference to the server works fine.

note: every time the user selects a preference the app sends a Put Request to the server. To limit the JSON traffic between Client and Server the application should present a submit button which if it is pressed, it triggers an unique post to the server.

Malformed post

jmeter test reference: Malformed Post

We tried to use Jmeter to send to the server some malformed post requests. for istance: with some null or missing fields. Those tests verified the robustness of the server and that the group spent a lot of time implementing checks functions, Enum types and extended

exceptions. This is not true for the client which is completely vulnerable and it relies heavily on what server sends to him. (see Security Issues).

Some exemple of this tests are in jmeter

Other test Case

- When submitting an event the "Customize" button does not overwrite the "Global Travel Preferences" or any personal preference category chosen before.
- Weather information is always the same.
- Vehicles not working: Car Sharing, Bike, Bike Sharing, Taxi.
- Vehicle working: Car Owned, Walk.
- Public transport: You cannot force a computation of the path with specified transport, because Google considers them only when their ETA is competitive respect the others vehicles. The preference setting on pubblic transport is just a filter on what google passes to the app.
- Every time I click the 'Server IP' button and confirm on the server box, The App receive always different "restore password" even if the server ip is not changed.
- json traffic is really high and reduces the performance. A lot of data is redundant and not self-explanatory. Also a json message for submit an event is about 2,6kB, which is not lightweight (see jsonMessage.json)

Security issues

in these part we present all the security issues found and a small guide to prove their existence.

Corrupt event submit

The client does not control the validility of the server's json responses. It is possible to man in the middle attack which is triggered when client submits an event and it sends back a corrupted response with invalid data.

for these guide we used `mitmproxy`, `mitmweb` `pathod`

`install mitmproxy. see guide on http://docs.mitmproxy.org/en/stable/install.html`

run the server war locally. In our example is setted on `localhost:8080`

open a terminal and run:

```
mitmweb -p 1234 -R http://localhost:8080
```

you can use `mitmproxy` if you like. With these command we will have the proxy server up on 1234 port which will catch all the http messages and it will redirect them to localhost on port 8080.

open android application. go to "Server Ip" and put the IP address of the machine wehere the proxy server runs and port 1234. for example

`192.168.0.1:1234`

now on the app try to submit an event. click first button, click second button (filling all the textfield with valid data).

on the path page don't click the button.

stop the proxy server and open a terminal and run

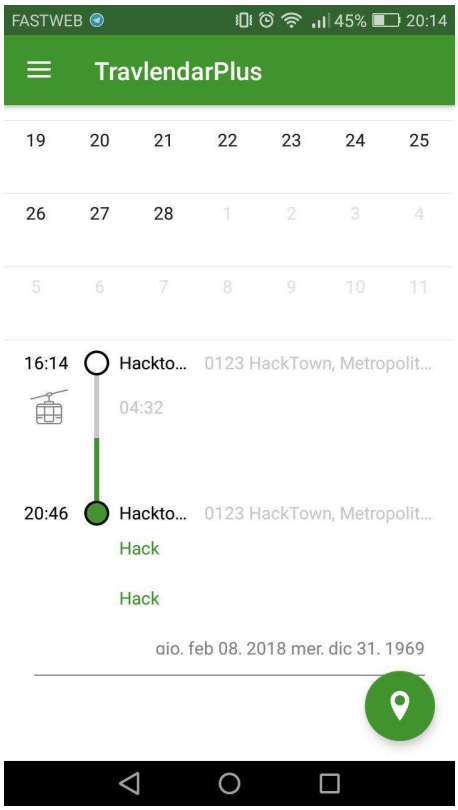
```
pathod -p 7070 -d ~/PATH/TO/JSON \
-a '/web/v1/user/appointment=200:c"application/json":b<invalid_response'
```

where `~/PATH/TO/JSON` is the path in which you have the `invalid_response` file which contains the json response. The json is provided to you in this repository. pathod now will run on the port 7070 and when it will receive an http request with /web/v1/user/appointment url will send back a 200 message with content type `application/json` with the `invalid_response` content as body.

now run

```
mitmweb -p 1234 -R http://localhost:7070
```

and click the third button of the app. On agenda you will see your invalid event submitted correctly!



Deleting Event

This bug involves data structure of travlendar plus project. All Events submitted are identified with an incremental Integer id chosen by the server. It is probably linked to the creation of data record on the mysql db. With a simply spoof of the access token it

is possible to delete a vast range of data simply with a for loop and a curl command. In these cases it is not only possible to delete user's events, but all users' events.

for these guide we used `mitmproxy`, `mitmweb` `curl`

setup the same above configuration.

run war server locally on `localhost:8080` in these guide we used a docker container.

open a terminal and run:

```
mitmweb -p 1234 -R http://localhost:8080
```

you can use `mitmproxy` if you like. With these command we will have the proxy server up on 1234 port which will catch all the http messages and it will redirect them to localhost on port 8080.

open android application. go to "Server Ip" and put the IP address of the machine wehere the proxy server runs and port 1234. for example

```
192.168.0.1:1234
```

make some Post request on the application. Go to the mitmweb page and copy the `Authorization` header. In these example

```
Authorization: Bearer 50ab7dc9-19d7-49b5-ac5f-08e210129d76
```

important all the token are "unique" so it is important to verify the header every time.

now open a terminal and run

```
for i in `seq 1 9999`; do
        curl -i -H "Authorization: Bearer 50ab7dc9-19d7-49b5-ac5f-08e210129d76" \
         -H "Content-Type: application/json" \
         -X DELETE  http://localhost:8080/web/v1/user/appointment/$i
done
```

remember to paste after the `-H` the session token discovered before.

And now all events with id between 1 and 9999 have been deleted. See the attached `delete.sh` file for additional information.

Other notes

- No Java Packages used.
- While the code features well-documented functions, there is no compiled javadoc html folder with them.
- APK installer says that no privilege are needed by the app, however at first login you need to accept their policies. No Android manifest is set.
- we were unable to build the server component via `mvn package`, as some of the required libraries were missing from the `pom.xml`.
- The use of a backtracking-regex for password checking was unnecessary, as a much simpler and flexible checkPassword could have been implemented with a for loop.