CmpE 436

Fall 2019

Final Project

Cinema Reservator

Gurkan Demir

2015400177

December 9, 2019

Contents

1 Abstract **3**

2 Introduction **4**

3 Approach & Methodology **5**

3.1 Diagram 5

3.2 Server Side 6

3.2.1 Waits for Request 6

3.2.2 Handle Request 6

3.2.3 Controllers 7

3.3 Client Side 13

4 Demonstration & Experiments **17**

4.1 Login & Signup 17

4.2 Saloons 18

4.3 Films 19

4.4 Seat Details 20

4.5 Reservation 21

4.6 My Tickets 22

5 Conclusion **22**

6 References **23**

1 Abstract

CinemaReservator is an application where users make reservation for desired film in selected saloon. There will be a list (on Android app) of all the cinemas available right now. Current cinemas can be added to database manually. When a client clicks on seat whose status is EMPTY to make reservation, then the client acquires lock from a binary semaphore on the server side and seat's status become IN_PROGRESS. From then on, user holds the lock for 1 minute. At the end of 1 minute, if user does not complete reservation, user's lock expires and seat's status become EMPTY. If user completes, the seat's status become FULL. Each saloon has fixed size number of seats(40). Users choose saloon, film and seat in order to make reservation. Assuming clients have infinite amount of money, there does not exist any payment methodology in this application. Project is implemented only in Android, using Java written server with socket programming.

The locking logic applies when a client wants to make reservation for seat. Also multi-client logic applies when more than one client can try to reach that limited resource. Race condition can be occurred while choosing same seat by different users. Source codes for both server and client side, and APK of my project can be found on my personal Github account.

2 Introduction

In the term project of CmpE436, we are expected to implement multi threaded, locking based Android application using socket programming. System must support multi client and there must not exist any race condition in application. Both server side and client side of application must be written in Java, and server must be deployed using web services like Amazon. So we need to find an idea about project which supports all things that I encountered above.

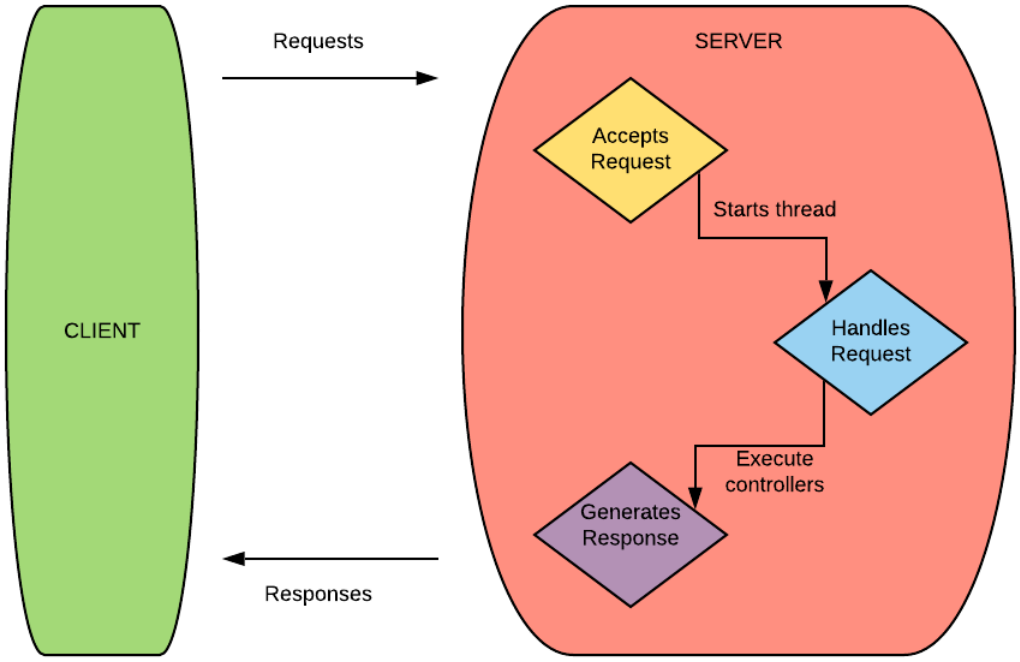
As an individual, who likes watching films in cinema centers and always makes reservation from online, I use some mobile applications to make reservation. So I decided to implement my own Android application where users make reservation for desired films in selected saloon.

At first I tried to get available films from SineBU's monthly schedule, but due to the fact that I have no idea about HTML crawling, I decided get film using 3rd party API. However, since there exists no 3rd party API which provides available films for each cinema saloon, I decided to add films to database manually.

The details and usage of application are explained in the following sections.

3 Approach & Methodology

3.1 Diagram



As I explained above, project is implemented using socket programming. Client side sends requests for different purposes. Server side always waits for a request, and as soon as request achieved, it generates a new thread in order to handle that request. Request is a string in JSON format, and in each request there exists a field named **function** which specifies type of request. According to function, server

side calls related controller, and controller generates a response which is a string in JSON format and transmits to client.

3.2 Server Side

3.2.1 Waits for Request

```
@Override
public void run() {
    while (true) {
        try {
            Socket connectionSocket = socket.accept();
            new Thread(new ConnectionHandler(connectionSocket)).start();
        } catch (IOException ex) {
            System.err.println("Server aborted:" + ex);
        }
    }
}
```

Server side runs on host 54.245.129.91, port 8080. In a while loop, it waits for a request to come. Whenever a request comes, it accepts it and creates a thread to handle that request.

3.2.2 Handle Request

After creating a new thread when a request comes, it calls **handleRequest** method in order to generate output. This method extracts **function** from JSON, and according to keyword calls necessary controllers.

```
public void handleRequest(JSONObject request) throws JSONException, SQLException {
    String function = request.getString( key: "function");
    if(function.equals("signup")){
        signUp(request);
    }
    else if (function.equals("login")) {
        login(request);
    }
    else if(function.equals("showTickets")) {
        showTickets(request);
    }
    else if(function.equals("search")){
        search(request);
    }
    else if(function.equals("reserve")){
        reserve(request);
    }
    else {
        unhandledFunction();
    }
}
```

3.2.3 Controllers

- **Login Controller**

  When keyword is **login**, it means client wants to login to system and then login controller is called. In this controller, **username** and **password** fields are extracted from JSON, and checks the USER table whether there exists a entry with given username and password. If so, it returns success otherwise it returns error.

  Example Request: {”function”: ”login”, ”username”: ”gurkandemir”, ”pass-

word”: ”qwerty”}

Example Response: {”status”:”200”,”username”:”gurkandemir”}

- **Signup Controller**

  When keyword is **signup**, it means client wants to signup to system and then signup controller is called. In this controller, **username** and **password** fields are extracted from JSON, and checks the USER table whether there exists a entry with given username. If so, it returns error otherwise it creates a new entry in USER table with that username and password. Before those controls, system must acquire a lock which is binary semaphore for database. After operation, system releases that lock.

  Example Request: {”function”: ”signup”, ”username”: ”test-user”, ”password”: ”test-password”}

  Example Response: {”status”:”200”,”username”:”test-user”}

- **Search Controller** Client can search for saloons, films, and seats. According to **detail** field in JSON, it calls necessary function in that controller.

  - **Saloons**

    When function is search and detail is saloon, it means client wants to get available saloons and **searchSaloons** method in search controller

is called. It checks SALOON table, and returns all saloons with their id, name, and location.

Example Request: {"function": "search", "detail": "saloon"}

Example Response: {"saloons":["name":"Cinemaximum","location":"Kanyon AVM","id":1],"status":"200"}

- **Films**

  When function is search and detail is film, it means client wants to get available films in selected saloon and **searchFilms** method in search controller is called. It extracts **saloon** field from JSON, which specifies selected saloon id and then checks FILM table with given saloon id, and returns all films with their id, name, director, and date.

  Example Request: {"function": "search", "detail": "film", "saloon": 1}

  Example Response: {"films":["date":"18.00","director":"Ozer Feyzioglu","name":"Cep Herkulu: Naim Suleymanoglu","id":1,"date":"19.30","director":"Roland Emmerich","name":"Midway","id":2,"date":"21.00","director":"Bong Joon Ho","name":"Parasite","id":3],"status":"200"}

- **Seats**

  When function is search and detail is seat, it means client wants to get

seat details in selected film and **searchSeats** method in search controller is called. It extracts **film** field from JSON, which specifies selected film id and then checks SEAT table with given film id, and returns seat details with their id, and availability.

Example Request: {”function”: ”search”, ”detail”: ”seat”, ”film”: ”2”}

Example Response: {”seats”:[”available”:0,”id”:40,”available”:0,”id”:41, ”available”:0,”id”:42,”available”:1,”id”:43,”available”:0,”id”:44,”available”:0,”id”:45, ”available”:0,”id”:46,”available”:0,”id”:47,”available”:0,”id”:48,”available”:0,”id”:49, ”available”:1,”id”:50,”available”:0,”id”:51,”available”:0,”id”:52,”available”:0,”id”:53, ”available”:0,”id”:54,”available”:0,”id”:55,”available”:0,”id”:56,”available”:0,”id”:57, ”available”:0,”id”:58,”available”:0,”id”:59,”available”:1,”id”:60,”available”:0,”id”:61, ”available”:0,”id”:62,”available”:0,”id”:63,”available”:1,”id”:64,”available”:0,”id”:65, ”available”:0,”id”:66,”available”:0,”id”:67,”available”:0,”id”:68,”available”:0,”id”:69, ”available”:0,”id”:70,”available”:0,”id”:71,”available”:0,”id”:72,”available”:0,”id”:73, ”available”:1,”id”:74,”available”:0,”id”:75,”available”:1,”id”:76,”available”:0,”id”:77, ”available”:0,”id”:78,”available”:0,”id”:79],”status”:”200”}

- **Reservation Controller** Client can start and complete reservation. According to **detail** field in JSON, it calls necessary function in that controller.

– **Start**

When function is reserve and detail is start, it means client wants to start reservation for selected seat and **reserveStart** method in reservation controller is called. It extracts **seat** and **username** fields from JSON, which specifies selected seat id, and username. In order to prevent multiple actions on same seat at the same time, for each seat, there exists a binary semaphore which is initially *True*. So in order to start reservation, client must acquire lock of that seat, after acquiring it checks whether seat status is still *EMPTY* or not, if not it releases lock and can not start reservation. If it it still *EMPTY*, it makes seat's status *IN_PROGRESS*, and updates SEAT table with given username. Then it releases lock of that seat.

Example Request: {"function": "reserve", "detail": "start", "seat": "2", "username": "gurkandemir"}

Example Response: {"status":"200"}

– **Complete**

When function is reserve and detail is complete, it means client wants to complete reservation for selected seat and **reserveComplete** method in reservation controller is called. It extracts **seat**, **username** and **email**

fields from JSON, which specifies selected seat id, username, user's email. In order to complete reservation, client must acquire lock of that seat, after acquiring it checks whether reservation of that seat is started from that user and whether seat status is still *IN_PROGRESS* or not, if not it releases lock and can not complete reservation. If all conditions are satisfied, it makes seat's status *FULL*, and creates new entry in RESERVATION table for that seat, username, user email. Then it releases lock of that seat.

Example Request: {"function": "reserve", "detail": "complete", "seat": "2", "username": "gurkandemir", "email": "ggurkandemir@gmail.com"}

Example Response: {"status":"200"}

- **MyTickets Controller**

  When keyword is **showTickets**, it means client wants to see user's previously reserved tickets and then tickets controller is called. In this controller, **username** field is extracted from JSON, and checks the RESERVATION table with given username. Then it returns all previously reserved tickets with their seat id, film name, film date and time, saloon name.

  Example Request: {"function": "showTickets", "username": "gurkandemir"}

  Example Response: {"tickets":["seat":2,"date":"2019-12-09","saloon":"Cinemaximum",

”film”:”Cep Herkulu: Naim Suleymanoglu”,”time”:”18.00”],”status”:”200”}

3.3 Client Side

- **Login Activity**

  In Login Activity, there exist 2 input texts one for username and other one for password, and 2 buttons one for login and another for signup. After clicking on any button, it checks whether inputs are empty or not. If at least one of them is empty, system warns users to fill username and password.

  After that controls, system executes **AsyncTask**. In this task, it makes initial configurations like opening a socket with server's host and port, and constructs a request in a JSON format. If login button is clicked, **function** field in request is **login**, else if signup button is clicked, field in request is **signup**.

  Then it requests to server, and waits for response. According to response, if it is success, users are redirected to saloon fragment, otherwise users are kept in the login activity until providing true credentials.

- **Saloon Fragment**

  When user in saloon fragment, initially system creates a new thread in order to get available saloons from server side. After creating a thread, main thread

waits for that thread to execute its operation.

In newly generated thread, first it makes initial configurations like opening a socket, constructing a reader and writer etc. Then it constructs a request in JSON format with **function = search** and **detail = saloon**.

Then it waits for response from server, according to response, it fills available saloons.

- **Film Activity**

  When user in film activity, initially system creates a new thread in order to get available films in selected saloon from server side. After creating a thread, main thread waits for that thread to execute its operation.

  In newly generated thread, first it makes initial configurations like opening a socket. Then it constructs a request in JSON format with **function = search** and **detail = film**, **saloon** = selected saloon id.

  Then it waits for response from server, according to response, it fills available films.

- **Seat Detail Activity**

  When user in seat detail activity, initially system creates a new thread in order to get seat details of selected film from server side. After creating a

thread, main thread waits for that thread to execute its operation.

In newly generated thread, first it makes initial configurations like opening a socket. Then it constructs a request in JSON format with **function = search** and **detail = seat**, **film** = selected film id.

Then it waits for response from server, according to response, it fills seats with related color. If seat is available, seat is colored green otherwise is it red.

In order to start reservation, users must click on one of the green seats. When user clicks on any green seat, system executes **AsyncTask**. In this task, after initial configurations, it constructs a request in JSON format with **function = reserve**, **detail = start**, **seat** = selected seat id, **username** = username.

Then it waits for response from server side. If response is success, users redirect to reservation activity, otherwise it user is kept in this activity until choosing a valid seat.

- **Reservation Activity**

  In reservation activity, there exist a input text for email, and a button for completing reservation. Also, there exists a countdown timer which starts from 60 seconds, which means users have 1 minute complete reservation. If users do not complete it in given time, users are redirected to seat details

page. After clicking on button, it checks whether input is empty or not. If empty, system warns users to fill email.

After that controls, system executes **AsyncTask**. In this task, it makes initial configurations like opening a socket with server's host and port, and constructs a request in a JSON format with **function** = **reserve**, **detail** = **complete**, **seat** = selected seat id, **username** = username, **email** = email.

Then it requests to server, and waits for response. According to response, if it is success, users are redirect to saloon fragment, otherwise users are redirected seat details page.

- **Tickets Fragment**

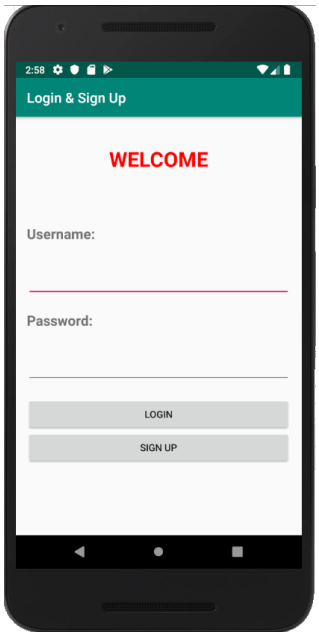
  When user in tickets fragment, initially system creates a new thread in order to get previously reserved tickets from server side. After creating a thread, main thread waits for that thread to execute its operation.

  In newly generated thread, first it makes initial configurations like opening a socket, constructing a reader and writer etc. Then it constructs a request in JSON format with **function** = **showTickets** and **username** = username.

  Then it waits for response from server, according to response, it fills available tickets.

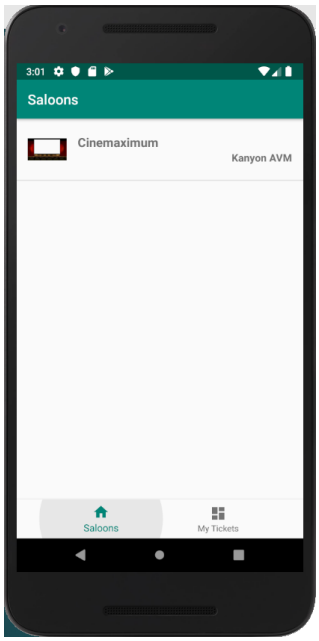
4 Demonstration & Experiments

4.1 Login & Signup



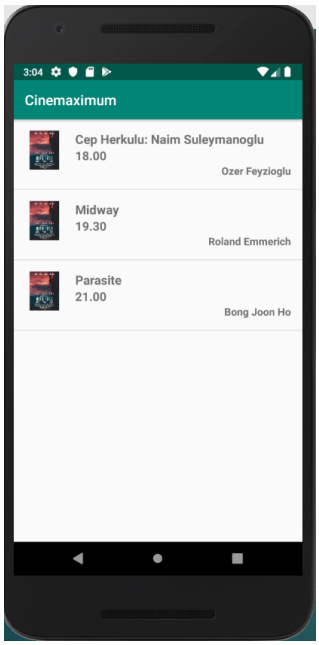
- Users must be logged into system in order to make reservation.
- Users can login by providing their username and password.
- If user has no account yet, user can signup by providing their username and password.
- After successful login, users are redirected to saloons page.

4.2 Saloons



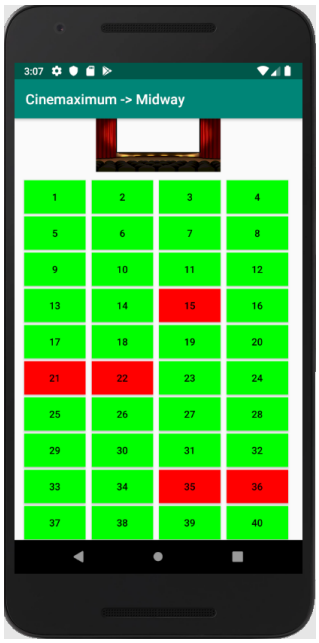
- Available saloons are listed with their location in this page.
- In order to see films in listed saloons, users must click on that saloon name.
- After clicking on saloon name, users are redirected to films page.

4.3 Films



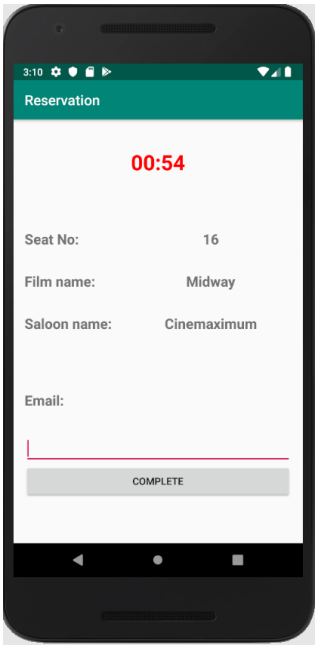
- Available films in selected saloon are listed in this page.
- Films are listed with their director, and time.
- In order to see seat details of any film, users must click on film name.
- After clicking on film name, users are redirected to seat details page.

4.4 Seat Details



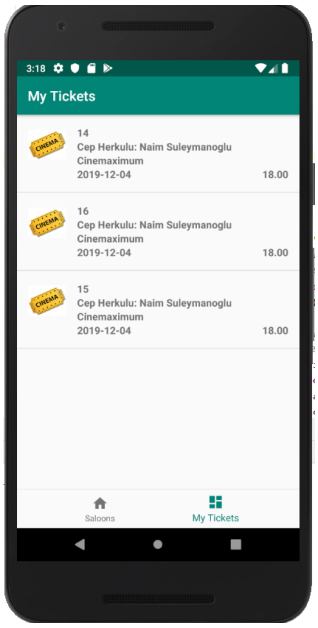
- Seat details of selected film are listed in this page.
- There exist 2 types of seats, which are seats in green or red.
- Green seat means seat is available.
- Red seat means other users has already reserved it.
- In order to make reservation on some green seat, user must click on that seat.
- Users can not make reservation for red seats.
- After clicking on green seat, users are redirected to reservation page.

4.5 Reservation



- After clicking on one of the available seats, users have one minute to complete reservation.
- In order to complete reservation, users must provide their email address.
- After one minute, system sends users back to seat details page.
- After successful reservation, users are redirected to home page.

4.6 My Tickets



- In home page, there exists a My Tickets section.
- In this page, previously bought tickets are listed.
- Tickets are listed with their saloon, film, seat, date and time.

5 Conclusion

Thanks to this project, I had a chance to have hands-on experience on Android programming from scratch. It was useful for me due to the fact that I was responsible from both server side and client side to deployment, I worked as a full

stack developer. Using socket programming, I implemented multi threaded, locking based application. This project was a mixture of things that we covered in CmpE436 during semester.

Of course there exists some features that I could not implement during development. I was planning to get current available saloons, films using 3rd party API, or from SineBU's monthly schedule. However I could not find any API, and had no idea about HTML crawling, I had to add current available films to database manually.

Overall, it was a great experience for me. I have learned many things thanks to this project, from Android programming to AWS deployment. In the future, missing parts that I mentioned above and payment methodology will be implemented and this application will be used for online reservation system for SineBU.

6 References

1. CmpE436 Lecture Notes

2. Amazon Web Services

3. Udemy - Android for Beginners

4. Github

5. Cinemaximum