Dental Clinic Management System

Product Documentation

Version 1.4

May 31, 2018

Dental Clinic

management system

EPOKA University
Computer Engineering
Software Engineering Course – CEN302

Team Members:
Arbli Troshani
Gerd Alliu
Enxhi Ferhati
Egi Gjevori

Project:
github.com/arblitroshani/SEproj

Table of Contents

1. Executive Summary 5
1.1 Project Overview 5
1.2 Purpose and Scope of this Specification 5
2. Product/Service Description 7
2.1. Product Context 7
2.2. User Characteristics 7
2.3. Assumptions 7
2.4. Constraints 8
2.5. Dependencies 8
2.6. Legal Concerns 8
3. Requirements 9
3.1 Functional Requirements 9
3.2 Non-Functional Requirements 12
3.2.1 User Interface Requirements 12
3.2.2 Usability 12
3.2.3 Performance 12
3.2.4 Manageability/Maintainability 13
3.2.5 System Interface / Integration 14
3.2.6 Security 15
3.2.7 Data Management 15
3.2.8 Standards Compliance 16
3.2.9 Portability 16
3.3 Domain Requirements 16
4. Software Analysis and Design 17
4.1 User Scenarios 17
4.2 Use cases 24
4.3. Behavioral Diagrams 30
4.3.1. Use cases 30
4.3.2. Activity Diagrams 35
4.3.3. State Diagrams 46
4.3.4. Sequence Diagrams 52
4.3.5. Collaboration Diagrams 59
4.4. Data Flow Diagram (DFD) 61
4.5. Entity Relationship Diagram (ERD) 65
4.5.1. Relational Model 65
4.5.2. Nonrelational Model 67

4.6. Structural Diagrams 68
4.6.1. Class Diagram 68
4.6.2. Object Diagrams 69
4.6.3. Component Diagram 71
4.6.4. Deployment Diagram 72
4.7. Mobile app – Sketches 73
4.8. Mobile app - Detailed Design 77
4.8.1. Color Palette 77
4.8.2. Dimensions 78
4.9. Mobile app – Screenshots 82
4.10. Web app – Sketches 90
4.11. Web app – Detailed Design 93
4.12. Web app - Screenshots 96
5. Implementation 100
5.1. General product overview. 100
5.2. NodeJS and Firebase 100
5.3. Code Snippets – Web app 102
5.4. Code Snippets – Android app 105
5.5. Code Snippets – Cloud Functions 107
5.6. Testing 108
5.7. Technology stack 110
5.8. Open Source Libraries used 110
5.8.1. Android app 110
5.8.2. Web app 110
5.9. Versioning 110
5.10. License 110
5.11. Instructions of use 111
5.11.1. Android app 111
5.11.2. Web App 111
6. Project Management (SPM) 112
6.1 Gantt Chart 112
6.2 Task distribution Chart 113
6.3 Software development lifecycle 114
6.4 Network Diagram 115

1. Executive Summary

1.1 Project Overview

Considering the heavy burden of multiple tasks dentists cope with daily, it seems of relevance to create a management system encompassing the handling of on-site tasks (in the clinic) and off-site tasks (outside the clinic) alike. The dentistry management system aims to create an easy-to-use, all-in-one tool for practitioners at every stage of their work.

The environment shall comprise a design such that it meets several important requirements presented by the practitioners and clients altogether, providing the latter with a fast and reliable way of managing their visits, payments, medical history, and treatment plans in a comprehensive and well-organized set of tools while aiding the management of the agenda and finances.

In order to substantiate our goals, we find it very important to make a strong connection between the user's needs and our implementation. This mindset is important to our approach, since we intend to continuously improve aspects of the product accordingly.

Furthermore, an important aspect of our work is providing different views of the system in a flexible way, making it easier for all types of users to access their information with ease.

To facilitate the aforementioned aspects in the design, the product shall incorporate different technologies such as NodeJS, Android, Firebase BaaS.

The implementation of these technologies will guarantee a high degree of optimization in terms of load balancing and refactoring. Using these technologies would also make it possible for users to access their data remotely, at all times.

Additionally, the system shall be organized in hierarchical cloud-based manner, ensuring some degree of abstraction and modularity, to make possible the management of different duties in the clinics independently, but with centralized control and whilst satisfying security concerns.

1.2 Purpose and Scope of this Specification

The purpose of this specification is to assess the current state of the product design and to document the entire process based on design issues and the audience.

This specification encompasses several aspects of the process being discussed in an as broad scope as possible. Thus, in this scope we address the following:

- In depth documentation of the features of the product
- Technical overview of the system processes and views
    - This is discussed in Part 2.1 and throughout the document
- User and System Requirements
- Components & Functional/non-functional requirements
    - These are discussed in Part 3 in some detail
- Definition of users' means of using and accessing the product
    - Use cases/scenarios discussed in Part 4
- Dependencies and Constraints
    - These are discussed in Part 2.4/5 of the Document

Aspects not included in the scope are as follows:

- Legislative requirements for the product in terms of privacy
- Auditing and financial considerations of the product

2. Product/Service Description

2.1. Product Context

DCMS is an essential work-tool which serves our purpose and mission, in that it provides all necessary functionalities and benefits of a management system. Given the lack of a present system which fulfills the broad scope of requirements in the current market, DCMS is built with the user in mind, thriving on a high level of user-friendliness, thus making it an asset to patients and practitioners alike.

This product gives a clean, flexible and efficient solution to the daunting task of managing patient's data and medical history on paper, with an electronic, cloud solution.

The flexibility is offered in several ways:

- Multiple platform accessibility: web and android
- Multiple ways to sign in: email and password, Gmail, Facebook, twitter, phone number
- Real time interaction and notifications between doctor and client

Many industries and businesses are now adopting this new philosophy, given the relatively cheap upfront cost of setting up and managing a cloud-based platform of apps.

It is related to the existing system of placing orders with the many depots related to the clinic.

2.2. User Characteristics

Our customers include the entire staff of a dental clinic: dentists, clients and administrative staff.

- Administrator / Manager
  Might as well be one of the other staff members. Acts as a secretary. Coordinates appointments, logistics and payments.
- Dentist
- Client

2.3. Assumptions

- The core part of our software is implemented in the android platform.
- This comes with the small overhead of having to be equipped with android devices, in case part of the personnel uses iOS devices. The clients are expected to have a basic knowledge of smartphones.
- The administrative and managerial part will be in a web interface. This assumes an active internet connection and a device which has access to the internet (laptop, PC, mobile device)
- We assume the staff is familiar with the English language.
- Also, we assume all client Dental Clinics follow a similar workflow.
- Based on the technologies we intend to use, we assume a consistent Cloud-Client connection.
  The advantages of using android devices are:
  - High variety of form factors (tablet, mobile phones)
  - High variety of devices in different price ranges

2.4. Constraints

Design options are constrained by:

- real time communication between doctors and clients
- real time notifications on the client side
- multiple platform integration
- security concerns on data accessibility
- high resolution photo evidences in the medical record
- offline usability

Chosen design options give us the following technical constraints:
- Using Cloud Firestore as a database
- NoSQL database design limits the execution of complex queries

2.5. Dependencies

Dependencies that affect the requirements:

- This product is dependent on a successful communication with the related dental depots. This is achieved through the weekly generated reports of the used products quantities.
- The product is also dependent on the Cloud performance.

2.6. Legal Concerns

Our system uses national identity number to validate and uniquely identify users. The main reason for this is to facilitate the work flow for clients that do not use the app. The advantage is seen when the user later opens an app account and all the data is automatically linked. It also provides a unified way of setting appointments and treatments among all clients. The application scans the users' data using Optical Character Recognition technology provided by BlinkID library.

The usage of the national ID card comes with some legal concerns though.

- The system does not process in any way the info stored in the chip (biometric data).
- The system does not distribute the information externally.
- The data is stored safely, backed by strong security rules in the database.
- The administrator and doctors do not have access to these data.
- There is no need to store a scanned copy of the ID card.

3. Requirements

Priority Definitions

- Priority 1 – The requirement is a “must have” as outlined by policy/law
- Priority 2 – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits
- Priority 3 – The requirement is a “nice to have” which may include new functionality

It may be helpful to phrase the requirement in terms of its priority, e.g., "The value of the employee status sent to DIS **must be** either A or I" or "It **would be nice** if the application warned the user that the expiration date was 3 business days away". Another approach would be to group requirements by priority category.

3.1 Functional Requirements

Req#	Requirement	Comments	Priority	Date Reviewed	SME Reviewed / Approved
R_S_1	Handling multiple account types	Based on the status of the user, each will have their own view of the system	1	May 31	Arbli Troshani
R_S_2	Handling unregistered users	The system should be able to handle unregistered users, providing restricted information	3	May 31	Arbli Troshani
R_S_3	Account login restriction	The admin should be able to login only on the web, clients and doctors only on the app.	3	May 31	Gerd Alliu
R_D_1	Keeping track of medical records and past treatments history	Doctors should be able to view and update patients’ medical records.	1	May 31	Arbli Troshani
R_D_2	Providing availability agenda	Doctors provide their availability information to the system.	3	May 31	Arbli Troshani
R_D_3	Submitting performance reports	Doctors or other staff members can compile reports on a weekly or treatment basis.	2	May 31	Arbli Troshani
R_D_4	Managing appointments	Doctors should be able to manage their appointments.	2	May 31	Arbli Troshani

R_D_5	Viewing work related info	Doctors should be able to view their specific work information such as wage and working hours.	3	May 31	Arbli Troshani
R_A_1	Managing new staff entries	The administrator should be able to approve the creation of doctors' accounts and other employees' data.	2	May 31	Gerd Alliu
R_A_2	Adding treatments offered by the clinic.	The administrator can create new treatment plans based on patients' requests.	2	May 31	Gerd Alliu
R_A_3	Managing treatments based on type and date	The administrator can edit and update treatments to better arrange them.	1	May 31	Gerd Alliu
R_A_4	Staff payroll management	The administrator manages payrolls based on base salary and percentage according to treatments' pricing.	1	May 31	Gerd Alliu
R_A_5	Pricing and services	The administrator should be able to add and edit services offered by the clinic.	1	May 31	Gerd Alliu
R_A_6	Summarizing and generating financial reports.	The admin should be able to dynamically create financial reports based on current financial data.	2	May 31	Gerd Alliu
R_C_1	Sign up and first login	The patients should be able to create their accounts providing credentials such as email or phone no.	1	May 31	Arbli Troshani
R_C_2	View list of services	The patients should be able to view the list of services offered by the clinic when signed in or not.	2	May 31	Arbli Troshani
R_C_3	Service details	Each service should be displayed with a description, list of doctors and list of photos.	2	May 31	Arbli Troshani
R_C_4	Extra profile info	After every successful new Gmail account login, the patient should provide birthday and phone no. credentials.	1	May 31	Arbli Troshani
R_C_5	Clinic info	The patient should be able to view basic information about the clinic including location, open hours, phone no. etc..	3	May 31	Arbli Troshani

R_C_6	Scheduling new appointments	The patients should be able to create an issue for a new appointment, providing information about their problem.	1	May 31	Arbli Troshani
R_C_7	Receive notifications on new appointment	The patients should be notified whenever their appointment has been set with the correct time and date info.	2	May 31	Arbli Troshani
R_C_8	Access personal medical files and records	Patients should be able to access their medical file at all times, in order to have knowledge of their treatment history.	1	May 31	Arbli Troshani
R_C_9	Client is presented with the paperless consent agreement.	Only applicable to select treatments, like surgeries and orthodontics.	2	May 31	Arbli Troshani

3.2 Non-Functional Requirements

3.2.1 User Interface Requirements

- Different screen resolutions based on devices
- Receive native push notifications in real time
- Sliding navigation drawer for the app
- Static navigation drawer for the web page
- Simplistic and responsive design

3.2.2 Usability

- Accessibility
    - The software shall be easy to access remotely and at all times, since both patients and doctors will use the application on their devices.

- Responsiveness
    - The software shall be responsive both in design and data transactions, especially because of the reliance on the Cloud services.

- Flexibility
    - The software shall be easy to update in order to accommodate new requirements
    - The software shall be designed in such a way that the isolation and management of errors is possible

- Effectiveness
    - The software shall provide both staff and clients with practical tools of managing their data and with a convenient way of communicating their needs across the platform.

- Efficiency
    - The software will provide users and administrators with a fast and reliable way of accomplishing their goals such as creating appointments or updating medical information in little time at their own convenience.

3.2.3 Performance

Capacity

The backend is built on top of Google's infrastructure and thus scales very well horizontally.

- Database writes are limited to 2500 per second, which will be more than enough for our use case.
- Maximum concurrent connections for mobile/web clients are limited to 100000 per database.
- Maximum API request size is 10 MB

- Maximum number of documents that can be passed to a Commit operation in a transaction is 500
- Maximum number of composite indexes for a database is 200
- Maximum function call depth is 20

Availability

- The app will be live 24/7
- Is has a very low probability of downtime, around 0.05%
- It will be region independent, but available in English only
- Impact of downtime will be very minimal, considering the high reliability of the Google infrastructure.

Latency

Database operations will have a latency of approximately 100ms in Cloud Firestore, and 10ms in Realtime Database.

3.2.4 Manageability/Maintainability

..4.1 Monitoring

The system will be subject to periodic evaluation. This evaluation will be performed by assessing the data integrity and by monitoring error logs generated automatically.

Few corner cases shall be predicted and handled within the design in order to suppress non-substantial errors and to detect and handle substantial errors appropriately.

To correct the errors, the administrator shall be able to follow specific procedures with many prompts and validations.

..4.2 Maintenance

To isolate and manage issues easily, the system shall be designed in an atomic and modular manner. This will be evident in the separation of views for different user types and the avoidance of rigid relational constraints in terms of database. Also, the administrator shall be provided with a proper interface to perform maintenance operations.

..4.3 Operations

Specify any normal and special operations required by the user, including:

- Approval of a major transaction such as Sign-up or Medical record access
    - Data integrity is not possible without the administrator’s approval
- Handling of idle and unattended periods in the app
    - The user shall operate under some constraints while some of his operations are not approved
- Backup operations
    - These operations shall be handled by the cloud storage

3.2.5 System Interface / Integration

Network and Hardware Interfaces

The app will use either Wi-Fi, or mobile data to connect to the internet. Other network related issues are automatically handled by Firebase Infrastructure, including connection monitoring, operation queueing during offline periods, etc.

System's Interfaces

User interfaces

The users and doctors will be able to authenticate using the following methods, but the app uses a unique identifier which is not affected by the possibly different sign in method.

- classic username and password
- email and password
- Gmail account
- Phone number

The signing of the consent will be done electronically, complying with all legislative regulations, according the specified template by the clinic.

Users will be able to access the system in a proprietary manner, being displayed with different UI interfaces and functionalities based on their roles and the respective platforms.

The supported platforms are Android based Mobile devices and Web, which constitute the Hardware interfaces as well.

The application on both platforms is suited to different time zones but is only available as an English-based locale.

Software interfaces

To maintain a reliable and efficient exchange of data between system components, the communication is based on the Firebase SDKs and APIs.

3.2.6 Security

Protection

- Firebase Realtime Database Rules determine who has read and write access to your database, how your data is structured, and what indexes exist.
- These rules live on the Firebase servers and are enforced automatically at all times.
- Every read and write request will only be completed if your rules allow it.
- By default, rules are set to allow only authenticated users full read and write access to your database.
- This is to protect your database from abuse until you have time to customize your rules or set up authentication.

Authorization and Authentication

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to our app.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with our custom backend.

To sign a user into the app, you first get authentication credentials from the user. These credentials can be the user's email address and password, or an OAuth token from a federated identity provider. Then, you pass these credentials to the Firebase Authentication SDK.

The backend services will then verify those credentials and return a response to the client in the form of JWT (JSON web token) which is the standard used by Firebase, and which can be used to extract user claims for further usage.

3.2.7 Data Management

- Cloud Firestore and Realtime Database, which we will primarily use are NoSQL databases.
  NoSQL stands for “non-SQL”, but also “not only SQL”, and it is a non-relational database, which stores data in a large file usually. It offers more flexibility in database design, which does not restrict us in the relations between entities. It also scales better horizontally, meaning that it is better distributed in a more efficient network of nodes.
- Data is saved in a JSON tree. This offers easy encoding and decoding directly from Java objects in Android.
- In Cloud Firestore, different from the Realtime Database, data is stored in Documents and Collections, thus offering better query support, and an offline first approach.

3.2.8 Standards Compliance

Dental Agreement Contract which includes the “Consent to Proceed” is a legal documentation signed by the user before each specific treatment. In this document is explained in detail the procedure of the treatment, the materials used, the drug prescription, the treatment fee and the client’s responsibilities in case there is any problem. This is a documentation that preserves the rights of clients and doctors according to law and ethics.

3.2.9 Portability

- Use of Firebase products, which offer flexibility in different platforms, such as: Android, iOS, Node.js, Java, Python and GO
- Real time updates
- Offline-first approach
- The same backend will power both the android client and web client, with possibility of adding an iOS client in the future, with not many interventions in the underlying infrastructure.

3.3 Domain Requirements

The system manages everything related to a dental clinic, including specific features like Perio charting, anamnesis, and appointment scheduling in the style of a dental clinic.

4. Software Analysis and Design

4.1 User Scenarios

Notice: We are using the terms "User" and "Client" interchangeably, depending on context.

Scenario C1: User is not logged in

1. User is presented with a list of services.
2. User is presented with detailed info about the clinic.
3. User is presented with the option to Sign in.

Scenario C2: User opens a service from the list

1. User is presented with a detailed view of the service, which includes detailed info, list of doctors and supporting photos.
2. User can share or bookmark the specific service.

Scenario C3: User opens a supporting photo

1. On the detailed view of the service, user taps one of the photos in the list.
2. The photo is opened in full screen.
3. User is able to share the photo in other apps like Google Keep or Messenger.

Scenario C4: Users signs in for the first time

1. User chooses to sign in, by pressing a button in the navigation drawer.
2. It is redirected to Sign in activity.
3. User is presented with 2 options: Sign in using Google or using their phone number directly.

Scenario C5: User chooses to sign in using their phone number

1. User presses "Sign in with phone number" button.
2. User selects the country, with the corresponding prefix.
3. User enters the phone number, without the 0 prefix.
4. User presses "Verify phone number".
5. An SMS arrives within seconds, and automatic verification happens, if Google Play Services version installed is > 22
6. Else, the user must input the received code manually.
7. On Success, User has successfully signed in using their phone number.

Scenario C6: User chooses to sign in using Google account

1. User presses "Sign in with Google" button.
2. User selects an account from the list of signed in accounts or adds a new one manually.
3. User is successfully signed in using their Google account.

Scenario C7: First time user sign in, add other personal data

1. If User is signed in for the first time with a Google account
2. User scans their national ID card, to extract info (name, birthday, id).
3. If User is signed in for the first time with a phone number, it should also enter his email.
4. User clicks Ok
5. Registration is successful and complete.

Scenario C8: User signs in (not for the first time)
1. User signs in with one of the methods.
2. In the database, an entry with the details exists, indicating that it is not a new user.
3. User is greeted with “Welcome back”
4. Name, email and profile picture are loaded from the database entry.
5. Sign out button is added.

Scenario C9: User signs out
1. User is already signed in.
2. User clicks Sign out.
3. User is signed out, and the next time he visits the app, will not be automatically signed in.
4. Profile picture and Name fields are replaced with the default values.
5. Sign out button is not shown.

Scenario C10: User opens “My Profile” section.
1. User clicks “My Profile”
2. User is presented with his account information and “Anamnesis”.

Scenario C11: User opens “Treatments” section of ‘My Profile”
1. User opens “My Profile”.
2. User scrolls to “Treatments” tab.
3. User is presented with a list of all treatments, till date.
4. Each treatment information card includes the name, date started and status.

Scenario C12: User opens a specific Treatment
1. User opens “My Profile”.
2. User opens “Treatments” tab.
3. User taps on one of the treatments.
4. User is presented with the list of Appointments in that specific treatment.
5. User is presented with the status of the Treatment, “undergoing”, “finished”.
6. User can see the payment status of the Treatment, “paid”, “unpaid”, “partially paid”.

Scenario C13: User opens a specific Appointment
1. User taps on one of the Appointments in the Treatment.
2. User is presented with a detailed description of the condition and has the dentist’s recommendations regarding medications and future care.
3. User can see the accompanying photos with his appointment.

Scenario C14: User opens Anamnesis section
1. User opens “My Profile”
2. User scrolls to “Anamnesis” tab on the right.
3. User is presented with their personal Anamnesis, which gets edited by the specific doctor.
4. User can share or print a pdf version of their Anamnesis.

Scenario C15: User manages appointments
1. On the home screen, User scrolls to the right, on “Appointments” section.
2. User is presented with all his past and future appointments.
3. Each appointment information card has a title, service required, date and status which is indicated by a color (Red, Green on Yellow)

Scenario C16: User sets a new appointment
1. On the home screen, User scrolls to the right, on “Appointments” section.
2. User taps on the floating action bar on the bottom right to schedule a new appointment.
3. User specifies the day he wishes to schedule the appointment, along with the service required and an optional descriptive message.

Scenario C17: User is assigned a specific appointment
1. Administrator assigns a timeslot in the specified day to the user’s appointment.
2. User receives a push notification, even if the app is closed.
3. User receives a reminder SMS 1 hour before the appointment (by default).

Scenario D1: Doctor creates an account
1. Doctor signs in the app as a regular user, using their preferred method.
2. The administrator assigns the doctor’s profile as an employee, by adding the required extra fields.
3. The next time the doctor signs in, he is presented with the Doctor’s view.

Scenario D2: Doctor is not an employee of the clinic anymore
1. The administrator removes the doctor’s account from the list of employees.
2. The doctor can access the app as a regular user.

Scenario D3: Doctor opens the app as an already signed in user
1. The user is checked if exists in the list of employees.
2. If positive, the specific View for the Doctor is shown.
3. Doctor is presented with a dashboard consisting of “My appointments”, “My patients”, “Services”, “My availability”.

Scenario D4: Doctor adds his available hours in the system
1. On every Saturday, the doctor should enter his available hours for the next week.
2. Doctor opens the Dashboard on the Home screen.
3. Doctor taps “My availability”
4. Doctor clicks the floating action bar to add different time-spans for each day of the upcoming week and finally clicks submit.
5. This information is presented to the administrator, who adjusts the timetable according to these specifications.

Scenario D5: Doctor views the list of appointments.
1. On the dashboard, the user taps on “My Appointments” tab.
2. The doctor is presented with a tabbed view consisting of “Today’s Appointments” and “Upcoming appointments”.
3. On each tab, there is a list with all the assigned appointments, ordered according to the start time.

Scenario D6: Doctor manages his patients and treatments.
1. Doctor taps on “My Patients” tab.
2. It is presented with a list of users.
3. Doctor presses on one of the records in the list.
4. The profile info of the patient in displayed.
5. The doctor can manage or start a new treatment.

Scenario D7: Doctor starts a new appointment in a treatment.

1. Doctor selects a user.
2. Doctor selects a treatment or starts a new one.
3. Doctor adds a new appointment.
4. Doctor enters detailed description of the appointment, which the user can later access in his account.
5. Doctor can add accompanying photos, and use a Perio chart, to better keep track of the problems.
6. Doctor assigns a price to the individual appointment, which gets added to the treatment total.

Scenario A1: Admin user login (successful):

1. Admin opens the /login page
2. Admin chooses an authentication method such as Google Sign-in or Facebook Sign-in by clicking on the respective button.
3. A popup window is displayed, prompting the user to choose an account
4. After selecting the account, the user might need to provide credentials for their account unless they are already signed in.
5. The authentication request is sent to the cloud
6. Upon successful authentication, the user is redirected to the Admin Panel page.

Scenario A2: Admin user login (failure):

1. Admin opens the /login page
2. Admin chooses an authentication method such as Google Sign-in or Facebook Sign-in by clicking on the respective button.
3. A popup window is displayed, prompting the user to choose an account
4. After selecting the account, the user might need to provide credentials for their account unless they are already signed in.
5. The authentication request is sent to the cloud
6. Upon authentication failure, the user is displayed an error message informing them about the error.

Scenario A3: Admin manages their profile

1. Admin is logged in
2. He/she navigates to the My Profile button on the Dashboard
3. After clicking the My Profile button, the personal information page is rendered within the same page, containing editable text fields/labels
4. He/she selects field to update and edits the current information
5. He/she clicks Save Changes
6. The form is submitted
    a. In case of error, an error message is displayed above the edit form
    b. In case of success, a success message is displayed above the edit form

Scenario A4: Admin view and manage existing services

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Services
4. After clicking the card, the list of services is displayed in a tabular form.
5. The admin clicks the Edit button for a specific service row.
6. A modal form is displayed
7. After changing the proper text fields, the admin clicks submit.

8. The form is submitted
    a. In case of error, an error message is displayed at the top of the modal
    b. In case of success, the modal is closed, and the information is changed in the table.

Scenario A5: Admin adds new services

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Services
4. After clicking the card, the list of services is displayed in a tabular form.
5. The admin clicks the New button
6. A modal form is displayed
7. After writing on the proper text fields, the admin clicks submit.
8. The form is submitted
    a. In case of error, an error message is displayed at the top of the modal
    b. In case of success, the modal is closed, and the information is displayed in the table in a new row

Scenario A6: Admin views clients' information

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Clients
4. After clicking the card, the list of users is displayed in a tabular form.
5. The admin clicks on any client's row.
6. A new page is rendered on top of the clients' list, containing the specific user profile information.
7. After clicking on certain labels or profile photo, the content is made editable where applicable.
8. When applicable, the edited information is submitted and saved

Scenario A7: Admin views doctors' information

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Manage doctors
4. After clicking the card, the list of doctors is displayed in a tabular form.
5. The admin clicks on any doctor's row.
6. After clicking on certain labels or profile photo, the content is made editable where applicable.
7. When applicable, the edited information is submitted and saved.

Scenario A8: Admin adds new doctor

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Manage doctors.
4. After clicking the card, the list of all users is displayed in a tabular form.

5. Each user's record has a button for setting as a doctor.
6. This happens in coordination with the fact that doctors have to create an account as a regular user first.

Scenario A9: Admin views and manages existing appointments

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Appointments
4. After clicking the card, the list of appointments is displayed in tabular form, with a text field showing the date of the current day.
5. Clicking on the date field, a calendar is shown, and the admin chooses the desired date
6. After changing the date, the appointments list is refreshed
7. The admin clicks on any appointment row.
8. After clicking on a specific row, a modal is displayed containing an edit form
9. After editing the proper text fields, the user clicks Save changes on the modal
10. The form is submitted
    a. In case of error, an error message is displayed at the top of the modal
    b. In case of success, the modal is closed, and the information is updated in the table.

Scenario A10: Admin confirms requested appointments

1. Admin is already logged in
2. He/she navigates to the Home section unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Appointments
4. The admin clicks the Requested button
5. After clicking the card, the list of appointment requests is displayed in a tabular form.
6. After clicking on a specific row, a modal form is displayed,
7. After writing on the proper text fields (where applicable) and selecting a date in the date field (calendar), the admin clicks Confirm.
8. The form is submitted
    a. In case of error, an error message is displayed at the top of the modal
    b. In case of success, the modal is closed, and the information is displayed in the table in a new row

Scenario A11: Admin adds new appointment for new client (not requested through the app)

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Appointments
4. After clicking the card, the list of appointments is displayed in a tabular form.
5. The admin clicks the New button
6. A modal form is displayed with
7. After writing on the proper text fields and selecting a date in the date field (calendar), the admin clicks submit.
8. The form is submitted
    a. In case of error, an error message is displayed at the top of the modal
    b. In case of success, the modal is closed, and the information is displayed in the table in a new row

Scenario A12: Admin views and manages payroll for doctors

1. Admin is already logged in
2. He/she navigates to the Home button unless the button is selected (highlighted)
3. When the home info is fully rendered within the current page, the admin clicks on the respective card with text: Payroll
4. After clicking the card, the list of staffers with their payroll information is displayed in tabular form.
5. The admin clicks on any appointment row.
6. After clicking on a specific row, a modal is displayed containing an edit form
7. After editing the proper text fields, the user clicks Save changes on the modal
8. The form is submitted
    a. In case of error, an error message is displayed at the top of the modal
    b. In case of success, the modal is closed, and the information is updated in the table.

Scenario A13: Admin manages the total cashflow

1. Admin is logged in
2. Admin navigates to the Finance section
3. He selects “Cashflow”
4. He is presented with a list of all inbound and outbound financial transactions of the clinic.
5. Admin can add a new transaction, and the total will be automatically updated.

4.2 Use cases

Name	Admin login (Web)
Summary	Admin chooses an authentication method to login in the system
Actor	Admin
Description	The admin opens the login page, chooses an authentication method such as Google Sign-in or Facebook Sign-in. Upon successful authentication the admin is directed to the Admin Panel
Precondition	The user who wants to login must have a Google account
Alternatives	There are no alternative options
Post Condition	Gain access to the Admin Panel

Name	Admin manages their profile (Web)
Summary	Admin goes to My Profile and can edit their data
Actor	Admin
Description	The admin opens My Profile. All text fields are editable in case of a change. After updating admin saves the new data
Precondition	Admin must be logged in
Alternatives	In case of an error, a message is displayed above the edit form
Post Condition	Admin views and edits their personal data

Name	Admin views information (Web)
Summary	Admin can view clients and staff personal information
Actor	Admin, Client, Staff member
Description	The admin can go to the list of clients and staff members. After clicking the respective client/staff member a new page with their personal information is displayed
Precondition	Admin must be logged in
Alternatives	There are no alternative options
Post Condition	Admin has general knowledge about the clinics clients and staff members

Name	Admin manages payroll (Web)
Summary	Admin can manage the payroll
Actor	Admin, Staff member
Description	The admin should manage the staff members payroll (working hours and wages)
Precondition	Admin must be logged in and must have information about clinics logistic and staff members working hours and static wages
Alternatives	There are no alternative options
Post Condition	Admin is aware about staff members payroll

Name	User / Doctor signs in (mobile app)
Summary	User or doctor authenticates in the application.
Actor	The User (Client), Doctor.
Description	The User uses one of the available authentication methods. If it is the first time, it also adds the extra required information.
Precondition	The user who wants to login must have a Google Account.
Alternatives	The User can sign in using an active phone number.
Post Condition	Have an active account in the dental clinic database. Can access personal files and set appointments.

Name	User sets new appointment from the app
Summary	The user can request a new appointment on a specific day, for a specific service.
Actor	The User (Client), The administrator
Description	The User, upon sign in, is eligible to request a new appointment electronically from the app.
Precondition	The user should be signed in.
Alternatives	Calling the clinic and having them do it for you.
Post Condition	The user is notified when the appointment is scheduled. He has the option to accept it or decline it if the assigned time slot is not suitable.

Name	User accesses details for a previous appointment
Summary	User accesses all descriptions, recommendations and photos, the doctor has uploaded from the appointment.
Actor	User (Client), Doctor
Description	In “Treatments”, open a specific treatment, then a specific appointment.
Precondition	The user has undergone at least one appointment.
Alternatives	The administrator does it from his account.
Post Condition	The user can share the photos or descriptions.

Name	User views their anamnesis information
Summary	User accesses their medical history information and are able to download every related information as a file at any time
Actor	User (Client), Doctor
Description	In “My Profile”, “Anamnesis”, a list in the form of a timeline containing the information is presented
Precondition	The user should have provided the information beforehand.
Alternatives	The user requests this information verbally from the assigned doctor.
Post Condition	The doctor can rely on this sensitive information in their work.

Name	The admin manages the requested appointments
Summary	Accept, Decline, specify time-slot for a requested appointment.
Actor	User, Administrator
Description	The admin gets a list of all requested appointments from all user accounts.
Precondition	The signed in users have requested an appointment from the app.
Alternatives	The admin sets up an appointment manually for a not signed in user.
Post Condition	The appointment is added in the timetable and presented to the doctor's appointment list for the day.

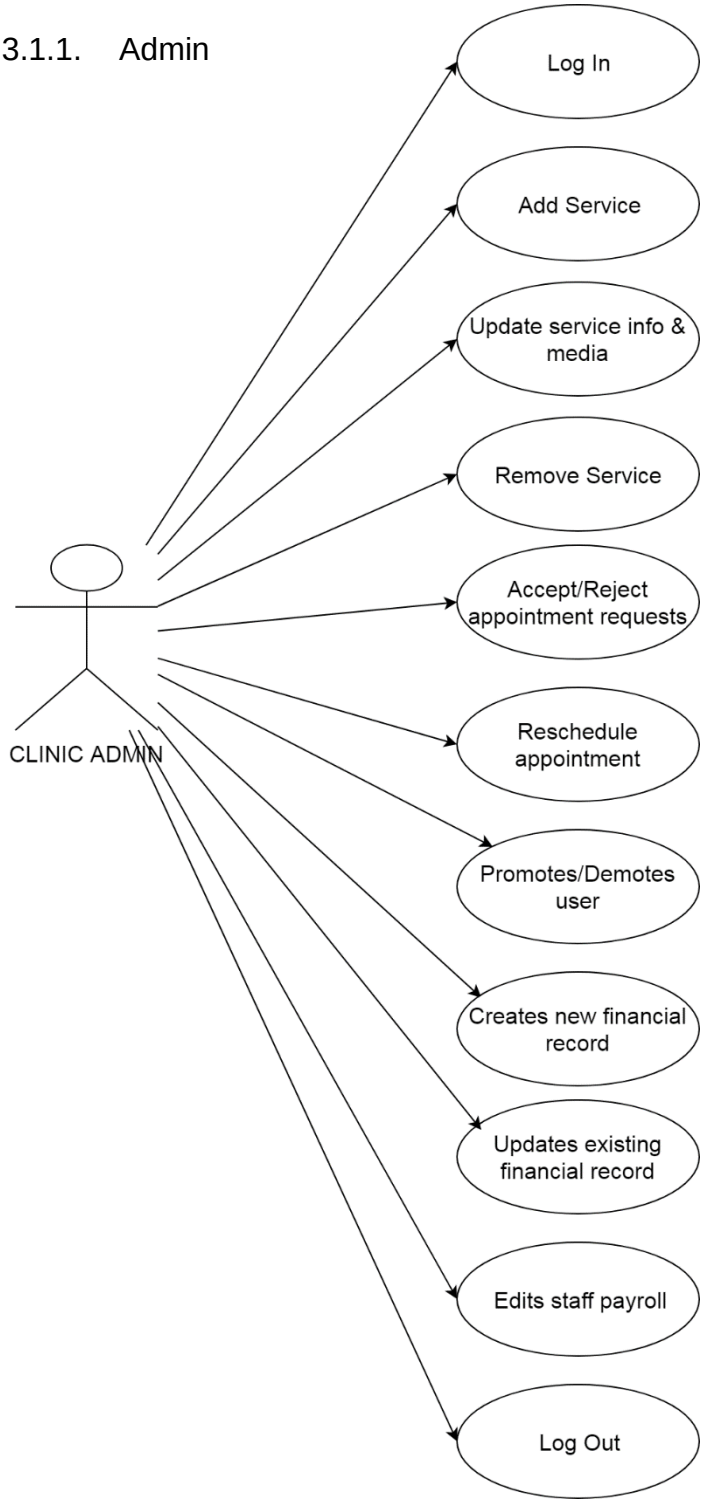
Name	The doctor provides their agenda information
Summary	The doctor should provide their information for their current agenda to the system.
Actor	Doctor, Administrator
Description	The doctor can access the My Availability tab in the navigation bar, and then provides the info for the respective days.
Precondition	The doctor has already been registered in the system. The doctor has already logged in.
Alternatives	No alternatives
Post Condition	The information is placed in the system and is used in the overall business logic.

Name	The doctor views his appointments
Summary	The doctor should access the appointments list for the current day and forthcoming days.
Actor	Doctor, Administrator
Description	The doctor can access the My Appointments tab in the navigation bar, and then views the proper information based on the day.
Precondition	The doctor has already been registered in the system. The doctor has already logged in.
Alternatives	No alternative
Post Condition	None

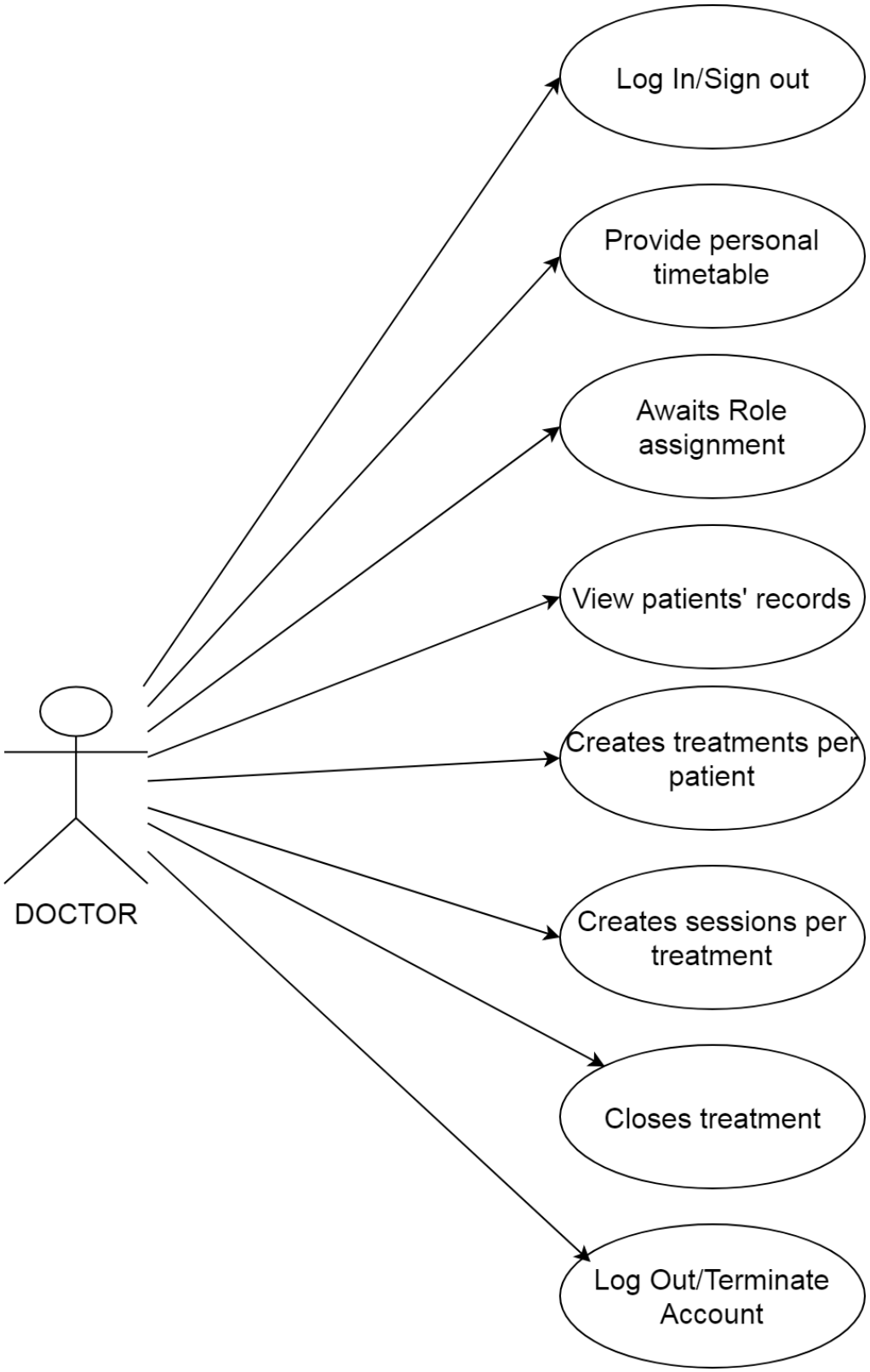
4.3. Behavioral Diagrams

4.3.1. *Use cases*

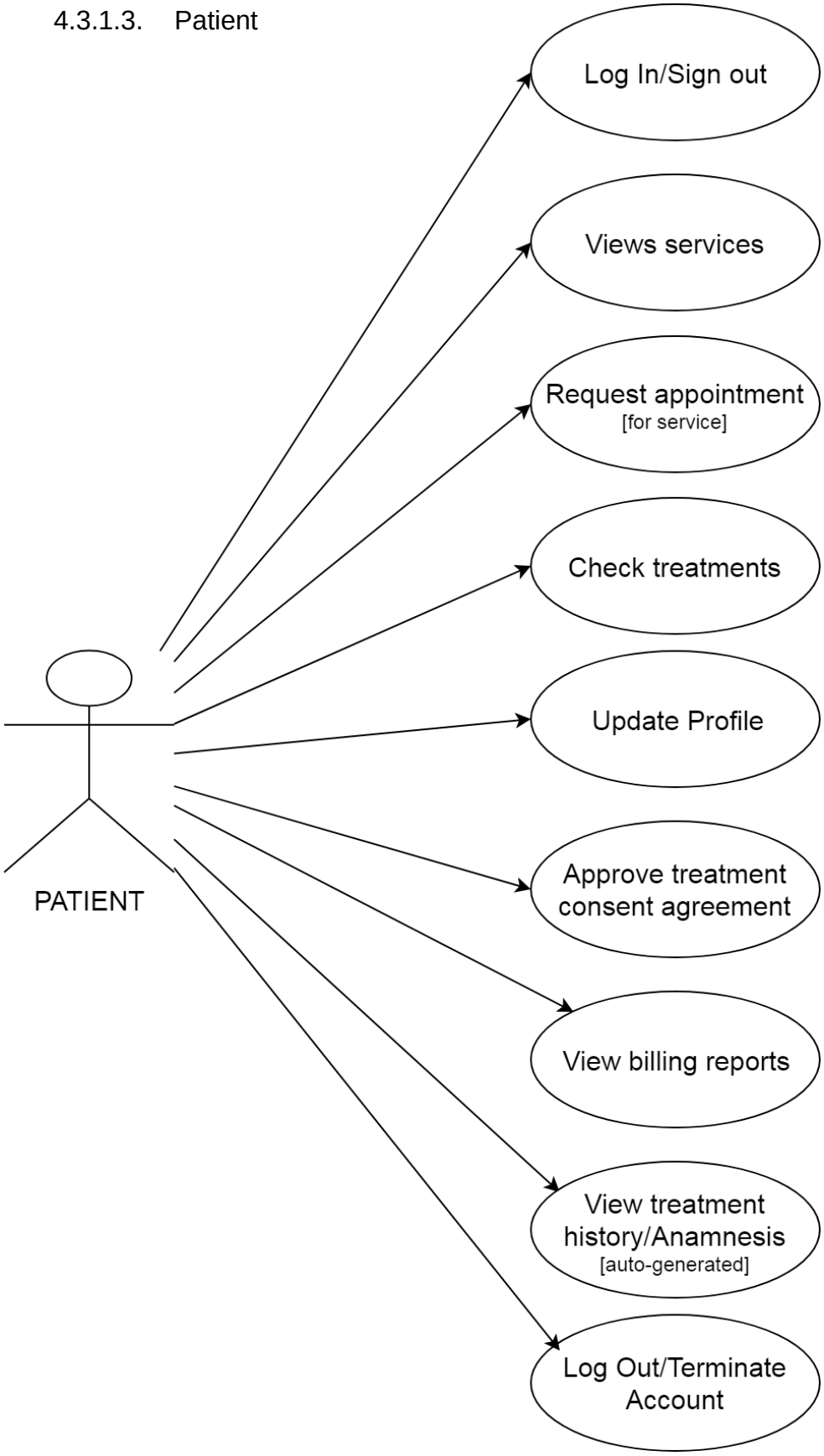
4.3.1.1. Admin



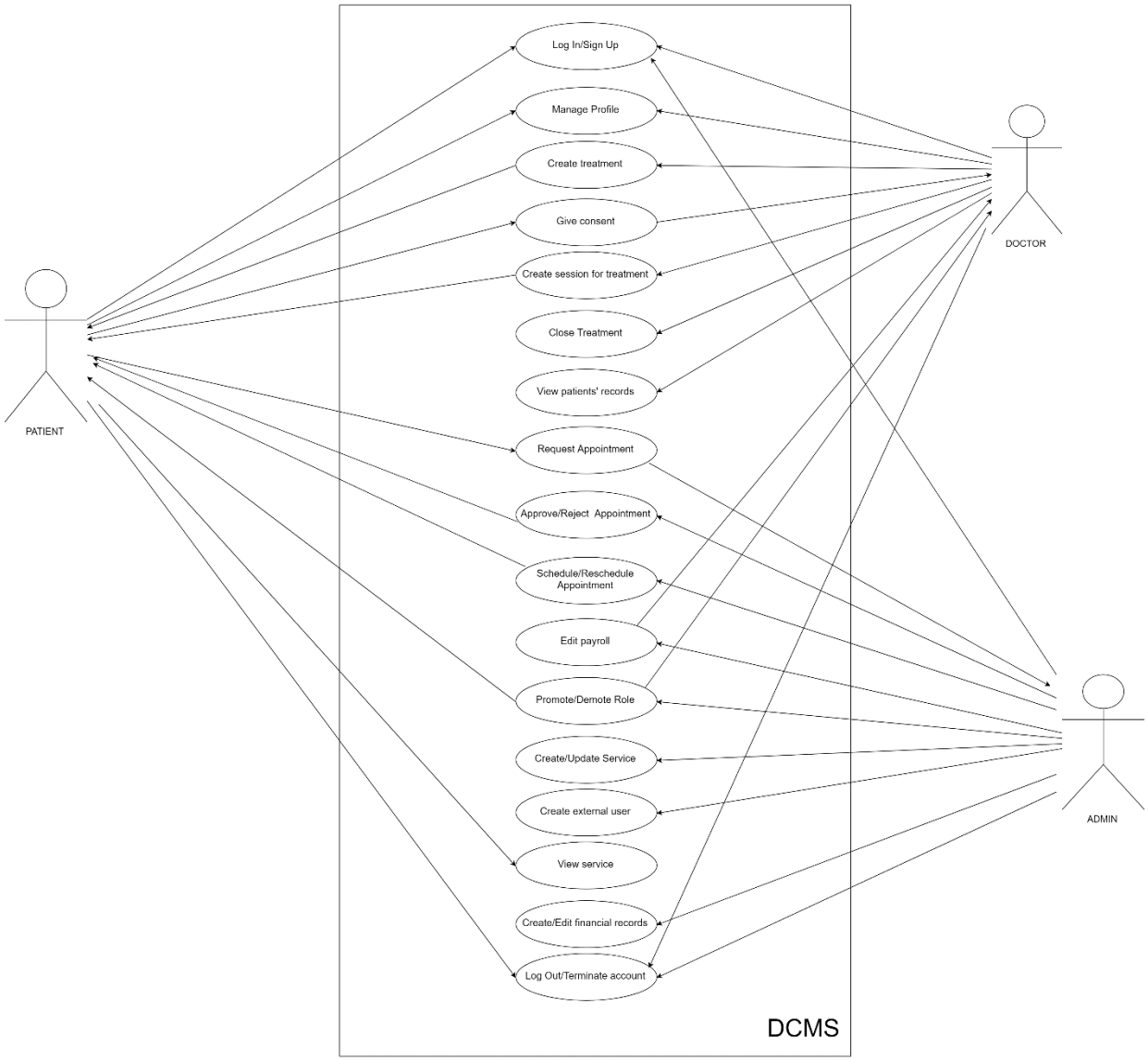
4.3.1.2. Doctor


Log In/Sign out
Provide personal timetable
Awaits Role assignment
View patients' records
Creates treatments per patient
Creates sessions per treatment
Closes treatment
Log Out/Terminate Account
DOCTOR

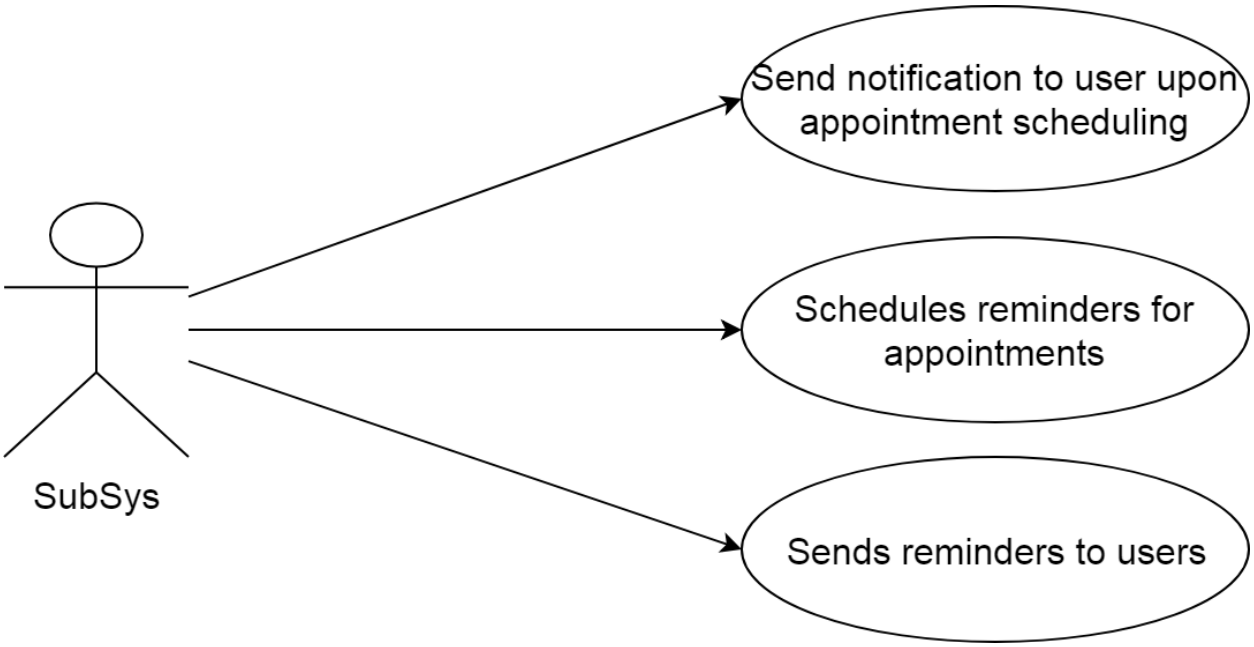
4.3.1.3. Patient



4.3.1.4. General

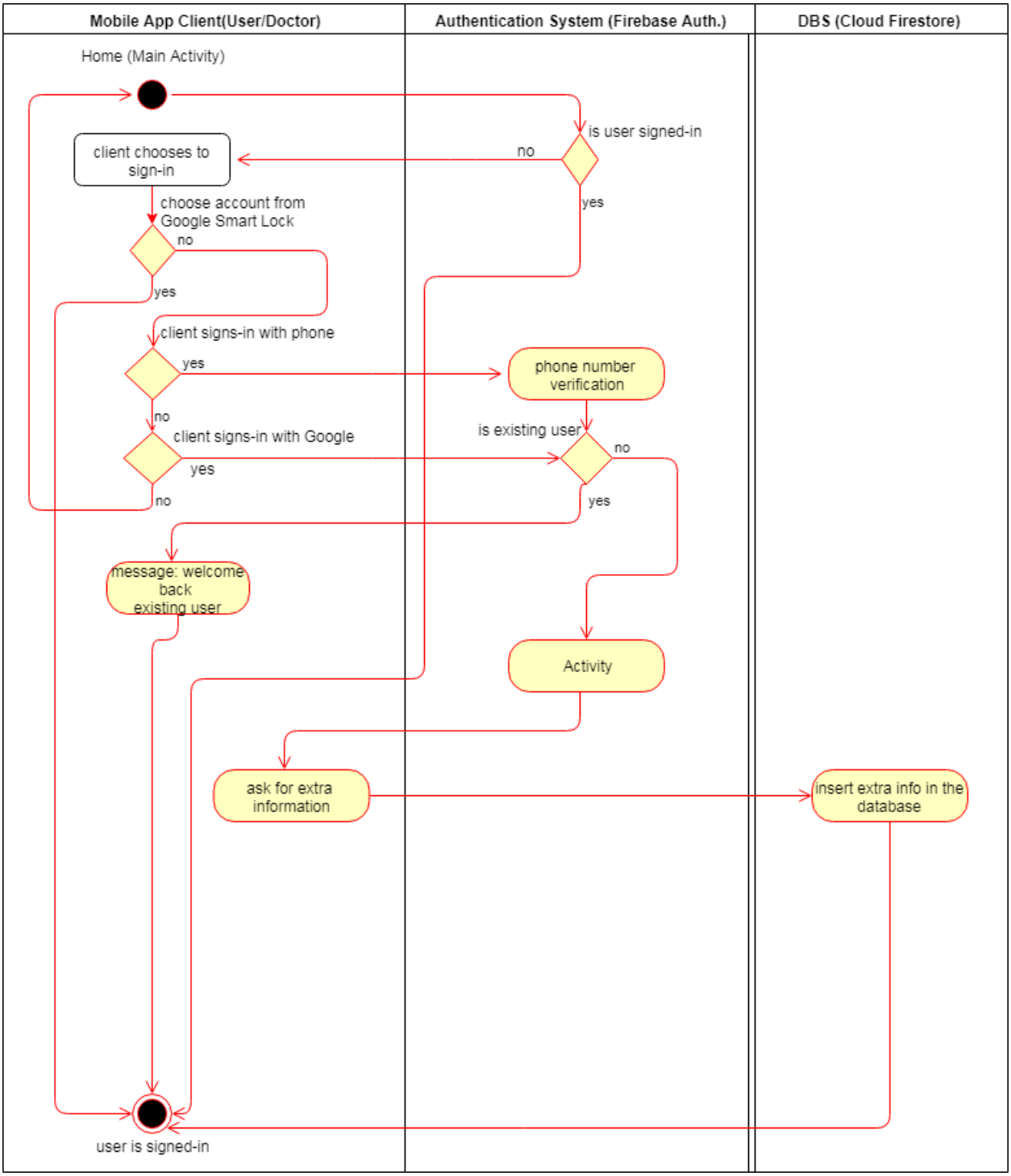


4.3.1.5. Subsystem

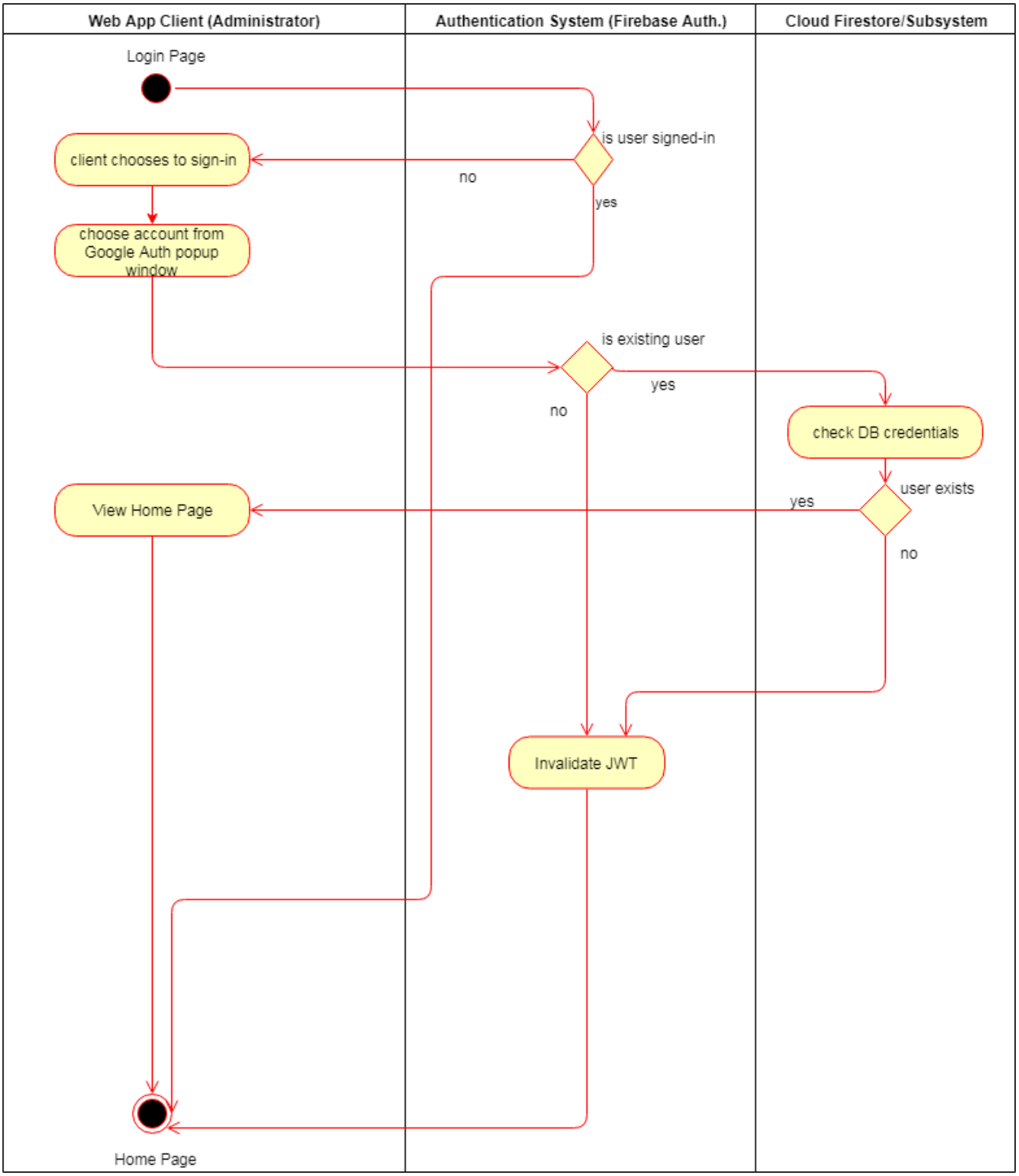


4.3.2. Activity Diagrams

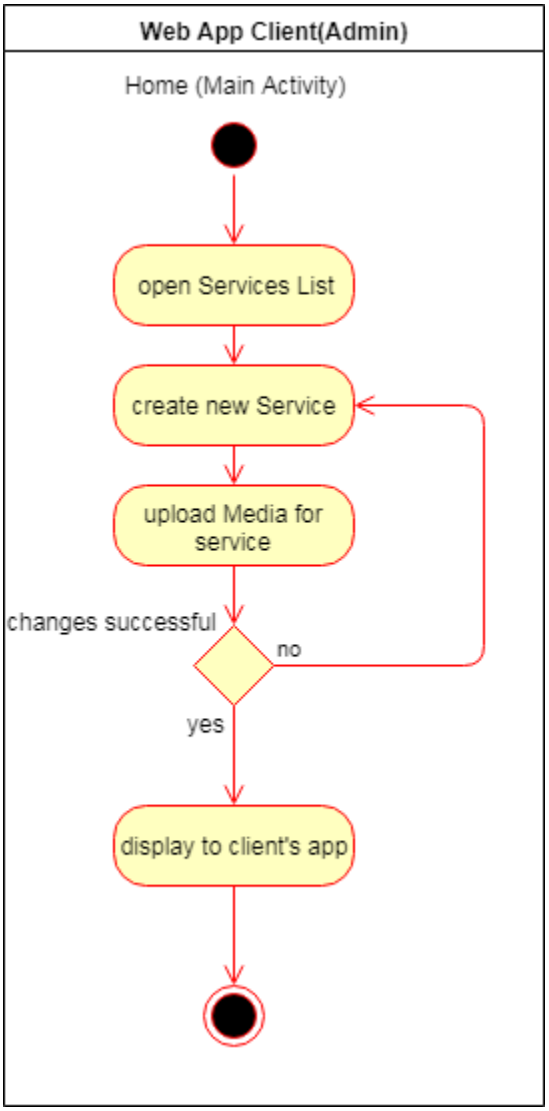
4.3.2.1. Mobile Authentication



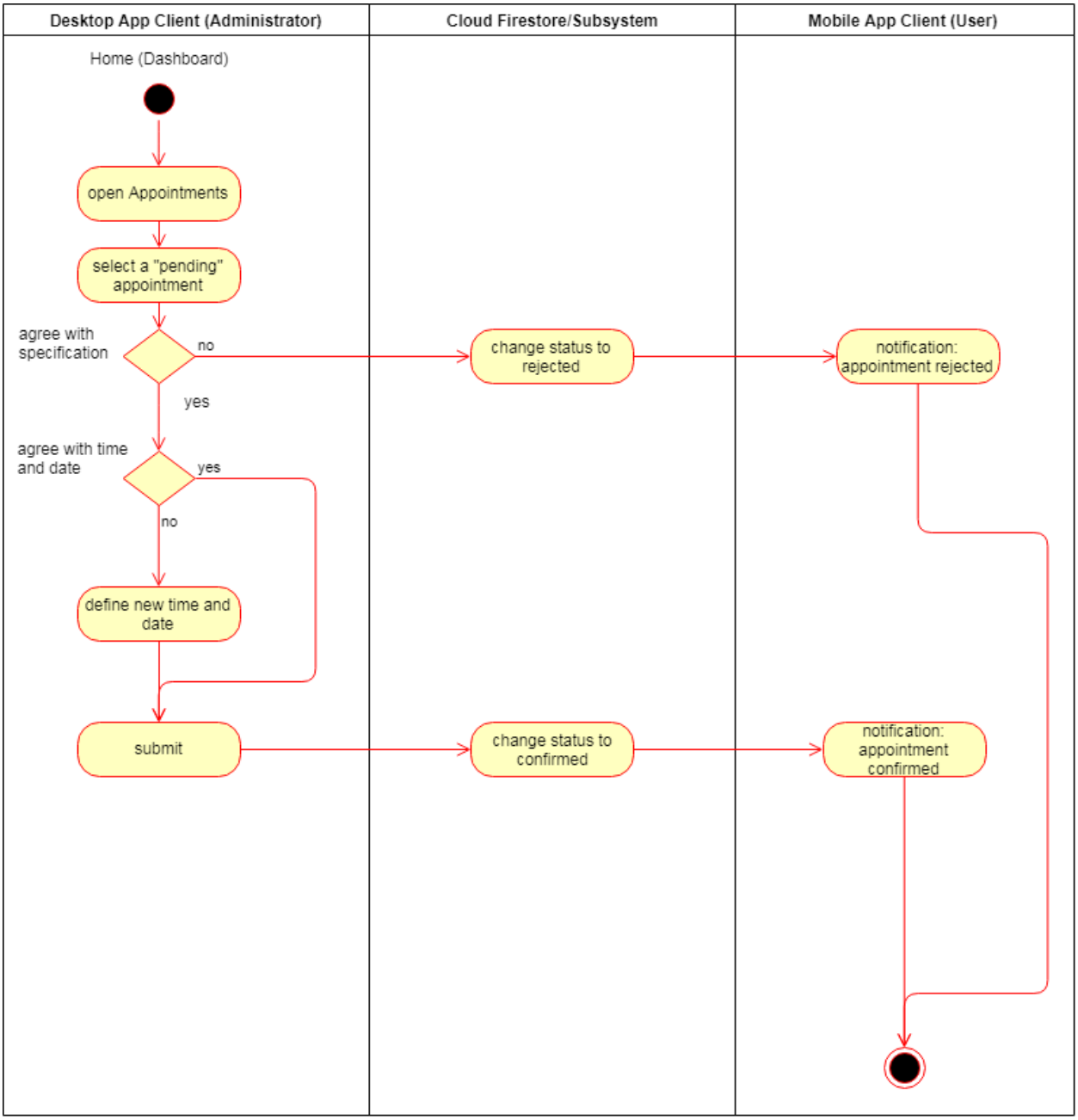
4.3.2.2. Web authentication



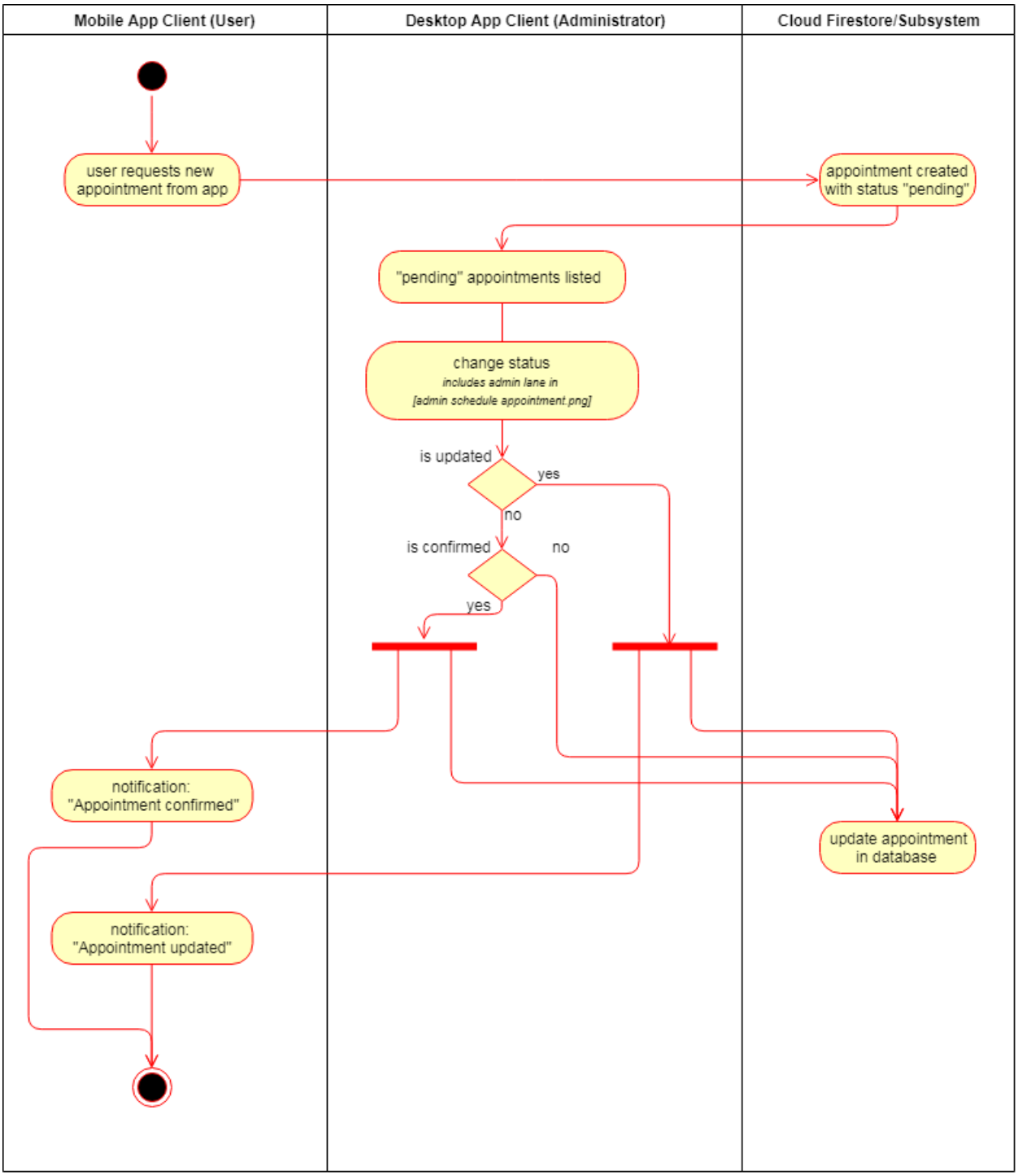
4.3.2.3. Add Service



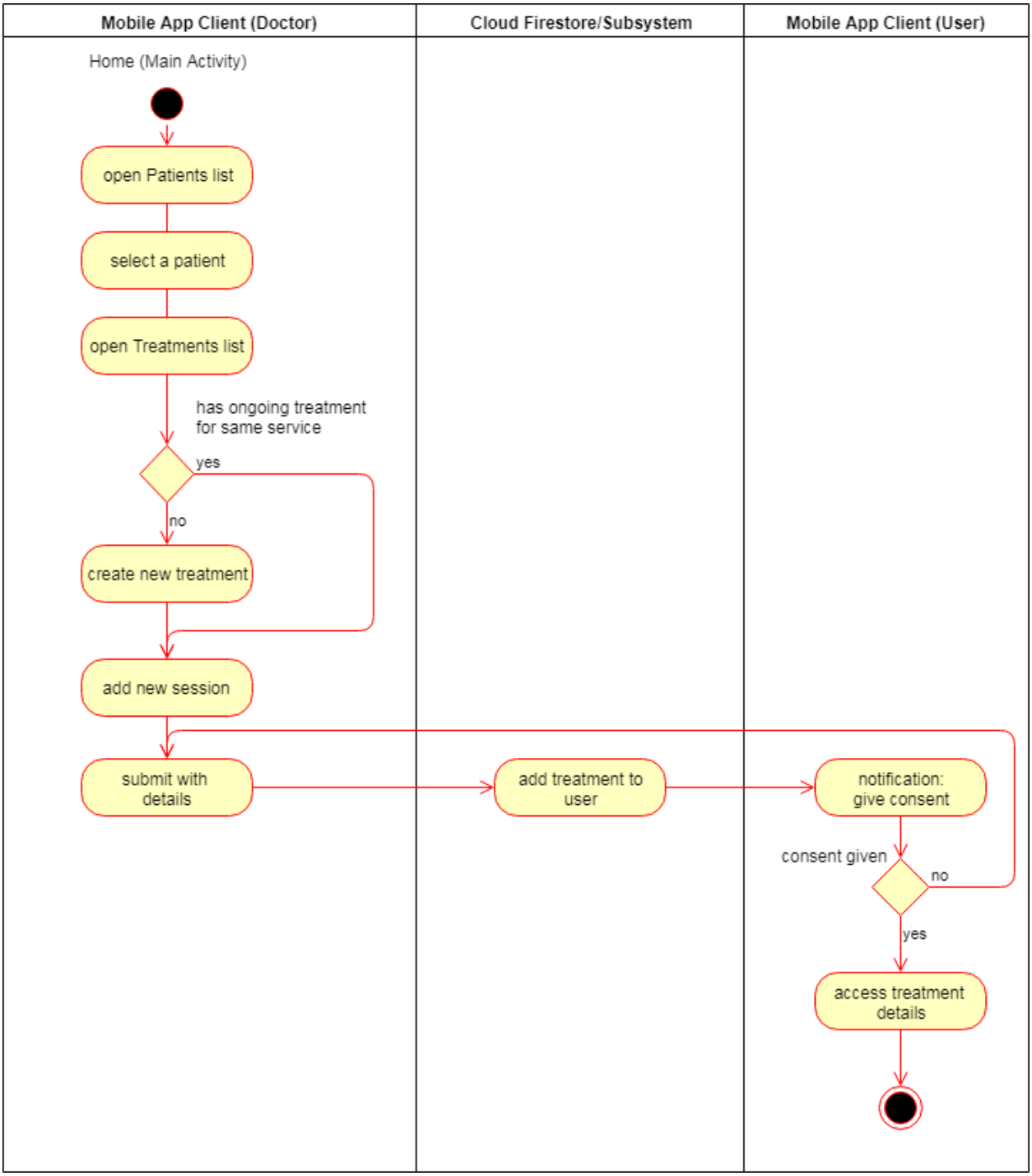
4.3.2.4. Admin – Schedule Appointment



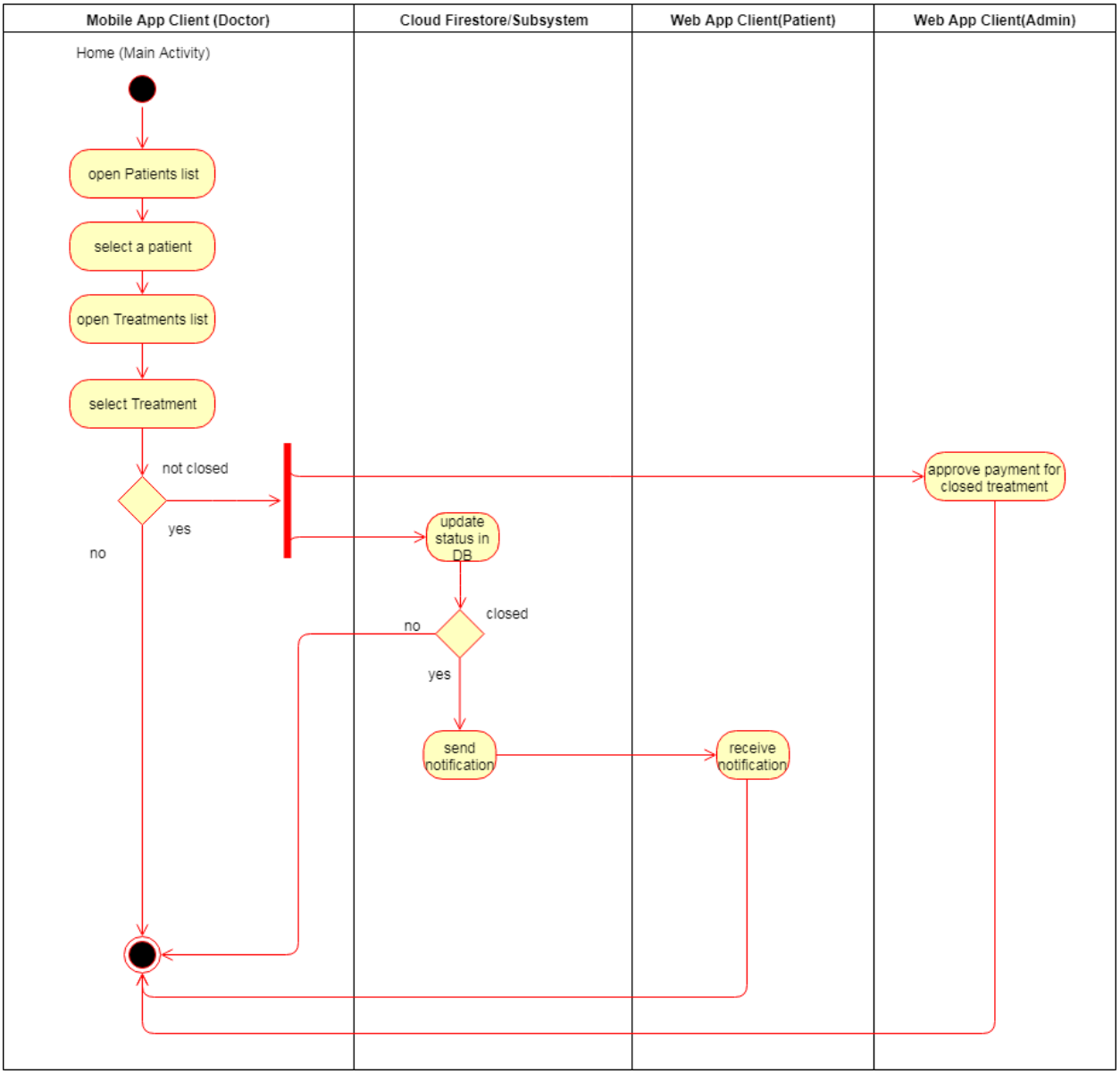
4.3.2.5. Confirm appointment



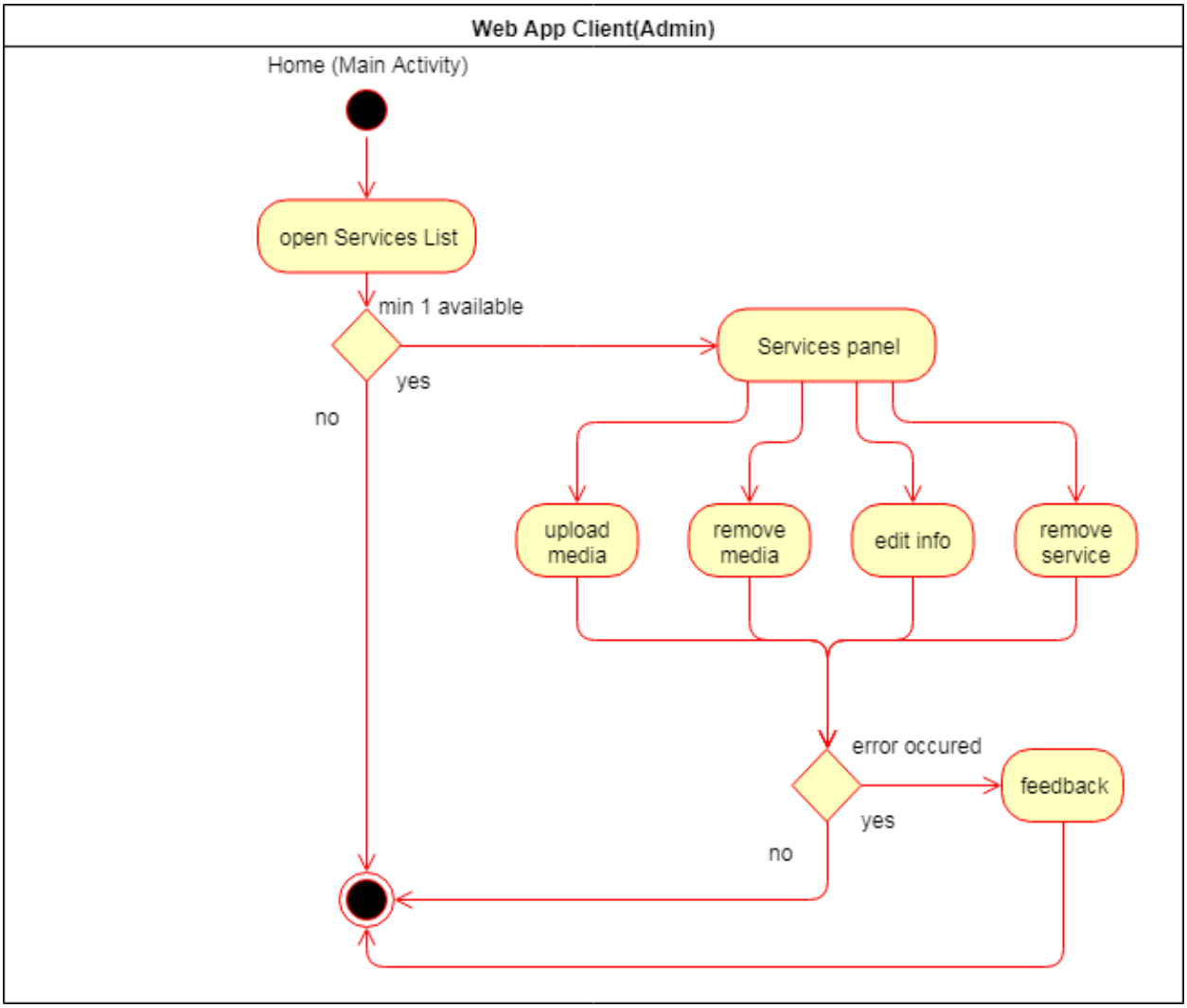
4.3.2.6. Doctor – Add Session



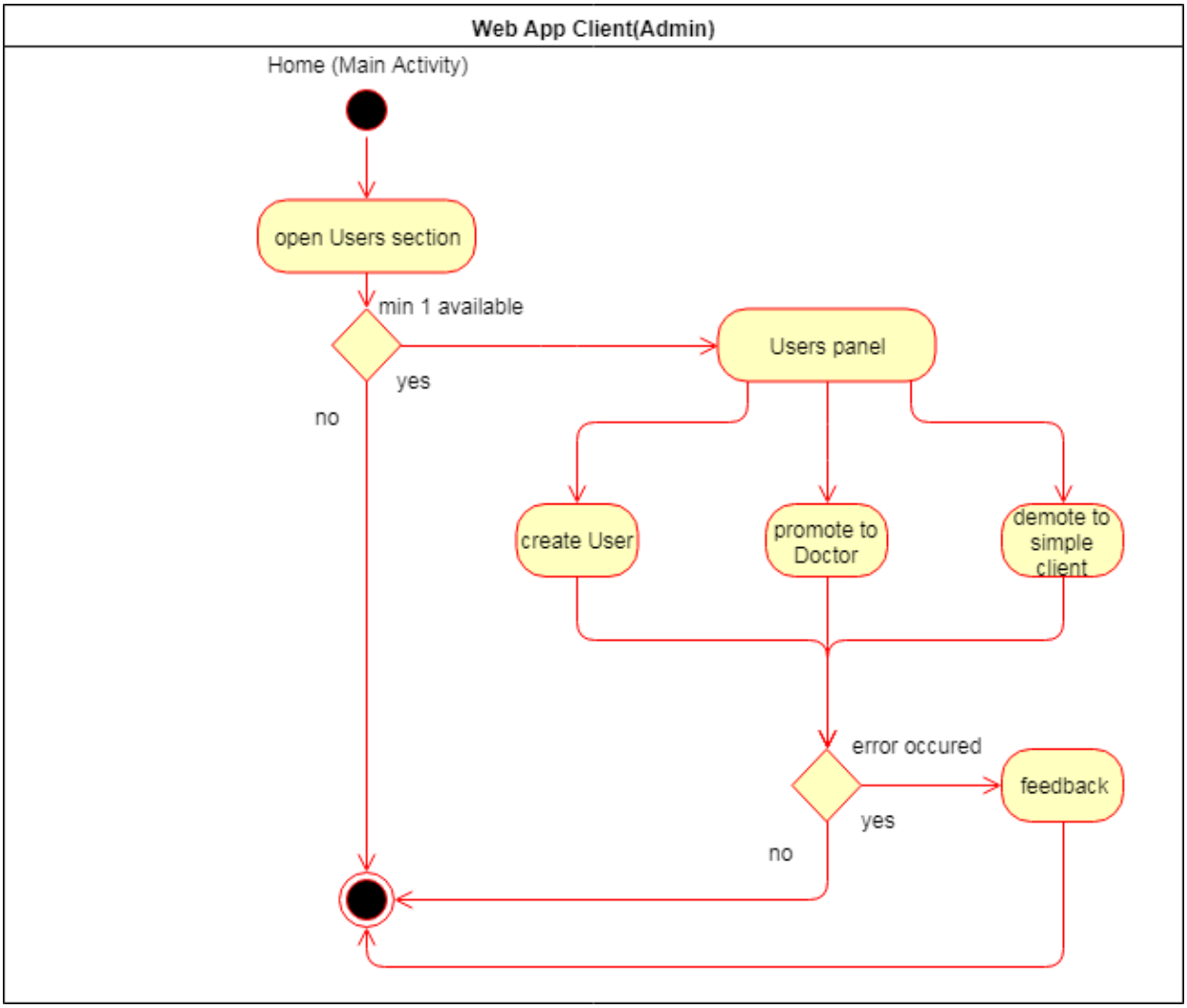
4.3.2.7. Doctor – Close Treatment



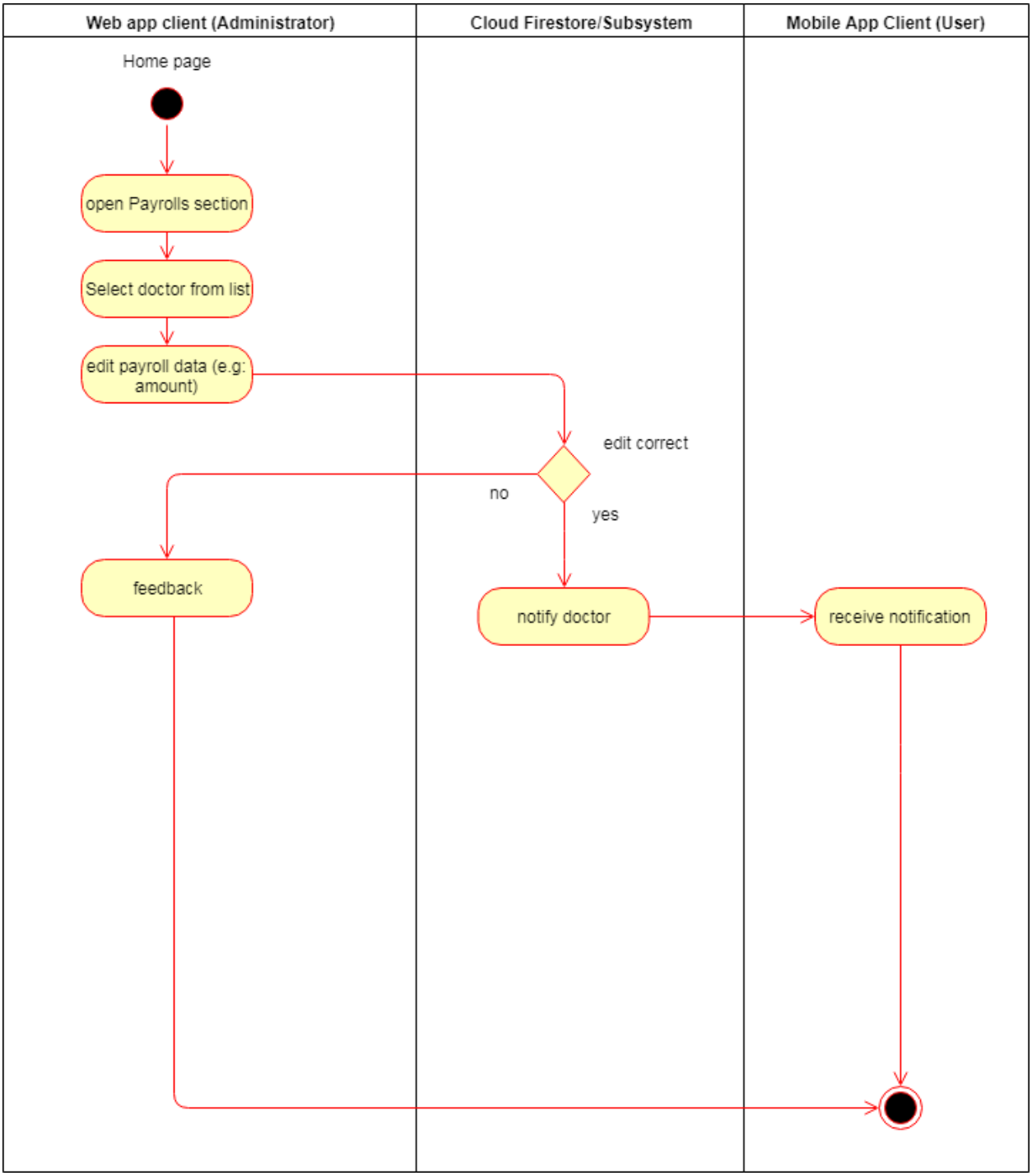
4.3.2.8. Edit Service



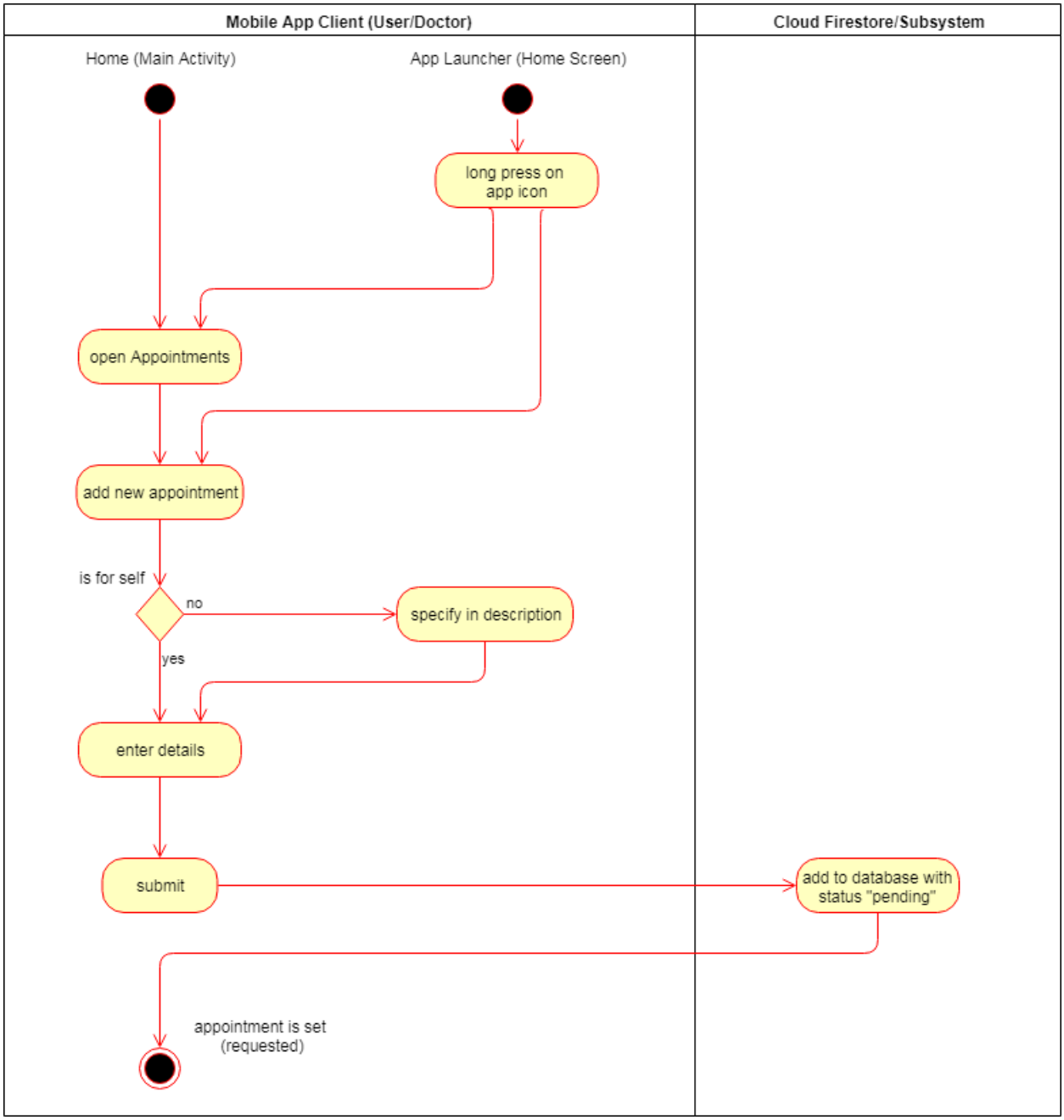
4.3.2.9. Manage users



4.3.2.10. Payroll management

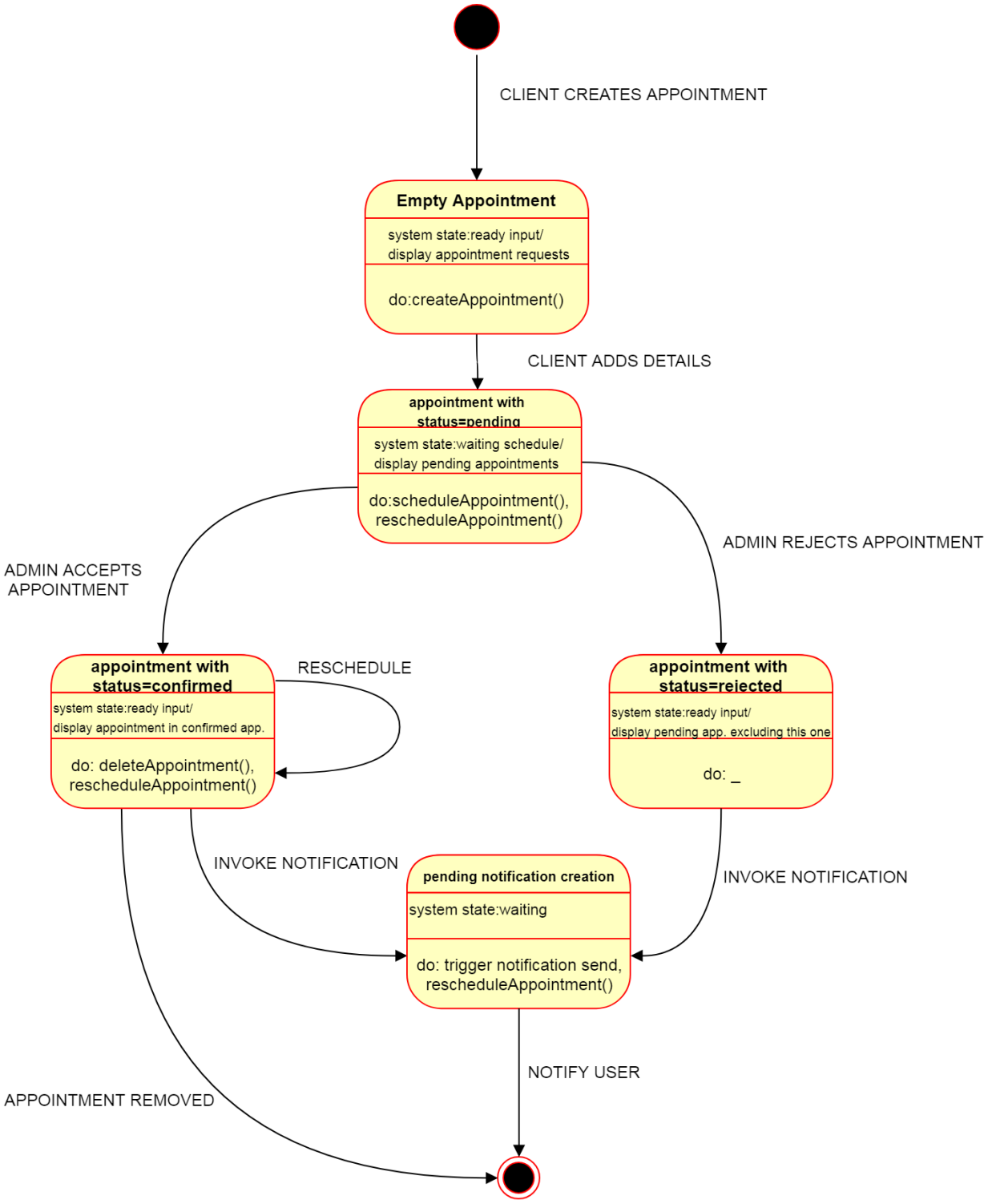


4.3.2.11. Set appointment

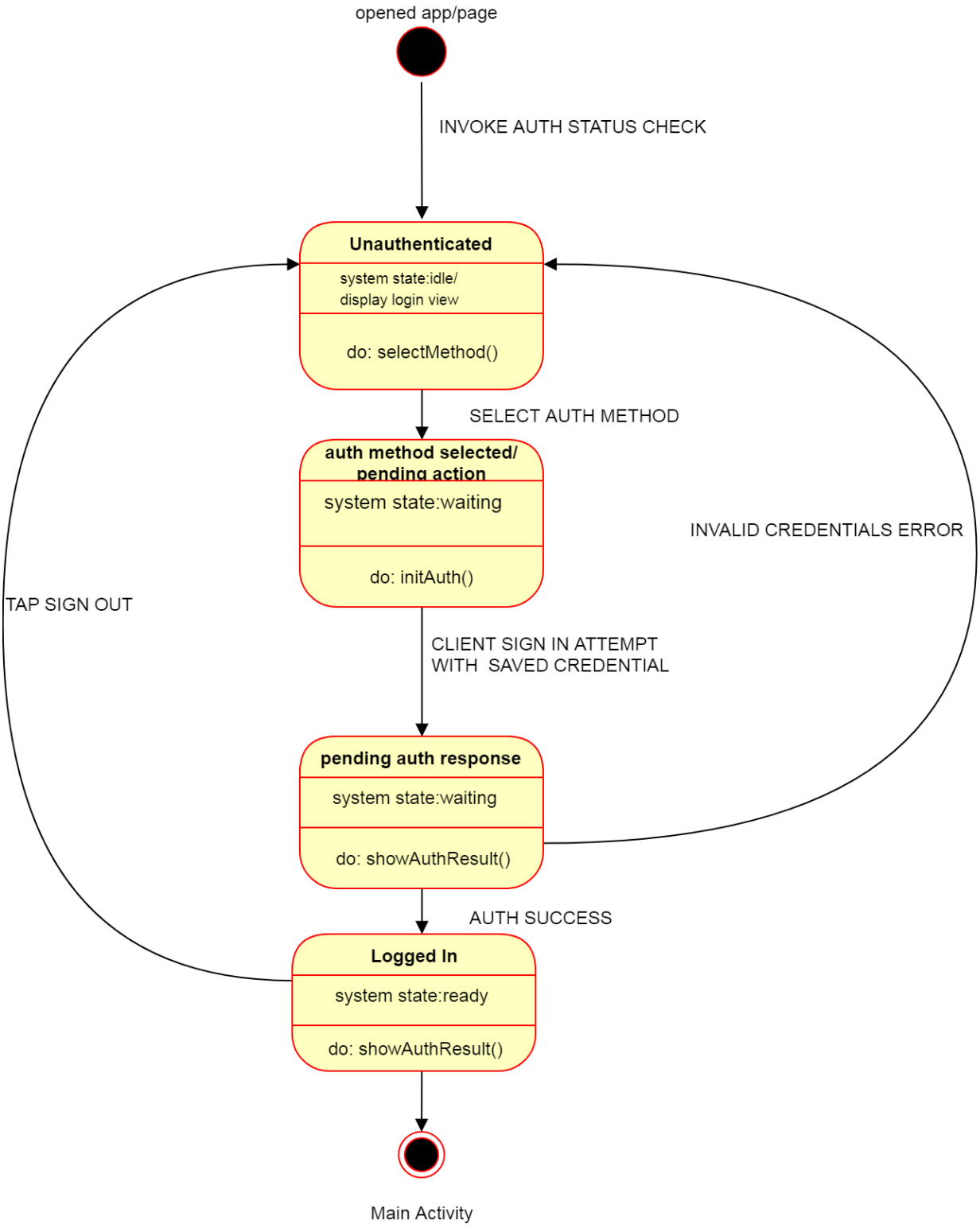


4.3.3. State Diagrams

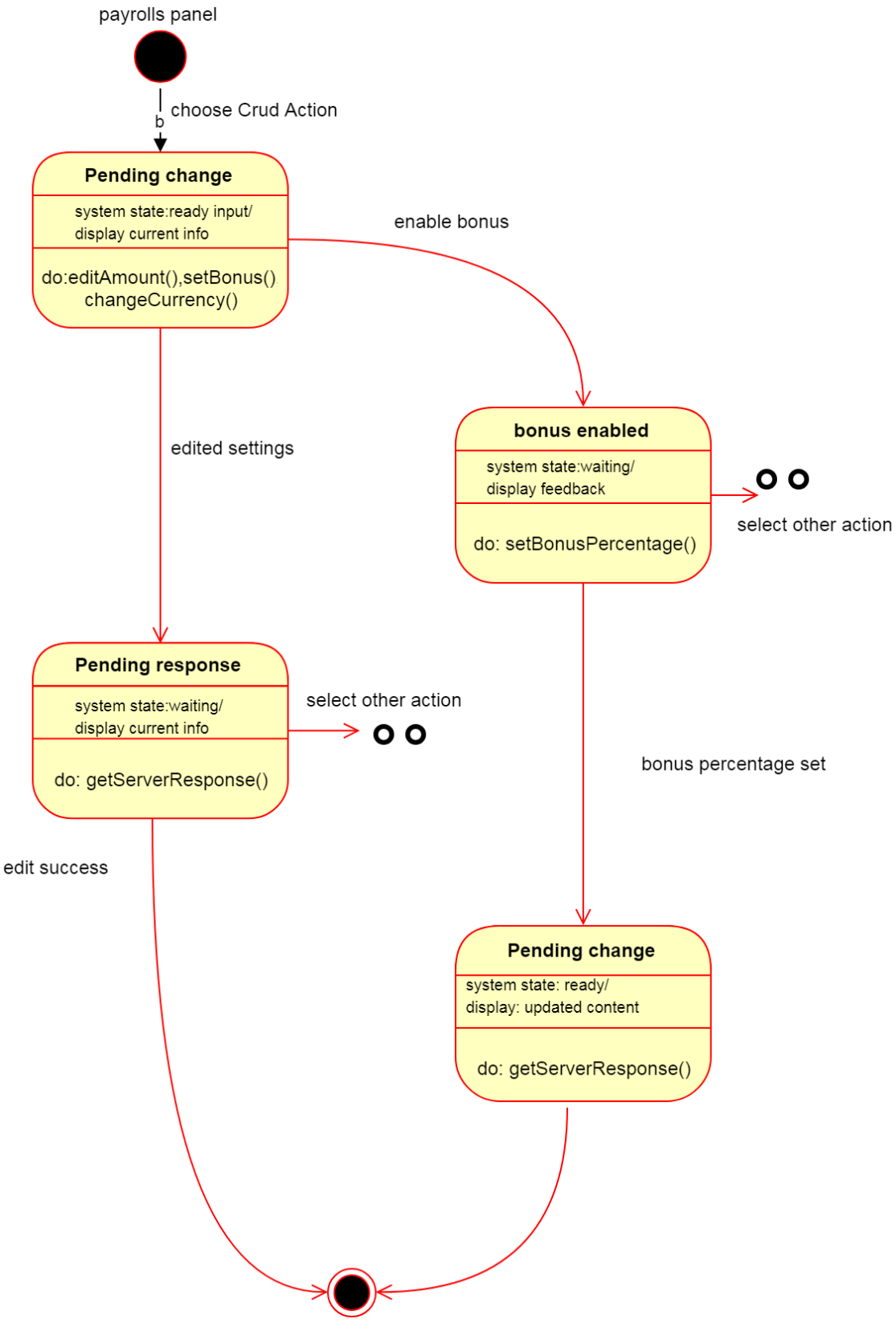
4.3.3.1. Appointments



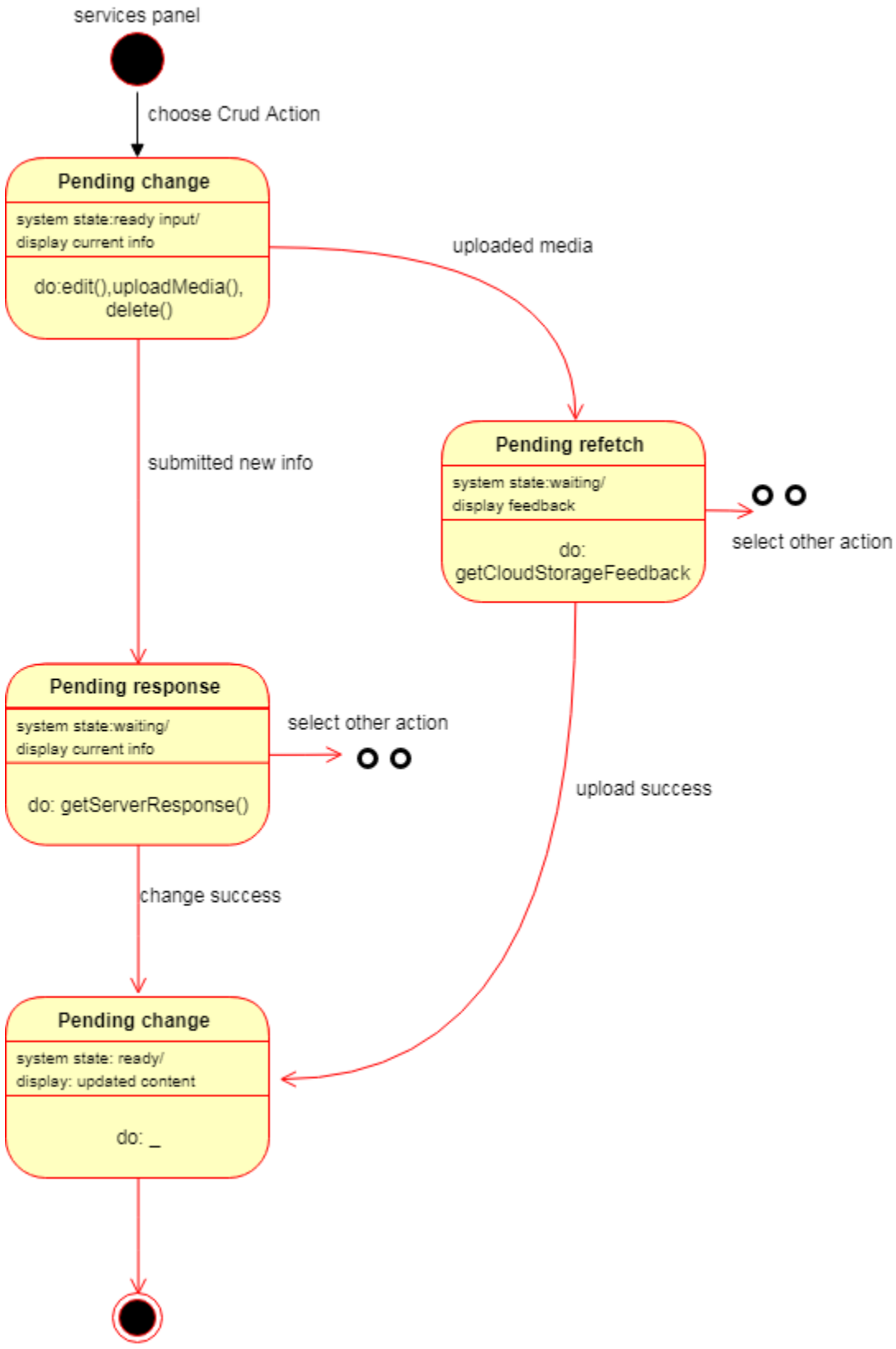
4.3.3.2. Authentication



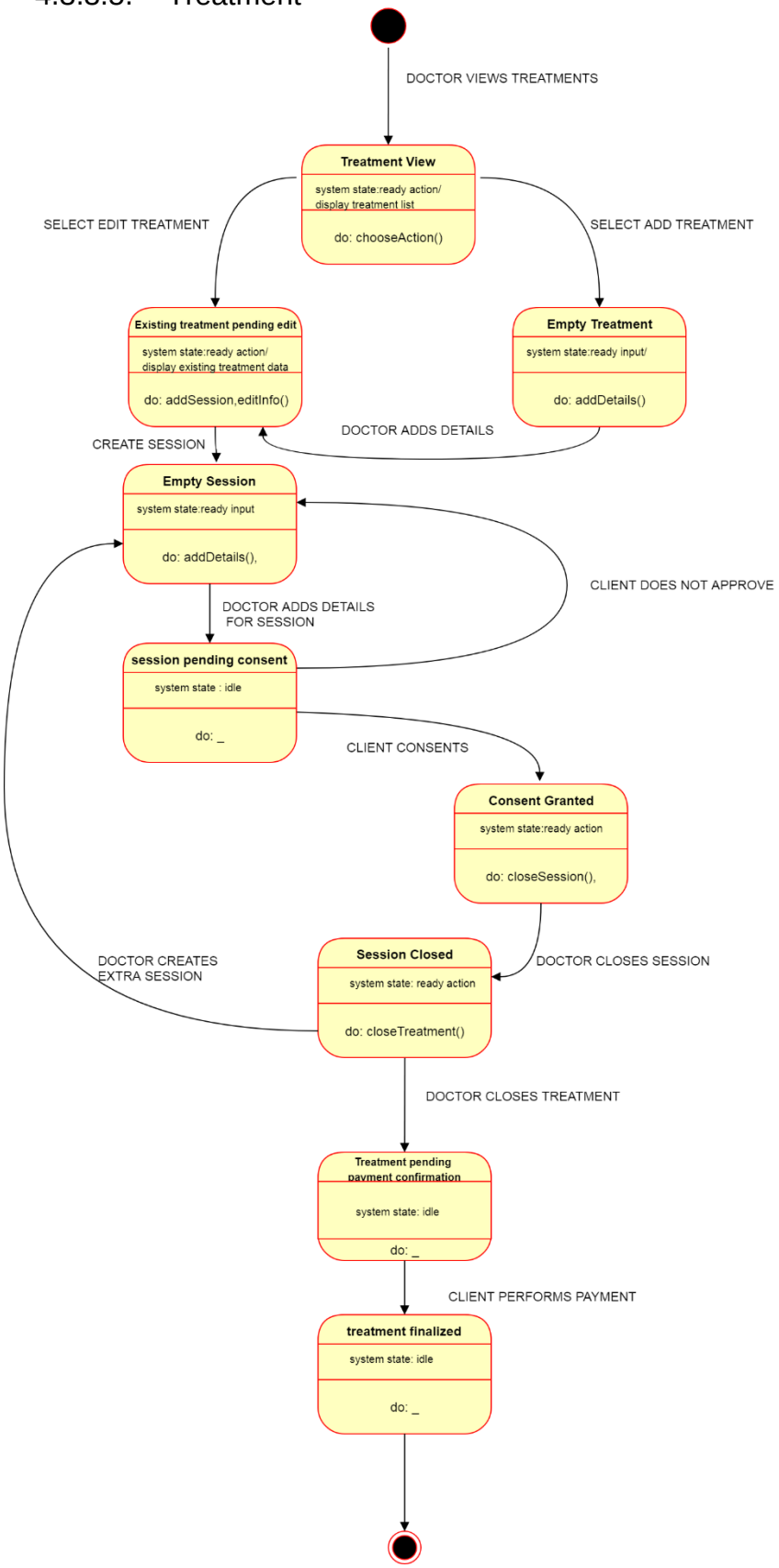
4.3.3.3. Payroll



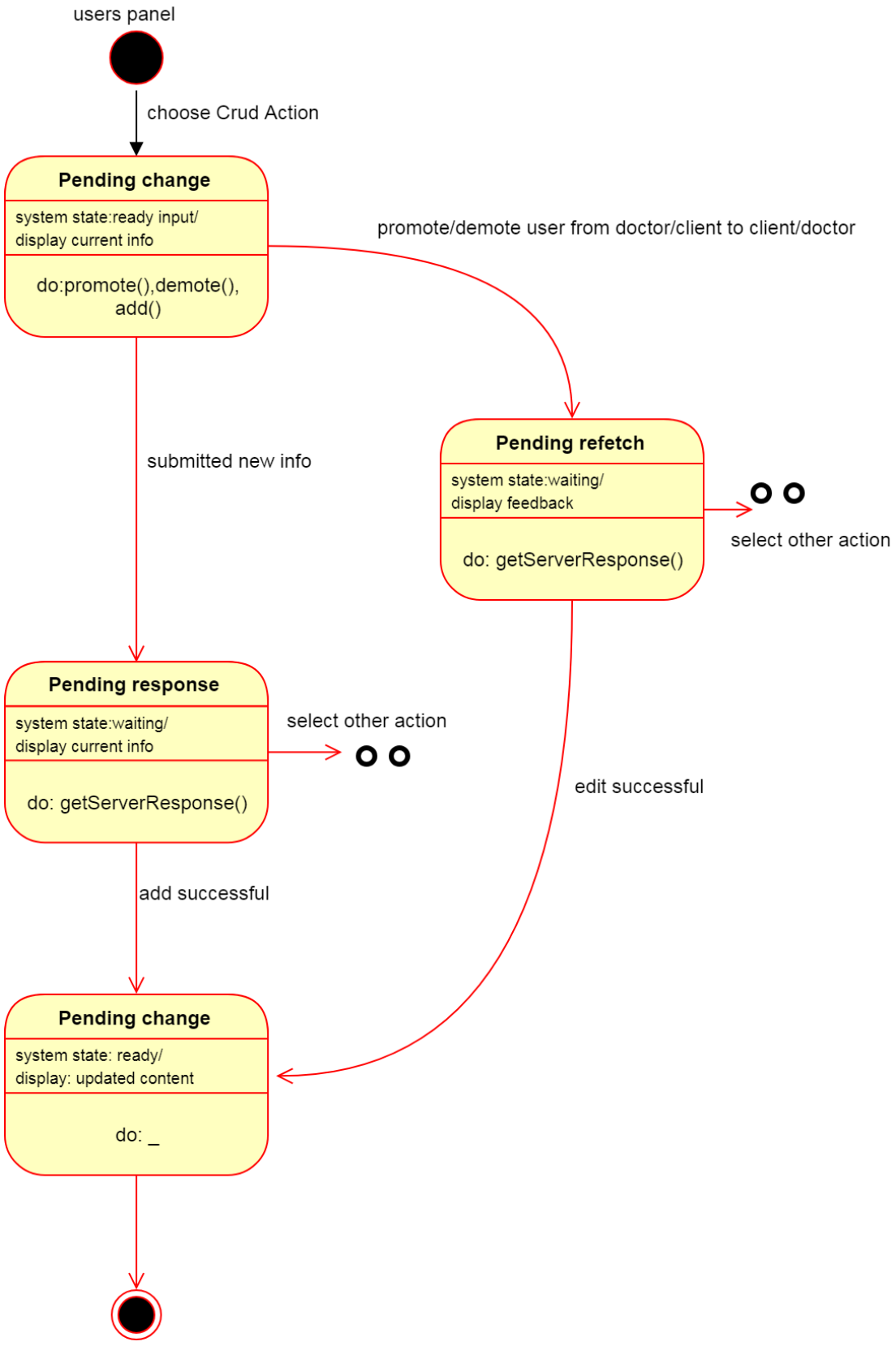
4.3.3.4. Services



4.3.3.5. Treatment

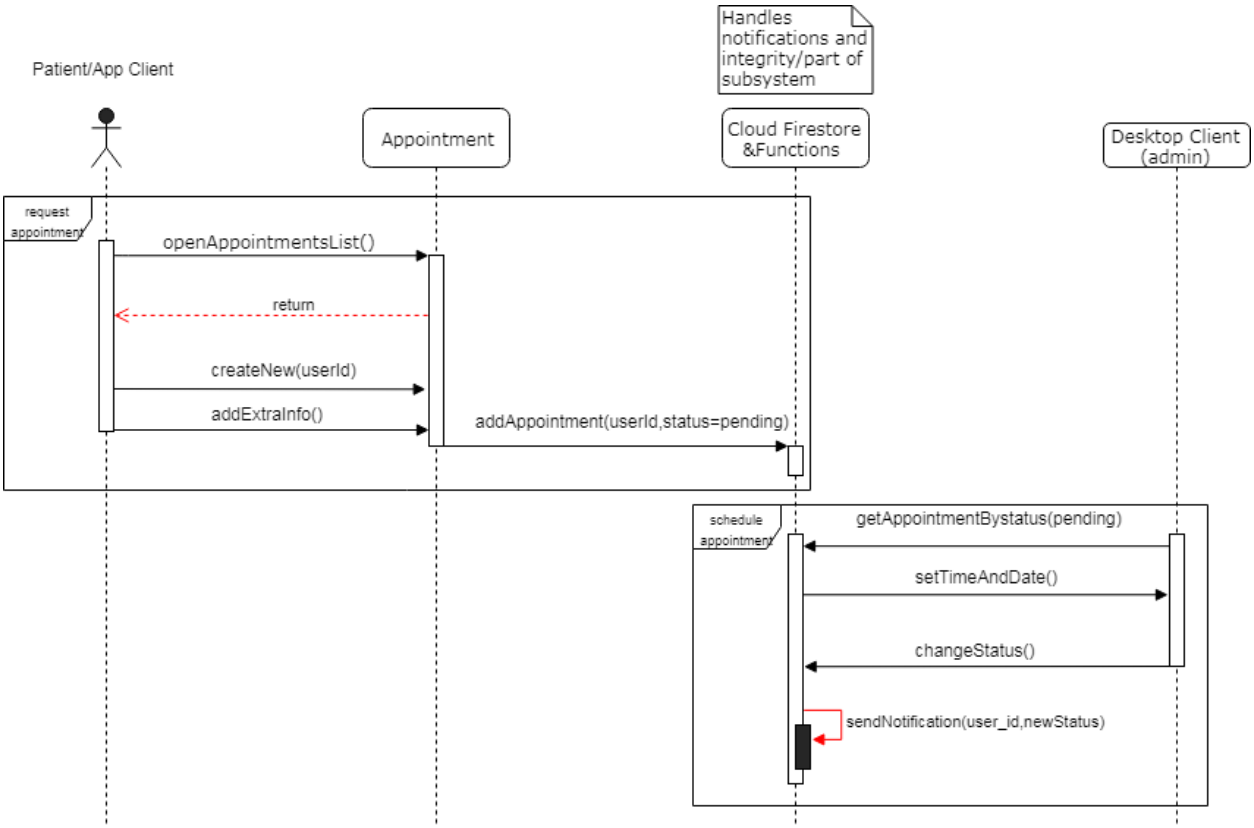


4.3.3.6. Users

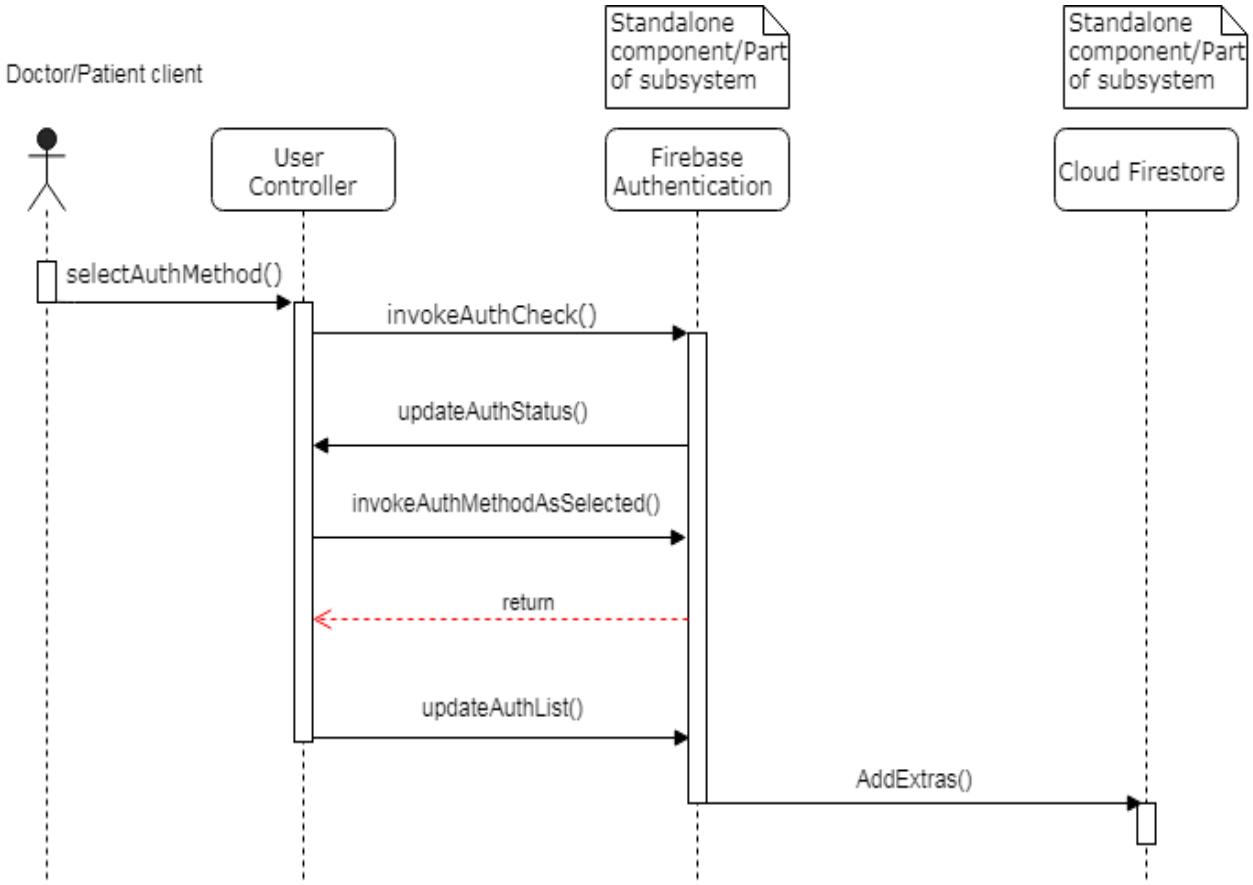

users panel
choose Crud Action
Pending change
system state:ready input/
display current info
do:promote(),demote(),
add()
promote/demote user from doctor/client to client/doctor
submitted new info
Pending refetch
system state:waiting/
display feedback
do: getServerResponse()
select other action
Pending response
system state:waiting/
display current info
do: getServerResponse()
select other action
edit successful
add successful
Pending change
system state: ready/
display: updated content
do: _

4.3.4. Sequence Diagrams

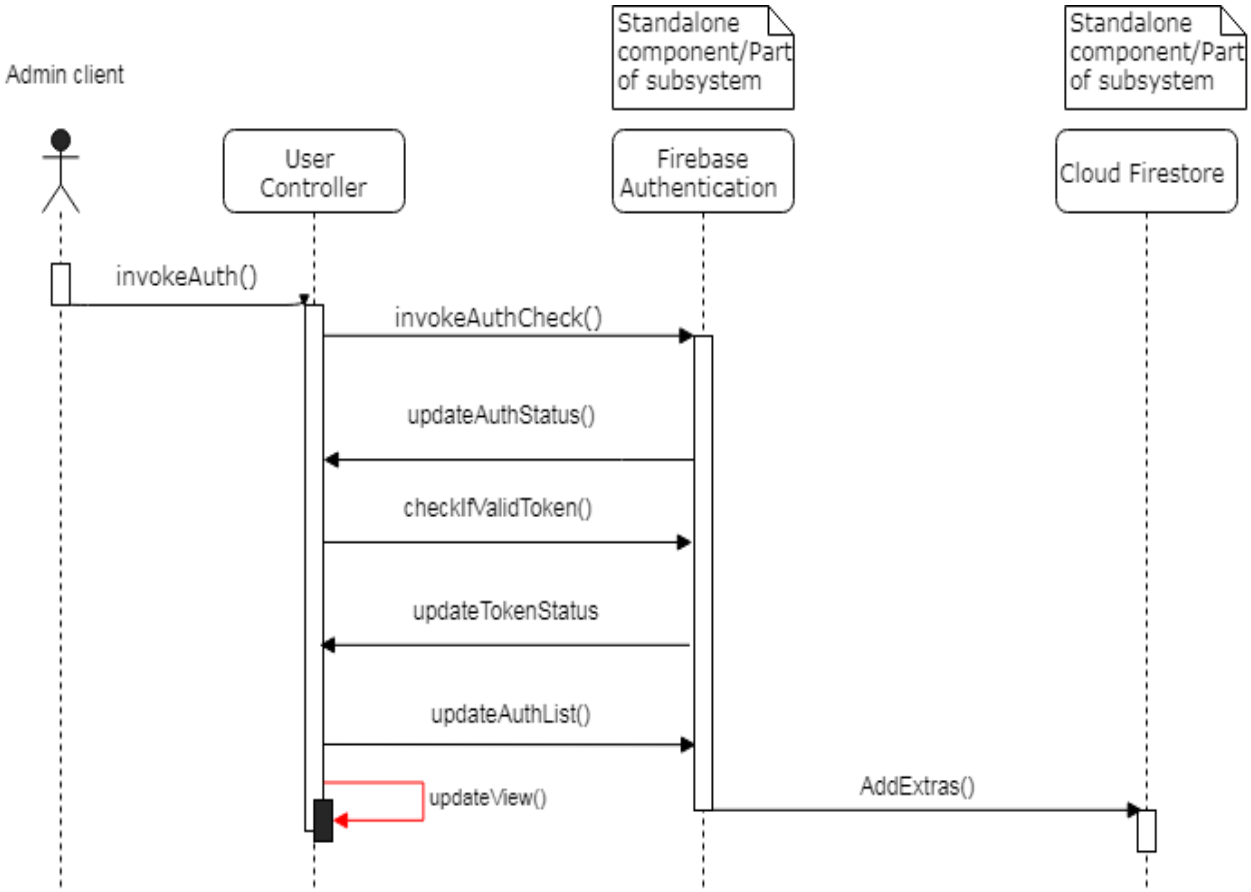
4.3.4.1. Appointments



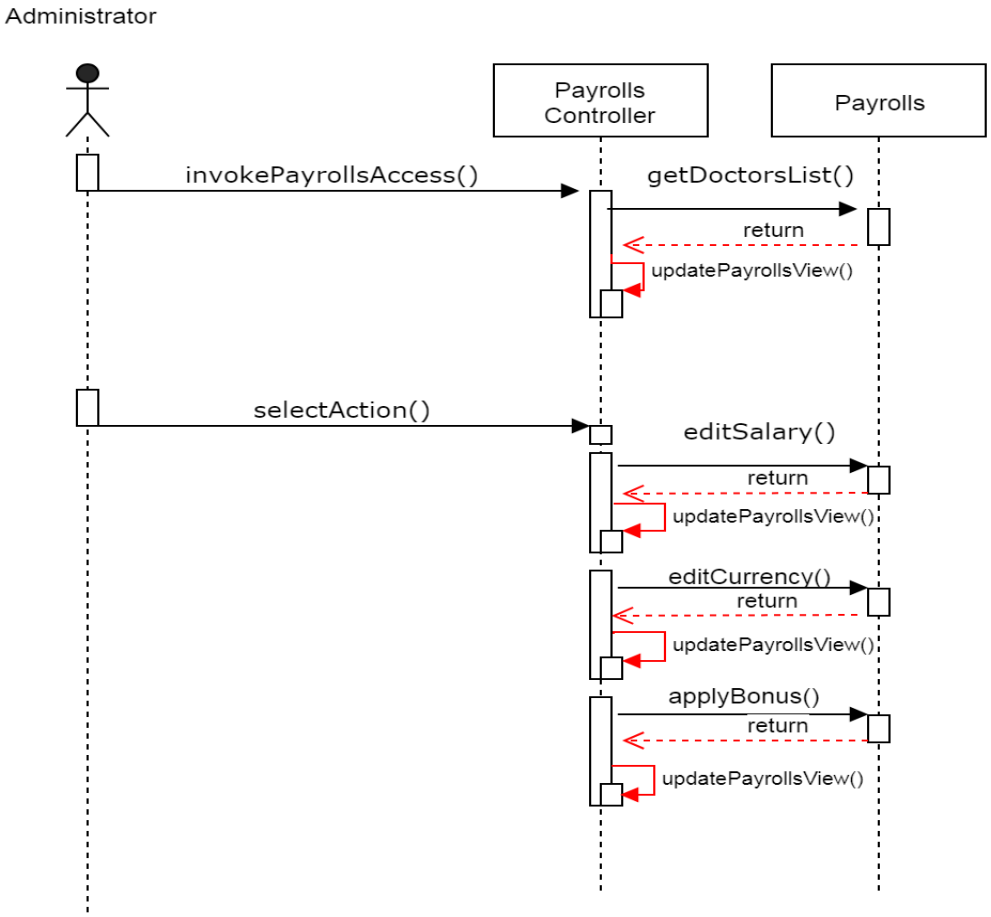
4.3.4.2. Mobile Authentication



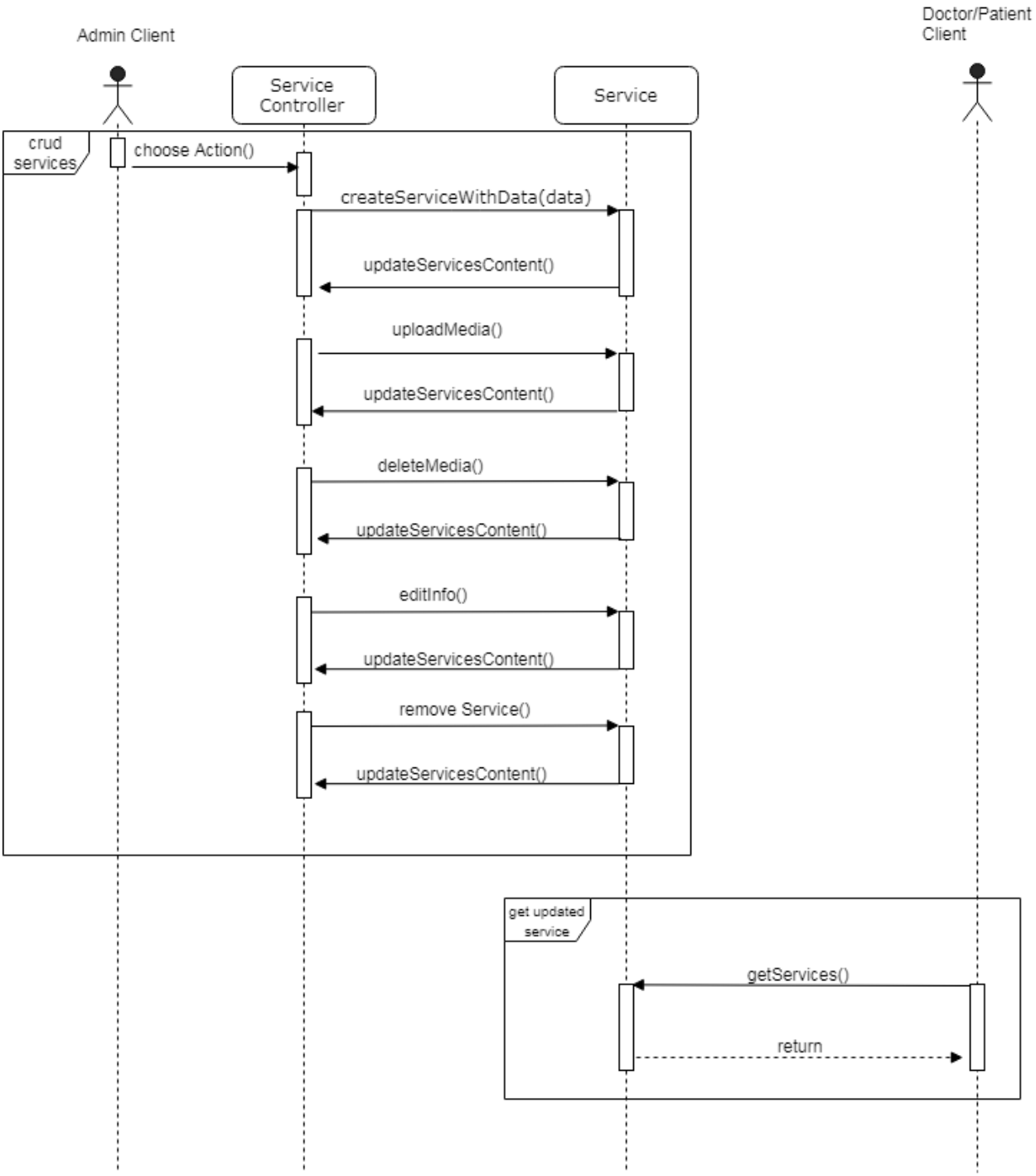
4.3.4.3. Web authentication



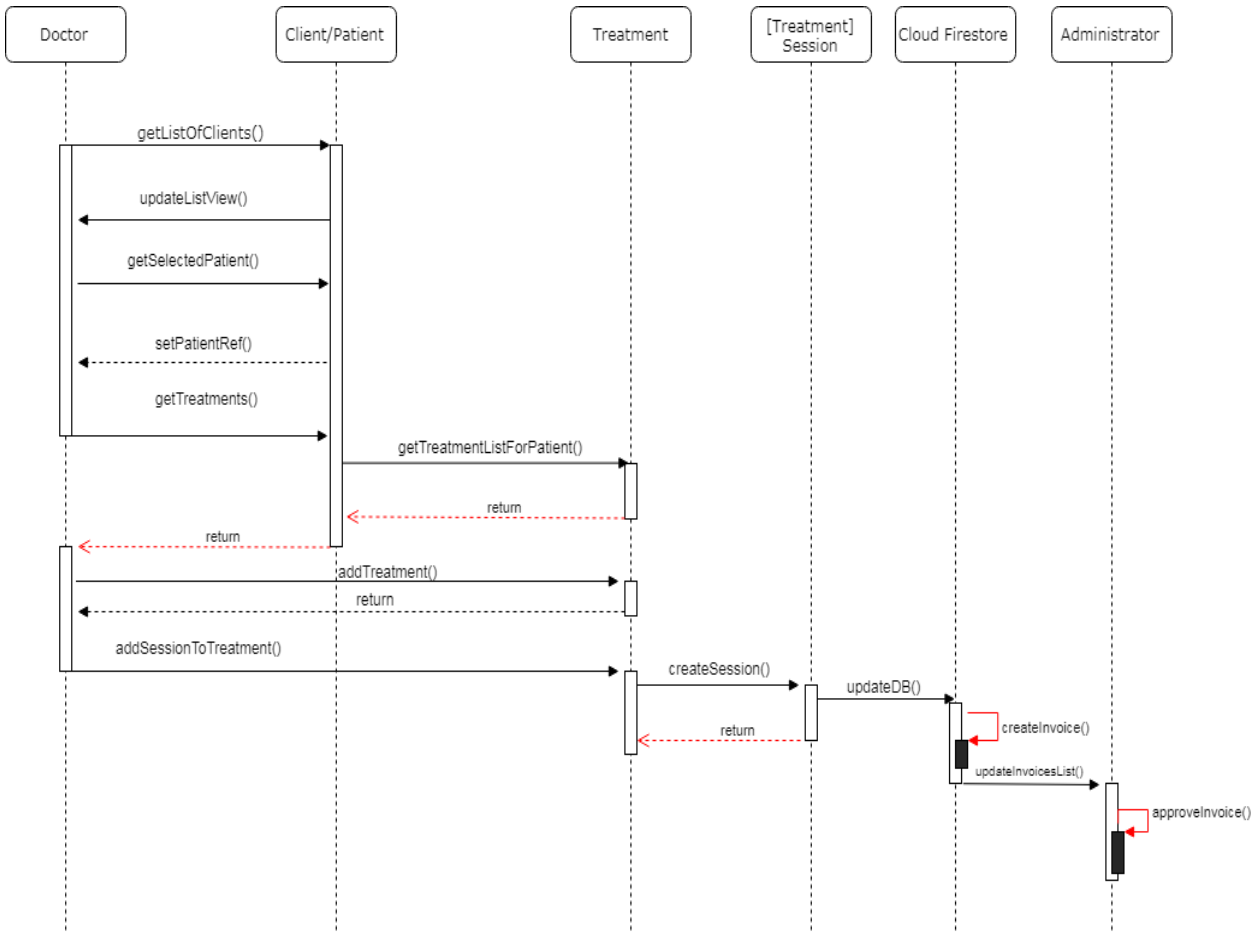
4.3.4.4. Payroll



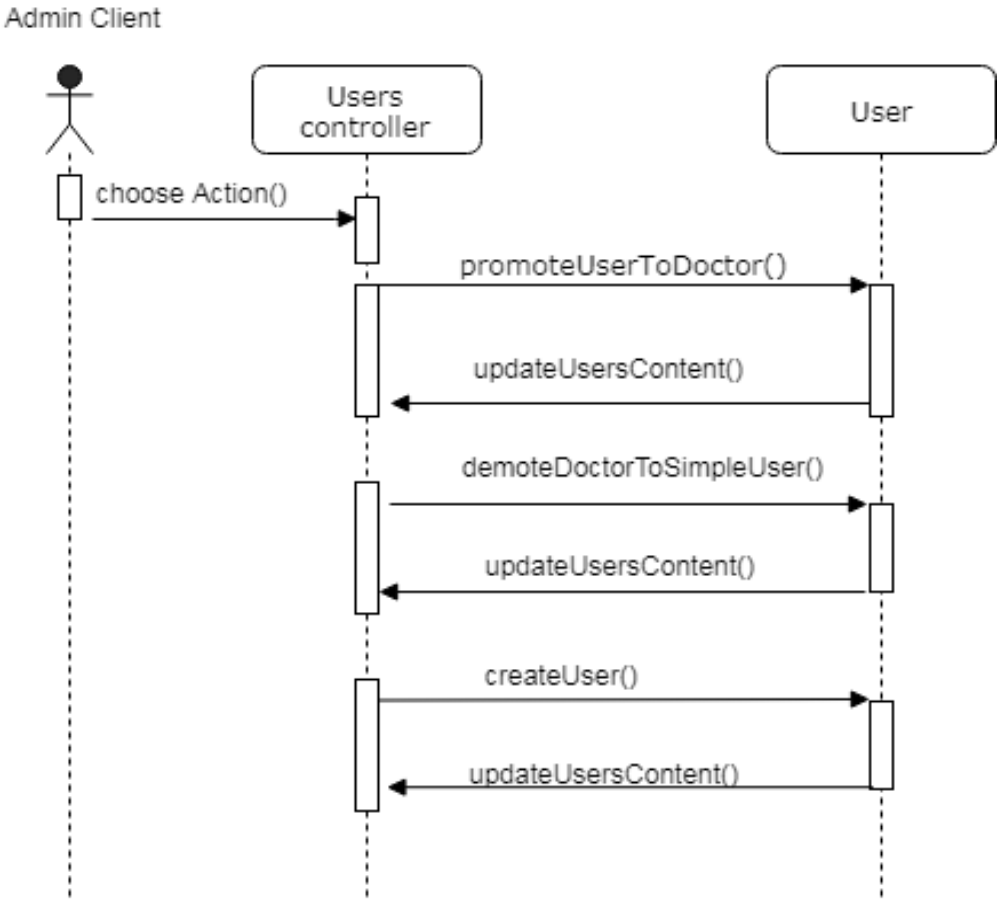
4.3.4.5. Services



4.3.4.6. Treatments



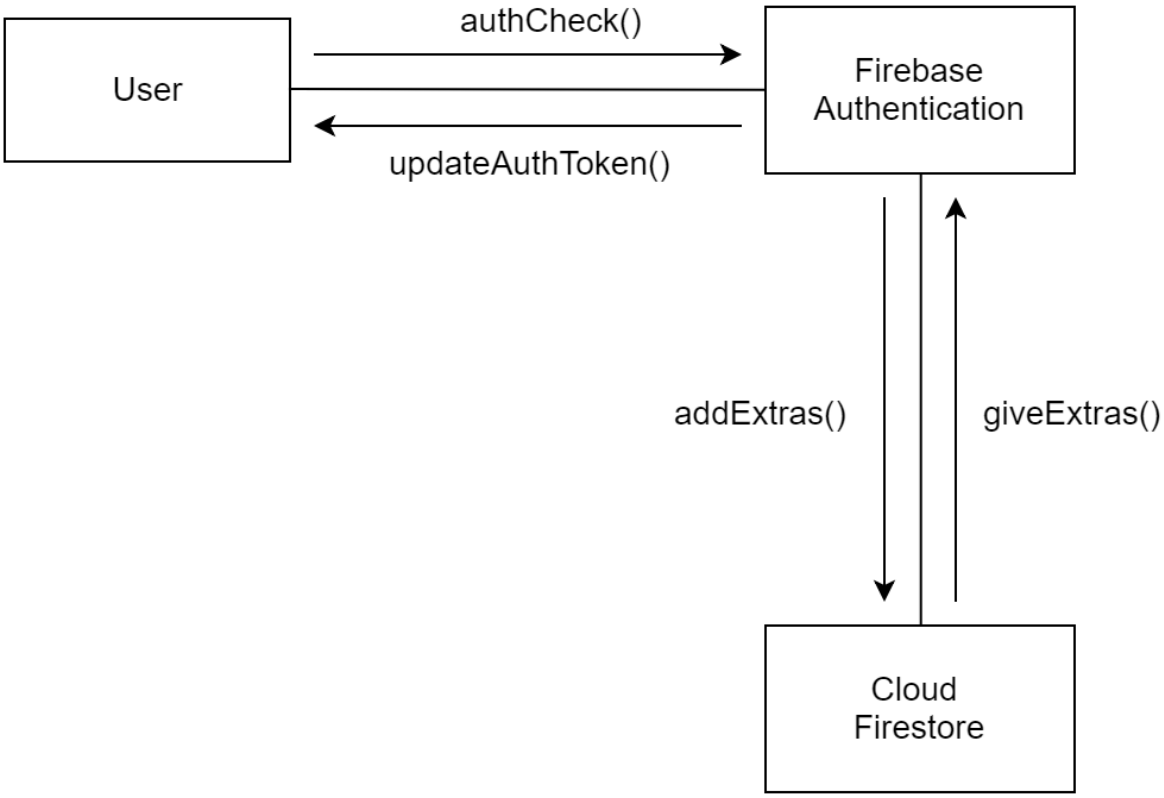
4.3.4.7. Users


Admin Client
Users controller
User
choose Action()
promoteUserToDoctor()
updateUsersContent()
demoteDoctorToSimpleUser()
updateUsersContent()
createUser()
updateUsersContent()

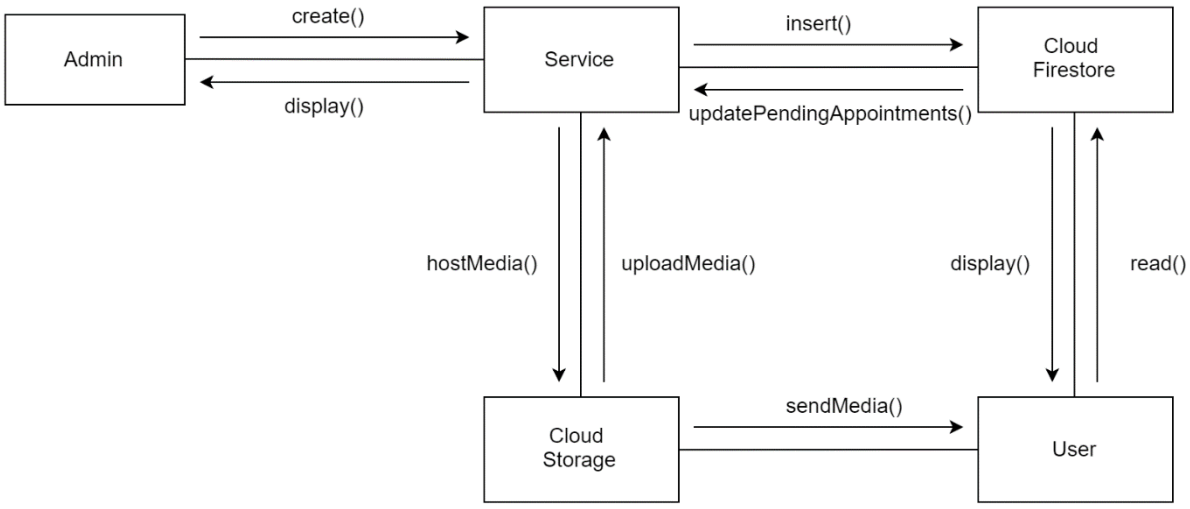
4.3.5. Collaboration Diagrams

4.3.5.1. Appointments

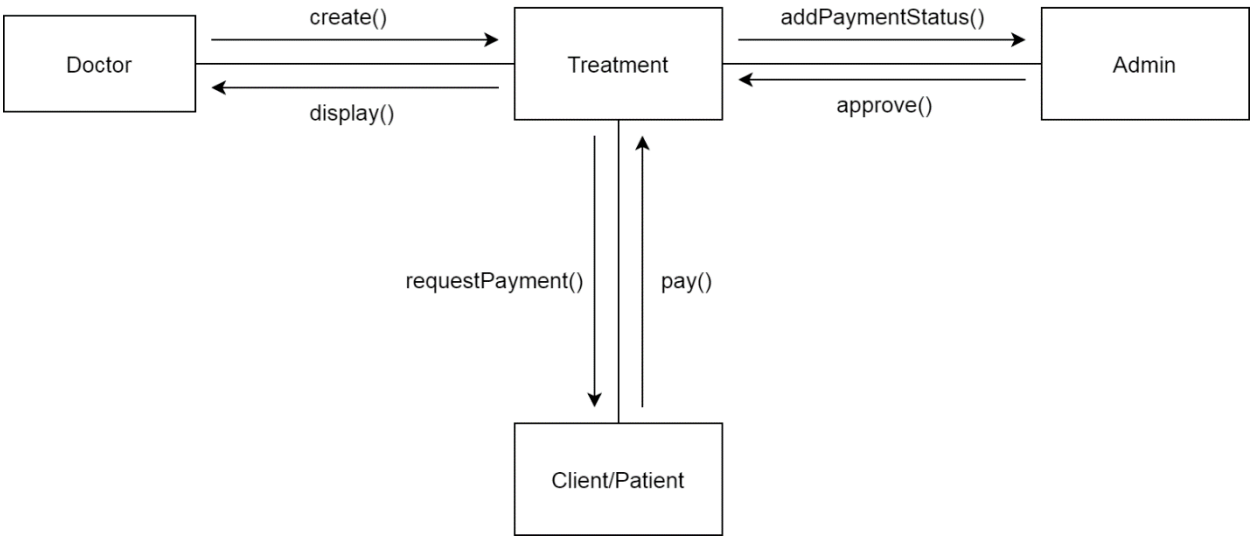
4.3.5.2. Authentication



4.3.5.3. Services

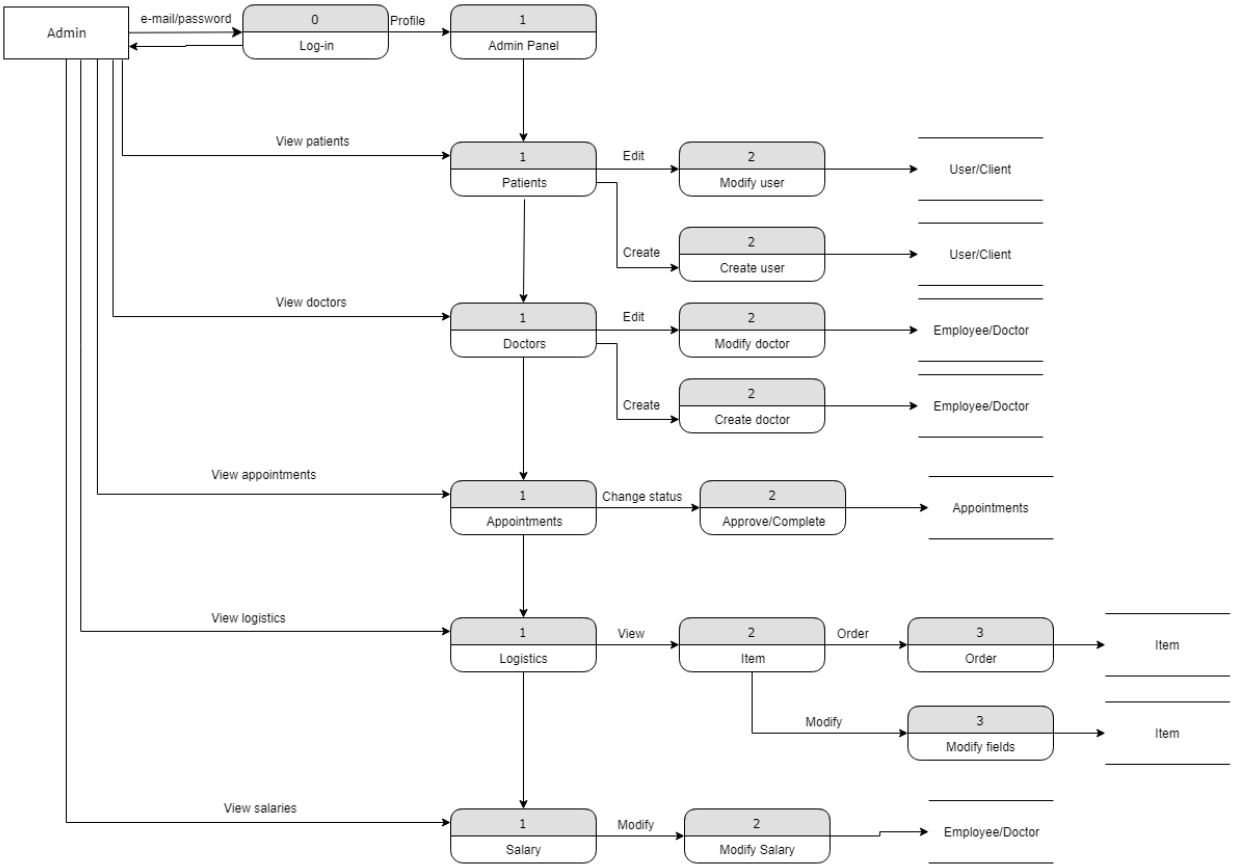


4.3.5.4. Treatments

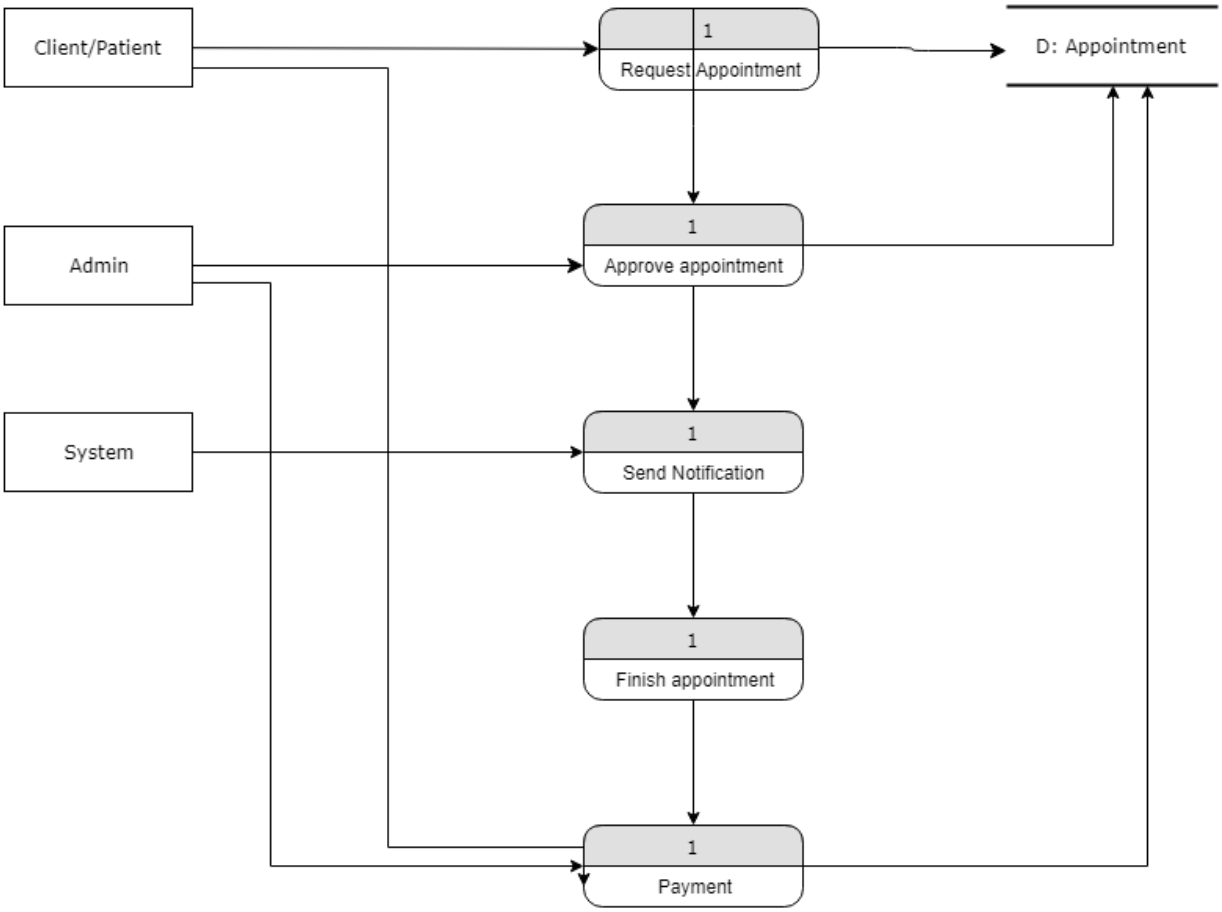


4.4. Data Flow Diagram (DFD)

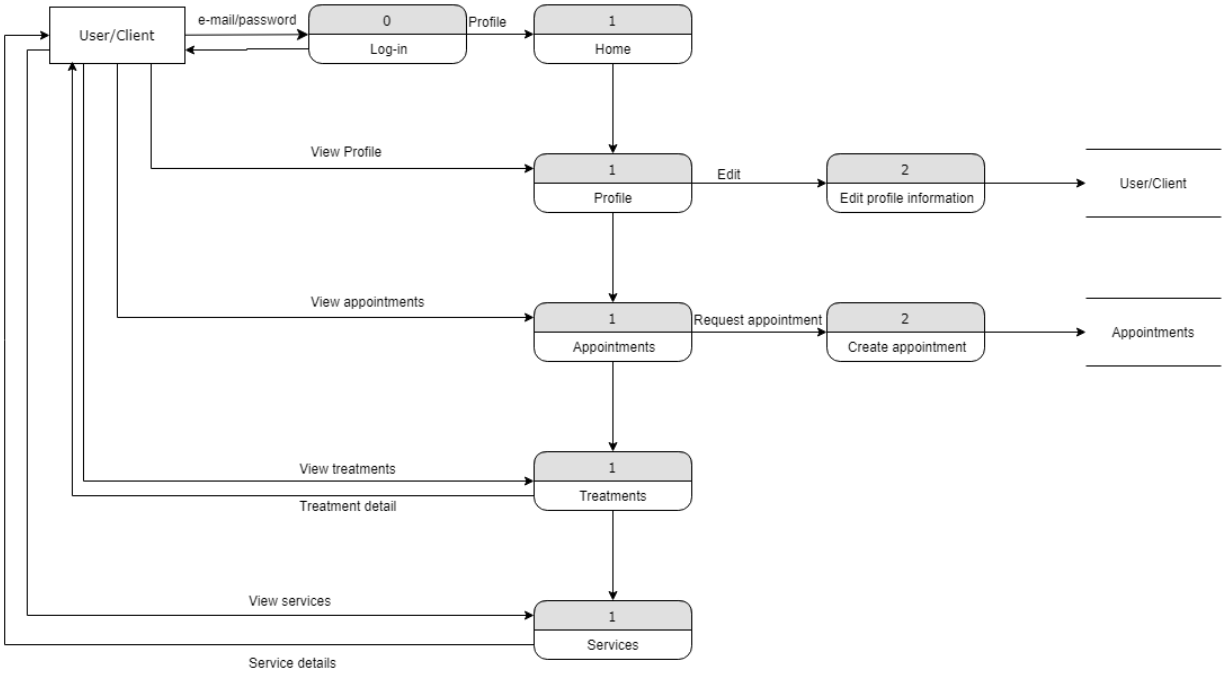
4.4.1. Admin



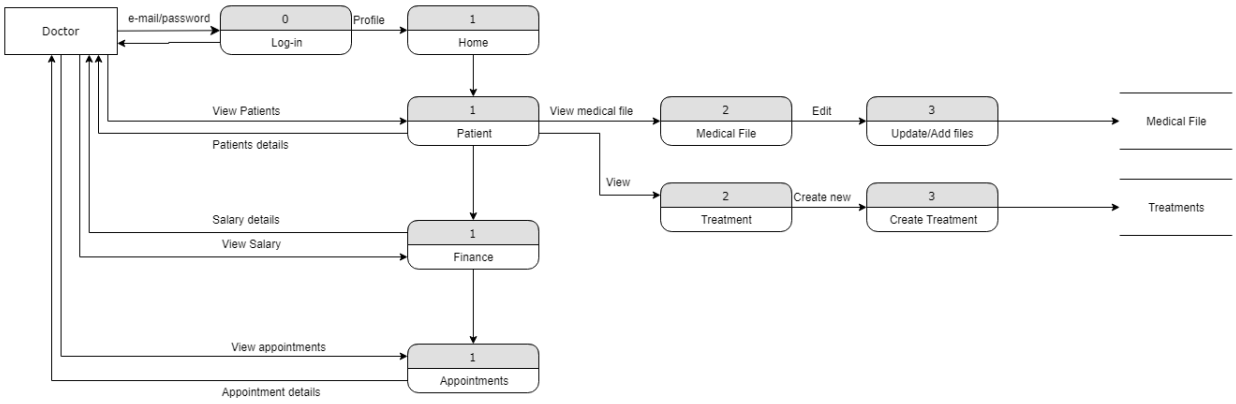
4.4.2. Appointments



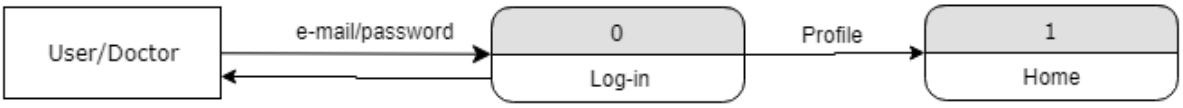
4.4.3. Client



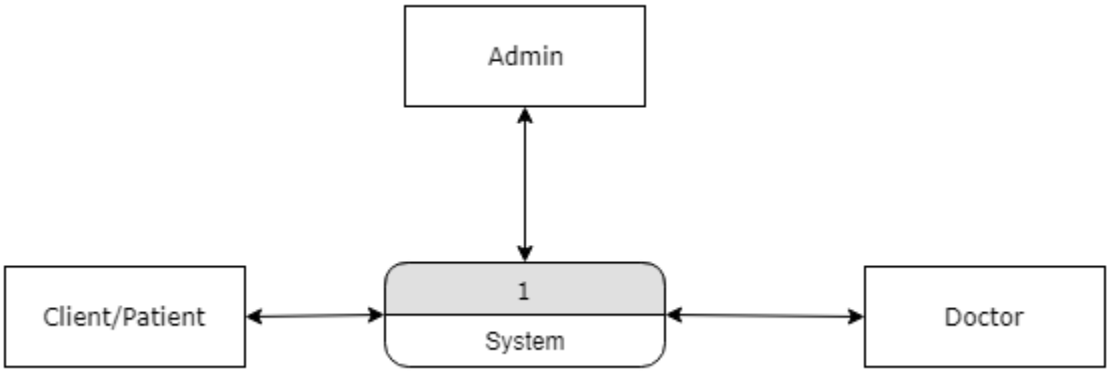
4.4.4. Doctor



4.4.5. Login



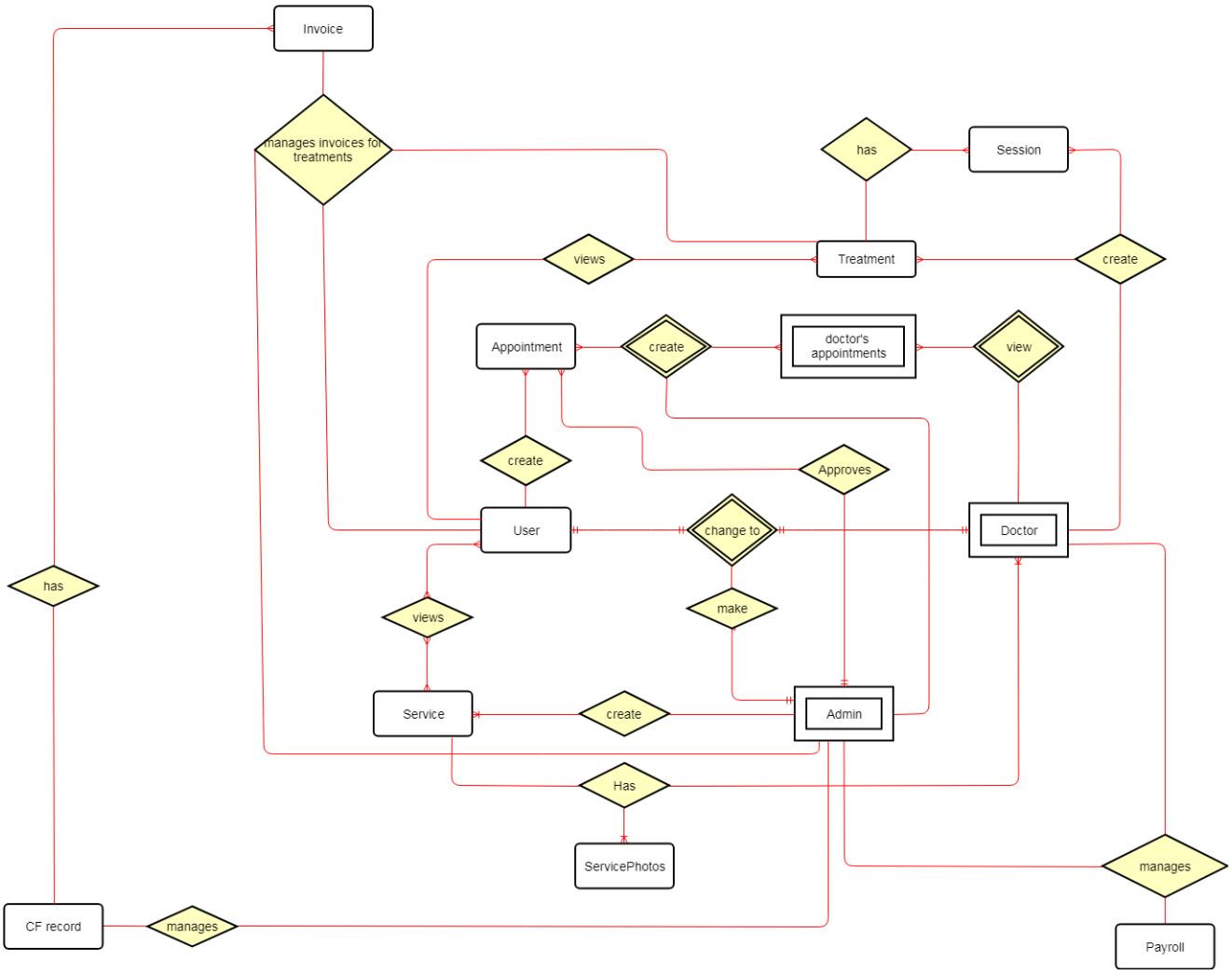
4.4.6. System access



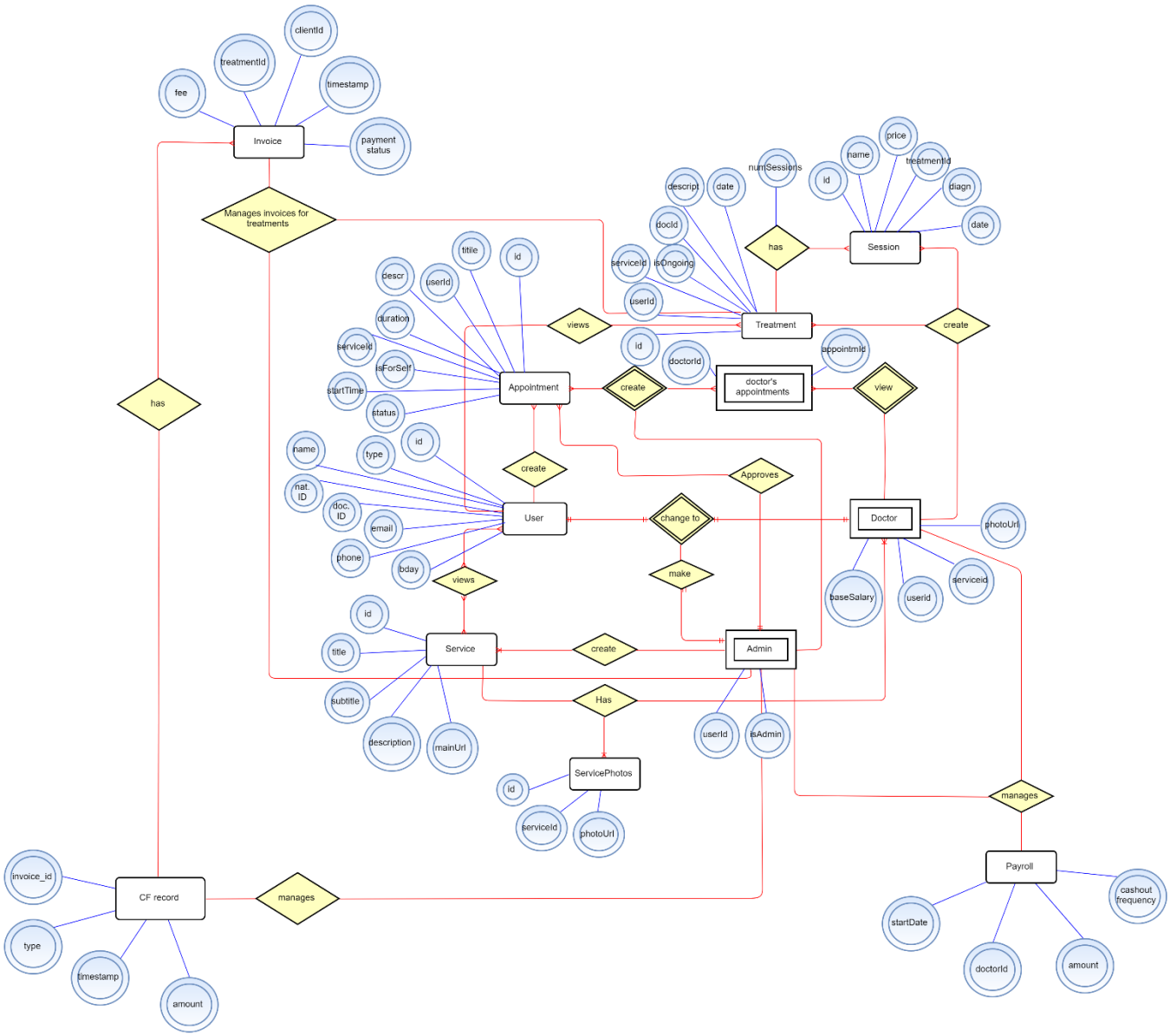
4.5. Entity Relationship Diagram (ERD)

4.5.1. Relational Model

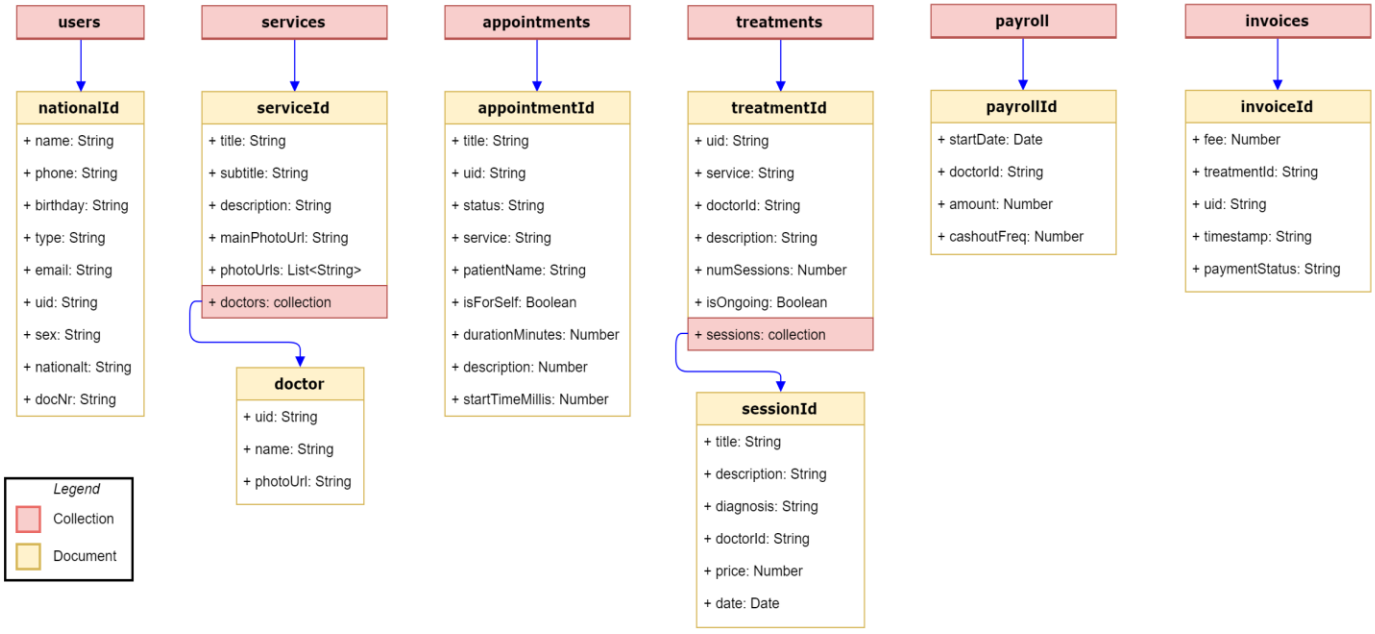
4.5.1.1. Without attributes



4.5.1.2. With attributes

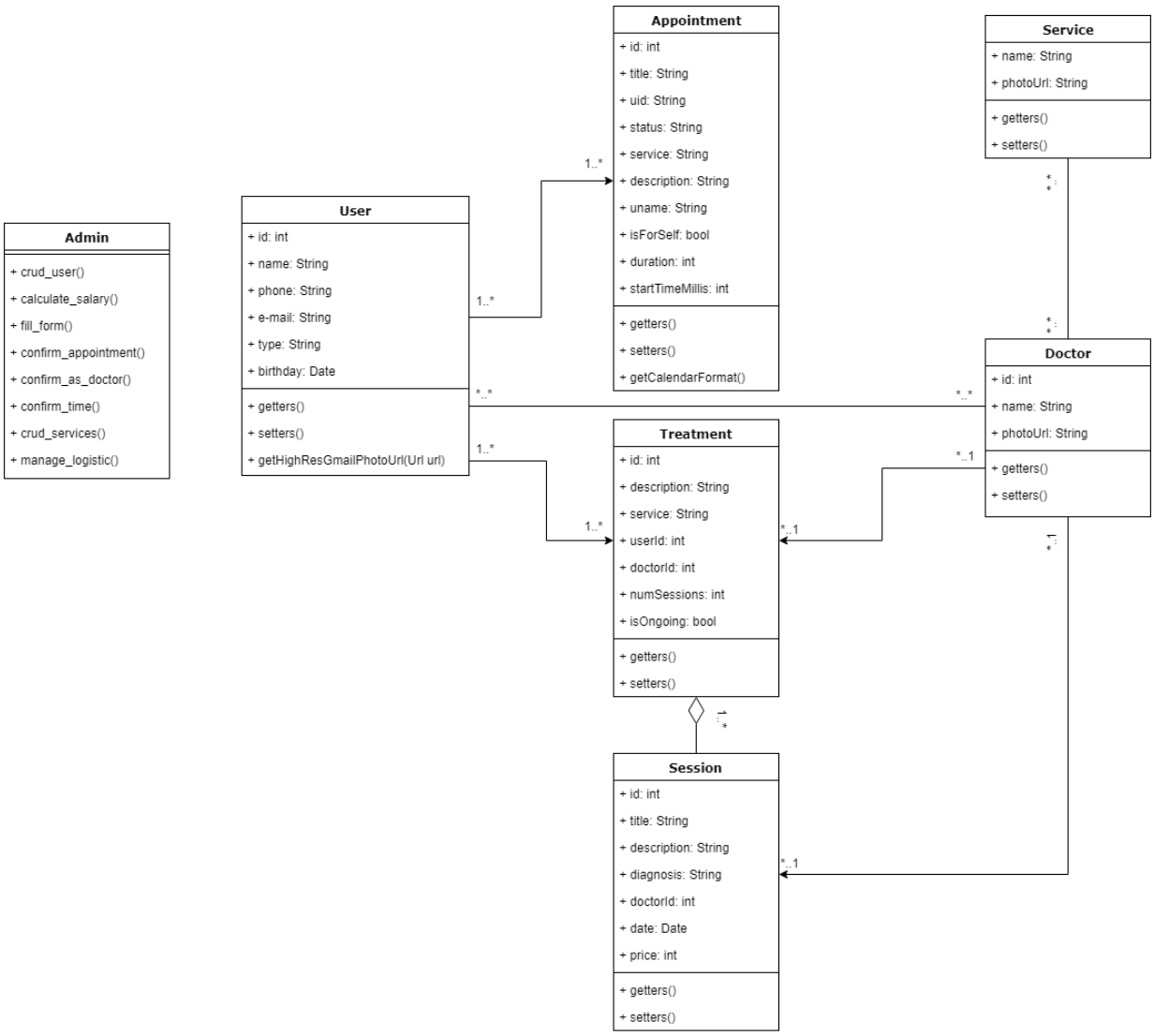


4.5.2. Nonrelational Model



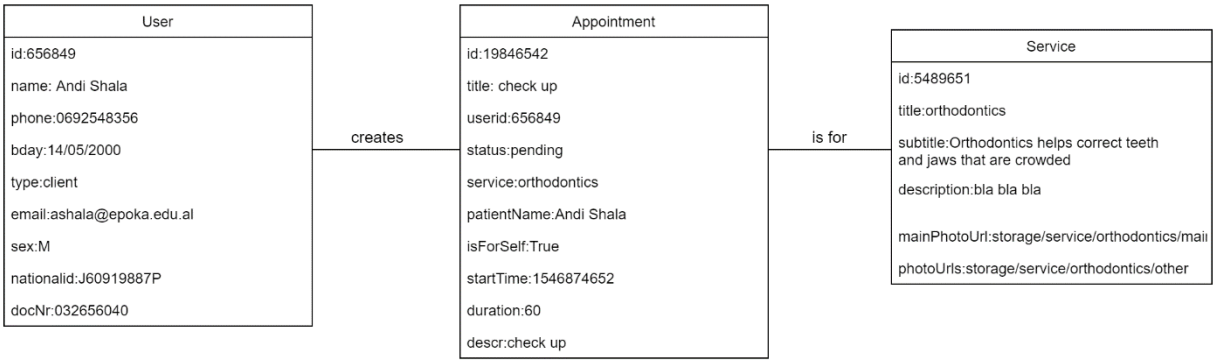
4.6. Structural Diagrams

4.6.1. Class Diagram

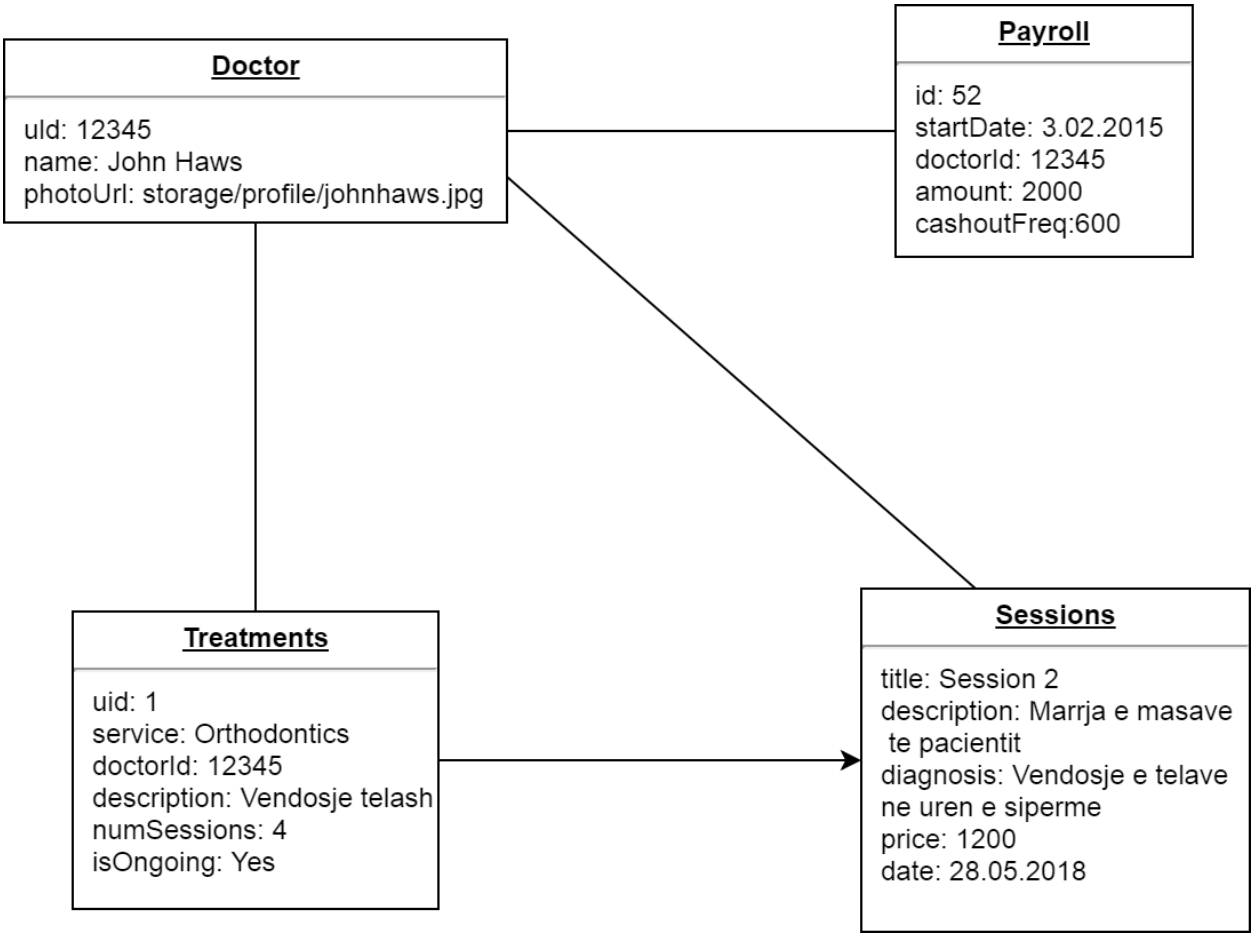


4.6.2. Object Diagrams

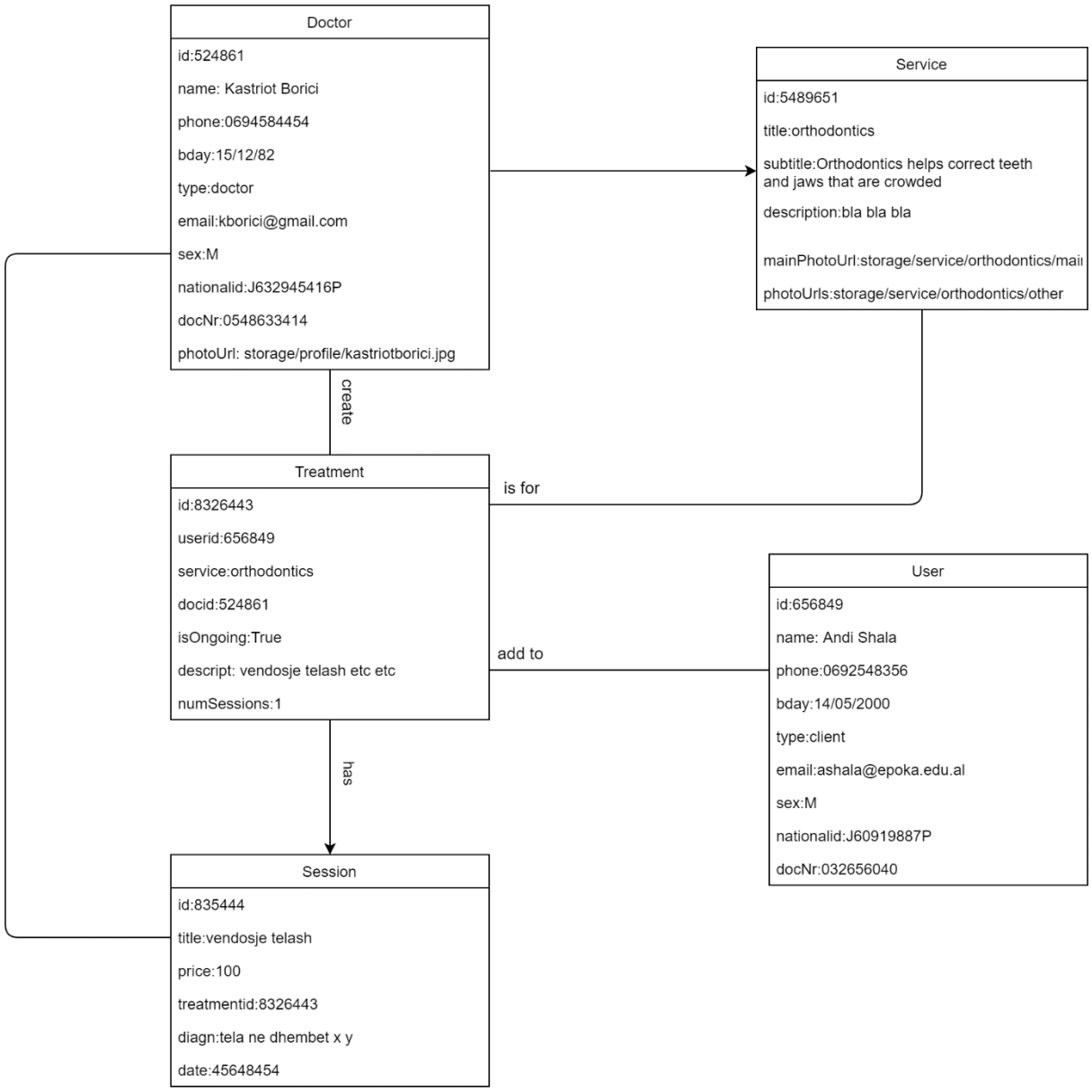
4.6.2.1. Appointment



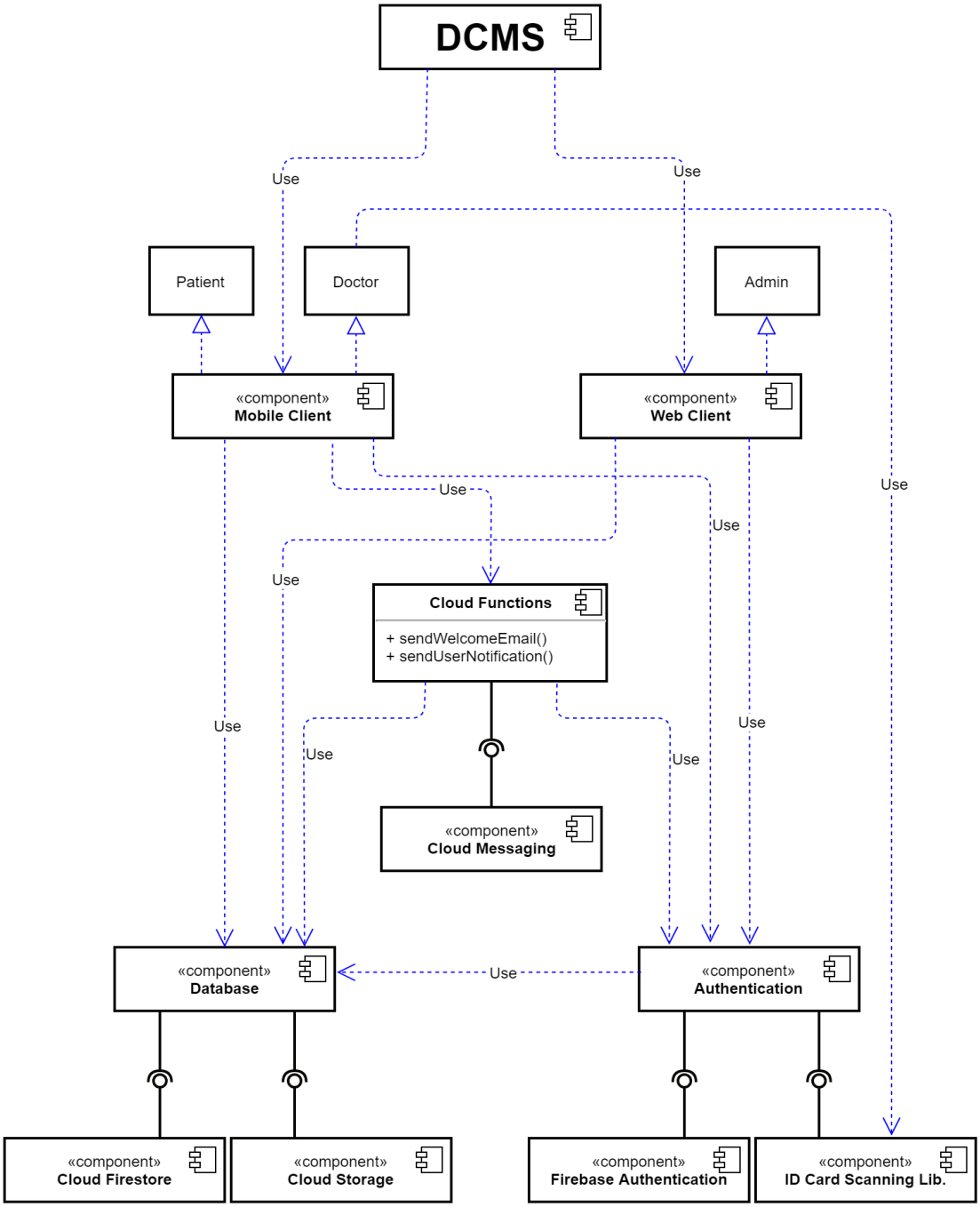
4.6.2.2. Finances



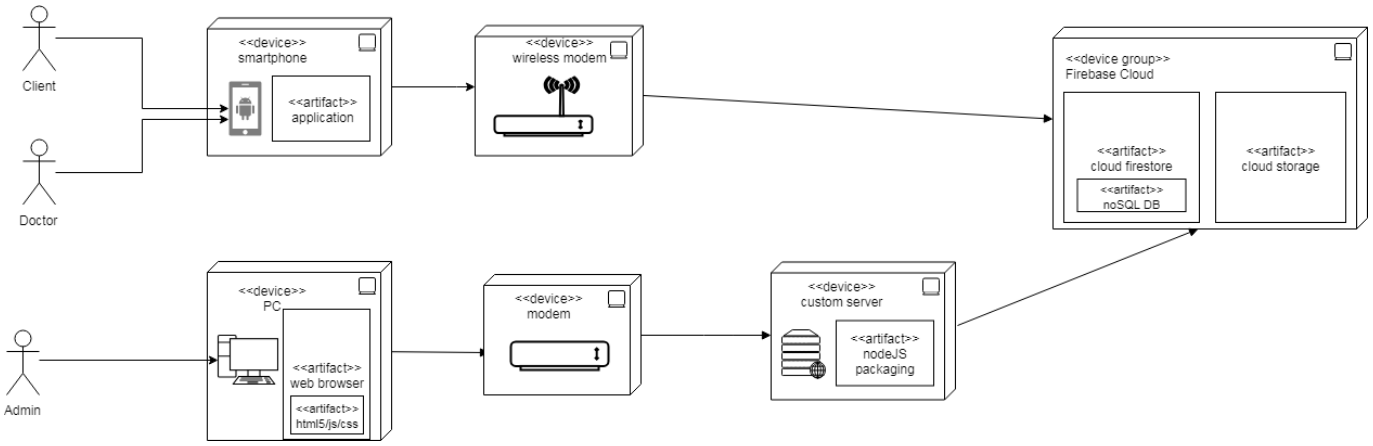
4.6.2.3. Treatment



4.6.3. Component Diagram

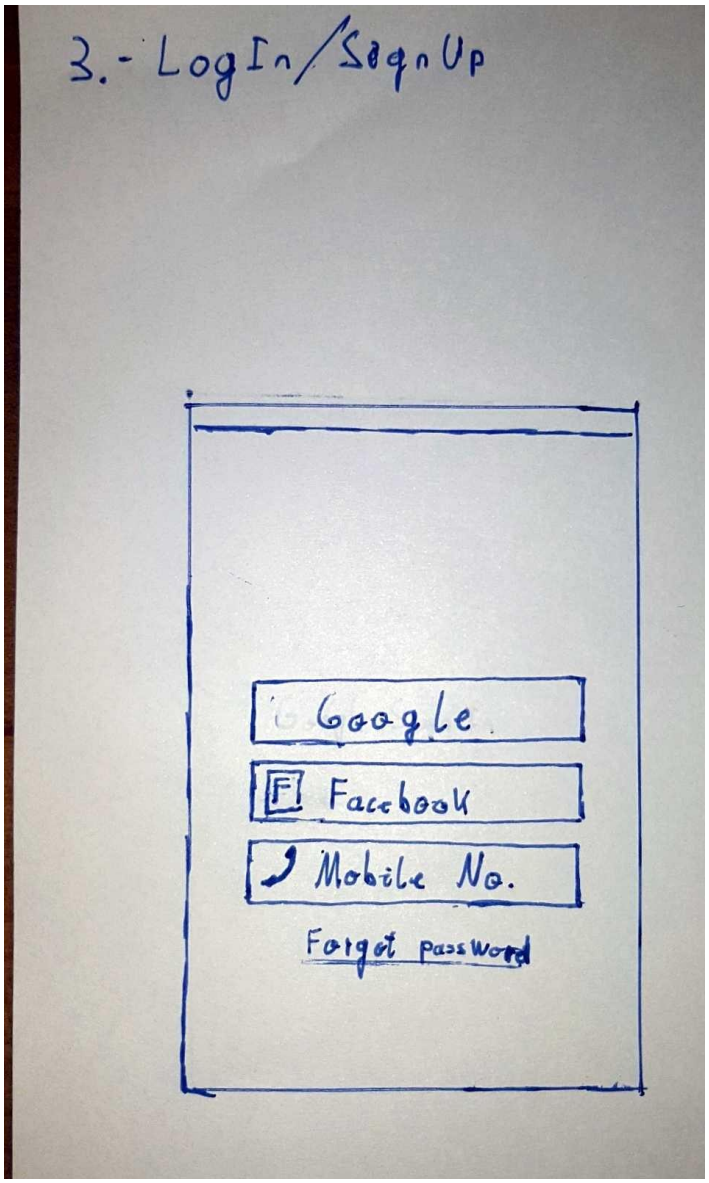


4.6.4. Deployment Diagram

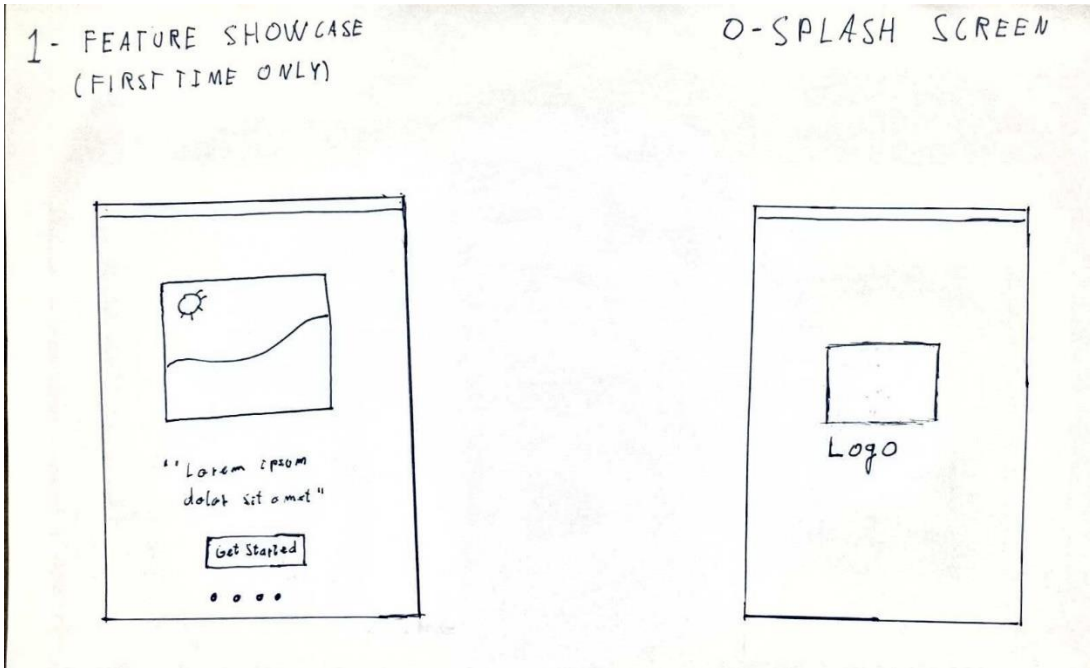


4.7. Mobile app – Sketches

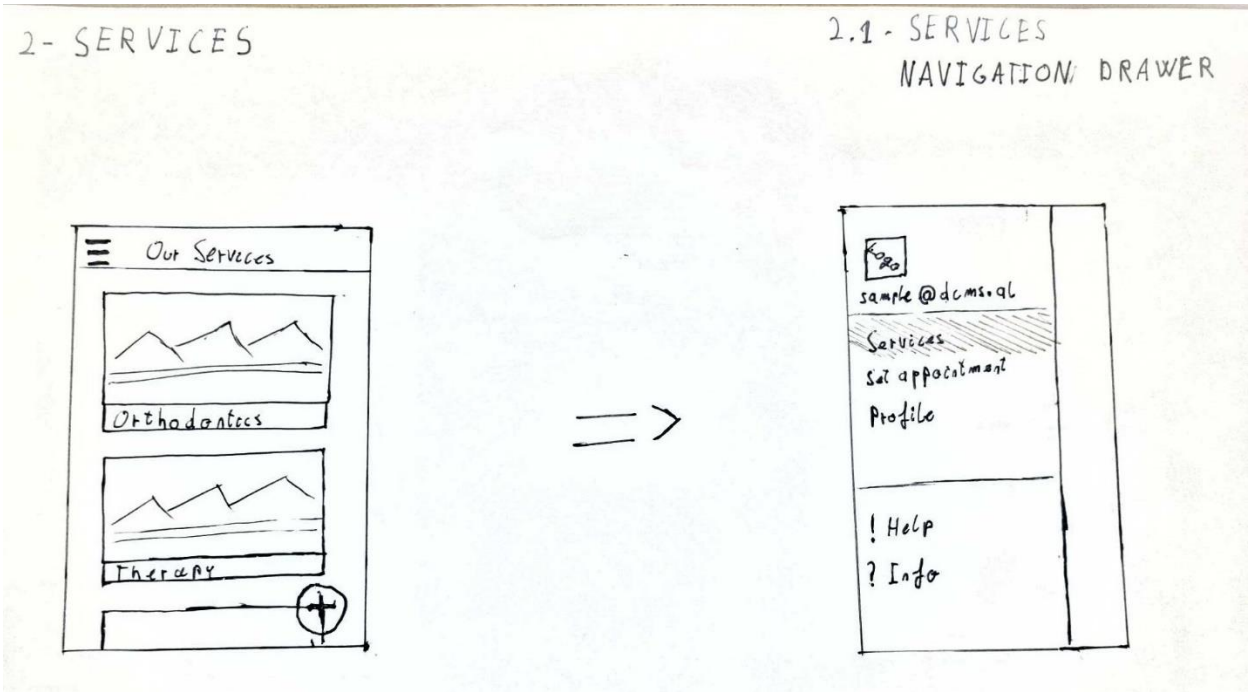
4.7.1. Login form



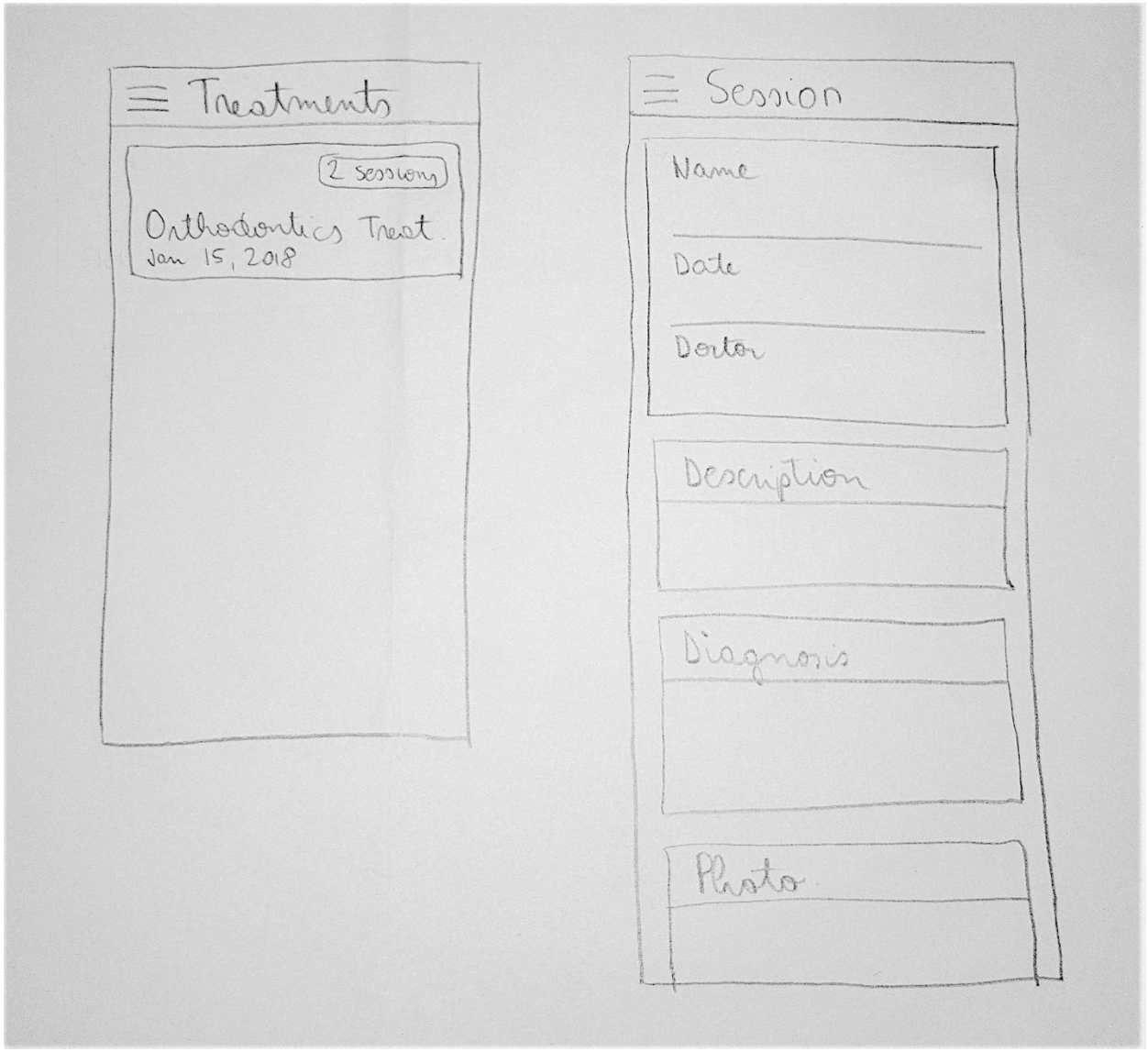
4.7.2. Splash Screen



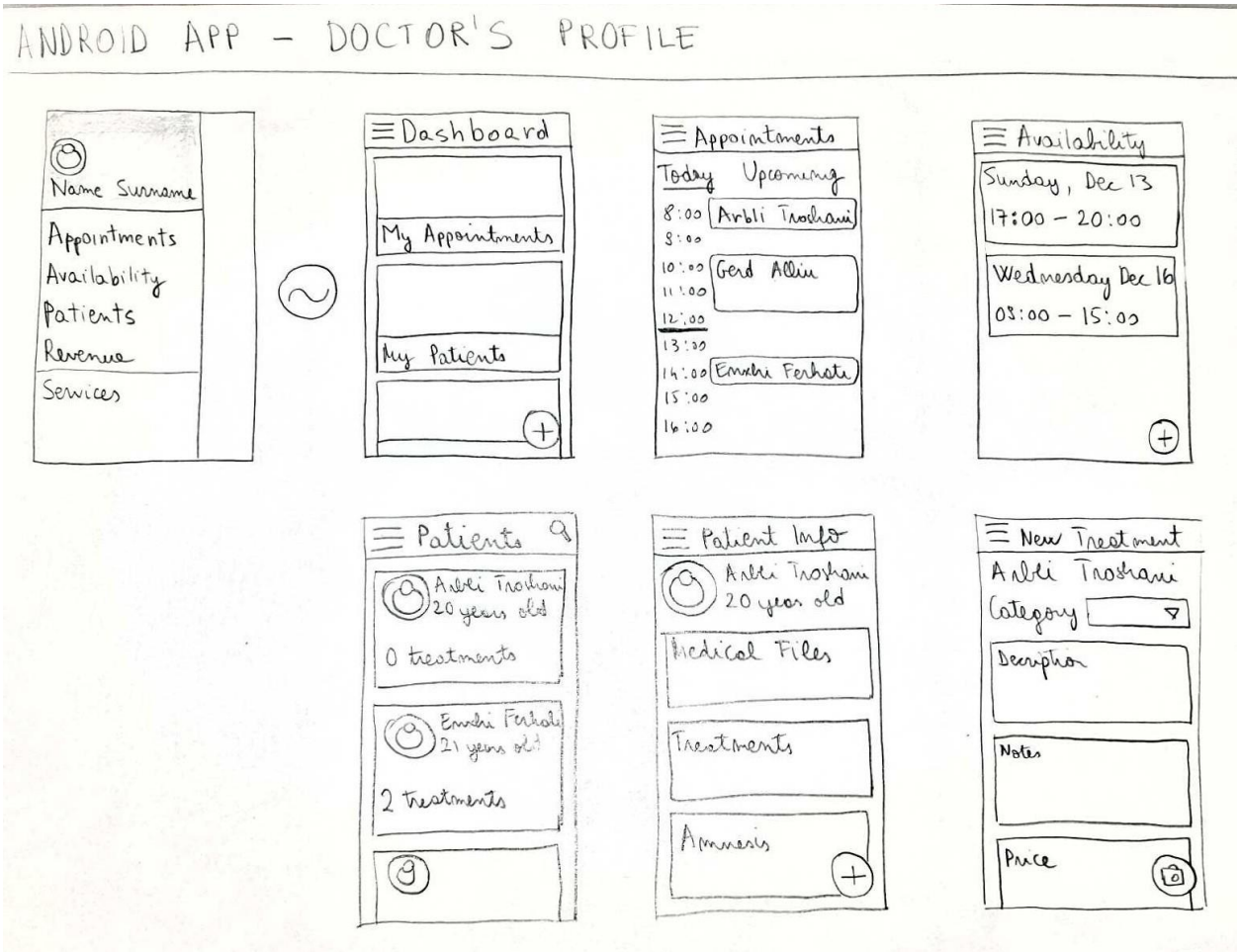
4.7.3. Services and Navigation Drawer



4.7.4. Treatments and sessions



4.7.5. Doctor's View



4.8. Mobile app - Detailed Design

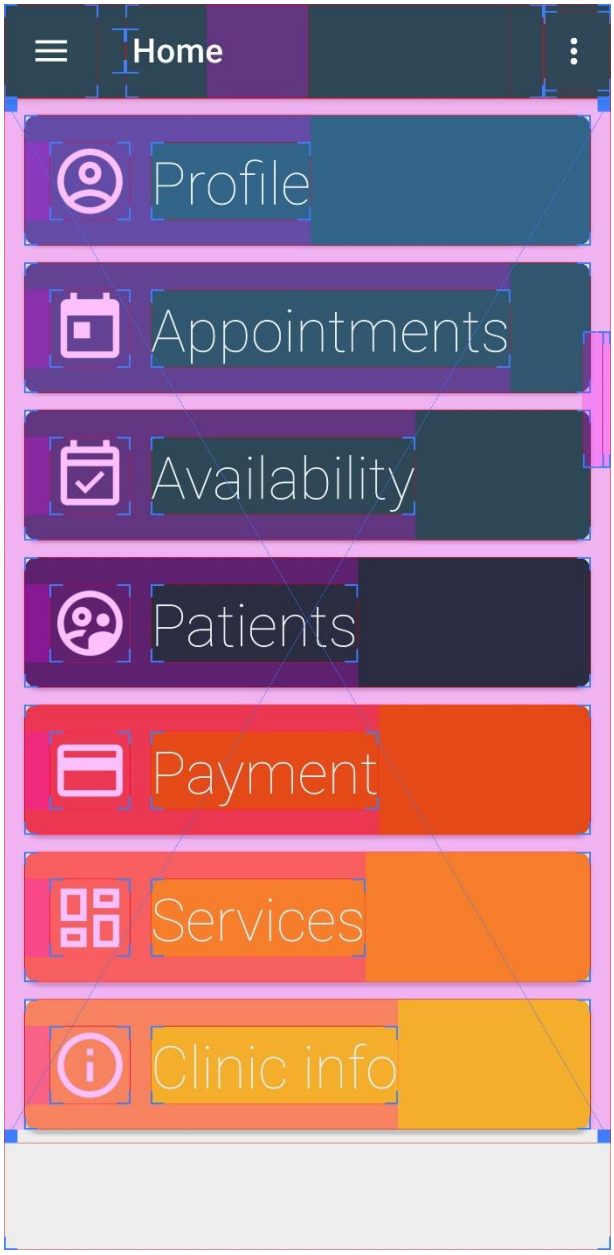
4.8.1. Color Palette

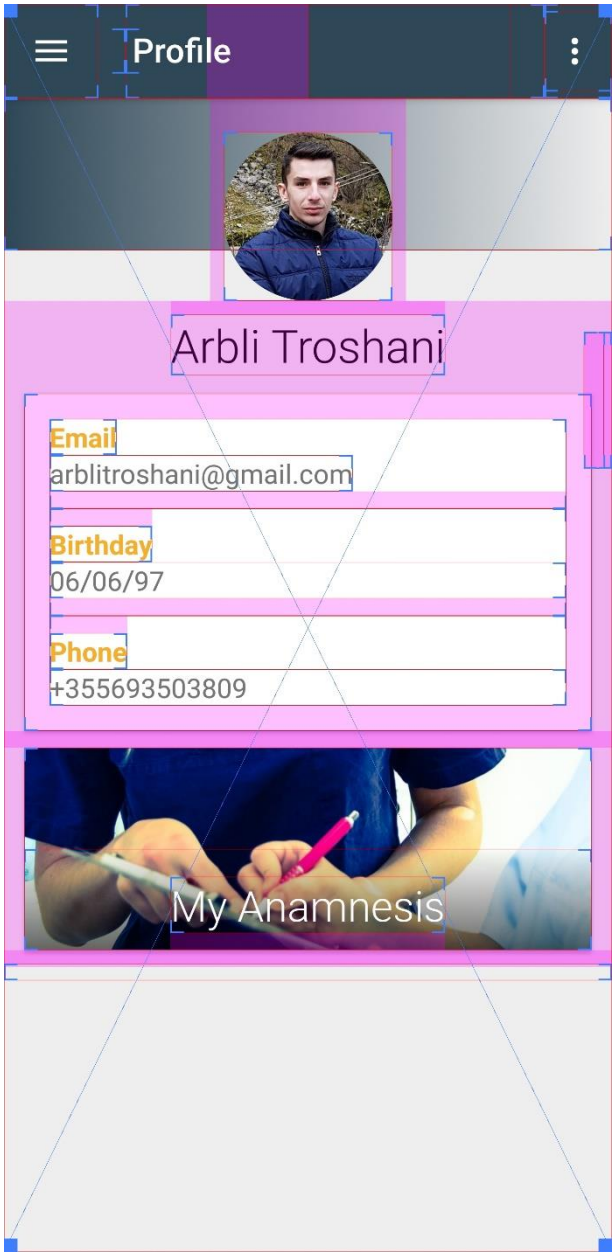
```
    <color name="black_alpha_60">#99000000</color>
    <color name="black_alpha_40">#66000000</color>

    <color name="orange_dark">#E64A19</color>
    <color name="blue_dark">#00537F</color>
    <color name="yellow">#F5A623</color>

    <color name="text_white_primary">@color/white</color>

    <color name="anti_flash_white">#edf2f4</color>
    <color name="gunmetal">#2b2d42</color>
    <color name="gray_blue">#8d99ae</color>
    <color name="red_pantone">#ef233c</color>

    <color name="turqoise">#55dde0</color>
    <color name="teal_blue">#33658a</color>
    <color name="midcolor">#315771</color>
    <color name="dark_slate_gray">#2f4858</color>
    <color name="saffron">#f6ae2d</color>
    <color name="giants_orange">#f26419</color>
    <color name="giants_orange_light">#f67e2d</color>

    <color name="appointmentCompleted">#02C39A</color>
    <color name="appointmentConfirmed">#028090</color>
```

4.8.2. Dimensions

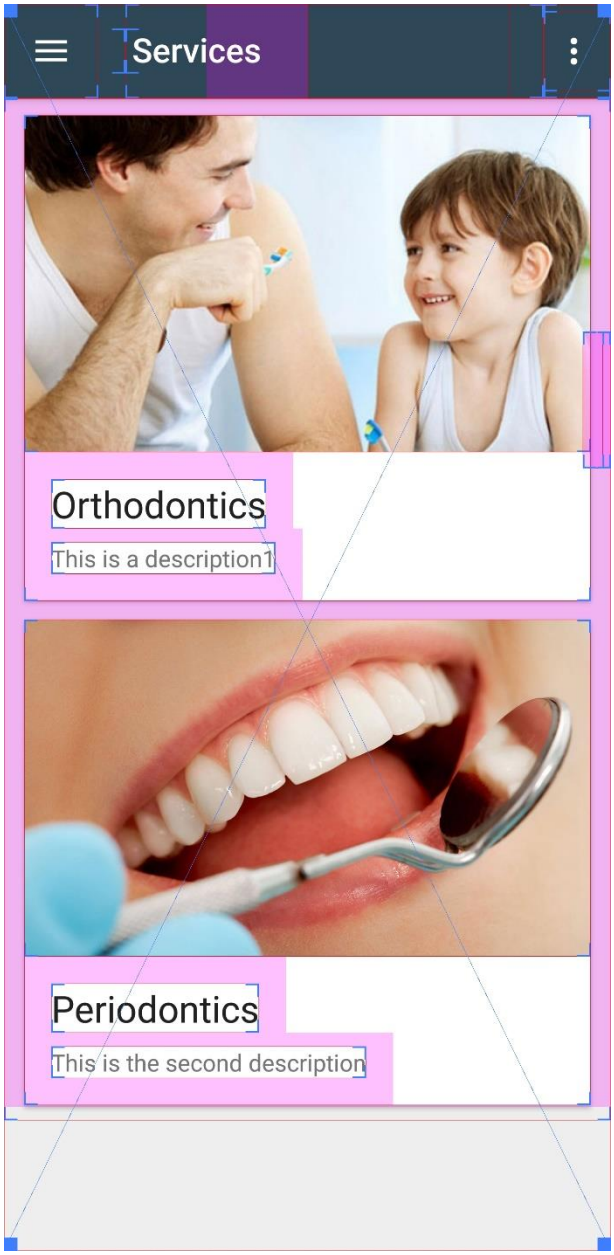
```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="nav_header_vertical_spacing">8dp</dimen>
    <dimen name="nav_header_height">176dp</dimen>
    <dimen name="fab_margin">16dp</dimen>

    <dimen name="main_card_margin_horizontal">12dp</dimen>
    <dimen name="main_card_margin_vertical">10dp</dimen>

    <dimen name="card_recycler_corner_radius">2dp</dimen>
    <dimen name="card_recycler_elevation">2dp</dimen>

    <dimen name="card_title_margin_top">16dp</dimen>
    <dimen name="card_title_margin">16dp</dimen>
    <dimen name="card_subtitle_margin">16dp</dimen>
    <dimen name="card_button_margin">8dp</dimen>

    <!-- Navigation Drawer -->
    <dimen name="navigation_drawer_max_width">320dp</dimen>

    <dimen name="spacing_8">8dp</dimen>
    <dimen name="spacing_56">56dp</dimen>
    <dimen name="vertical_keyline_first">16dp</dimen>
    <dimen name="nav_drawer_profile_image_size">64dp</dimen>

    <dimen name="elevation_app_bar">4dp</dimen>
    <dimen name="elevation_nav_drawer">16dp</dimen>
    <dimen name="appbar_padding_top">8dp</dimen>
    <dimen name="app_bar_height">192dp</dimen>
    <dimen name="text_margin">16dp</dimen>

    <dimen name="card_main_title">35sp</dimen>
    <dimen name="card_main_title_margin_start">75dp</dimen>
    <dimen name="card_main_title_margin_bottom">15dp</dimen>
    <dimen name="card_main_icon_margin_start">15dp</dimen>
    <dimen name="card_main_subtitle_margin_start">22dp</dimen>

</resources>
```

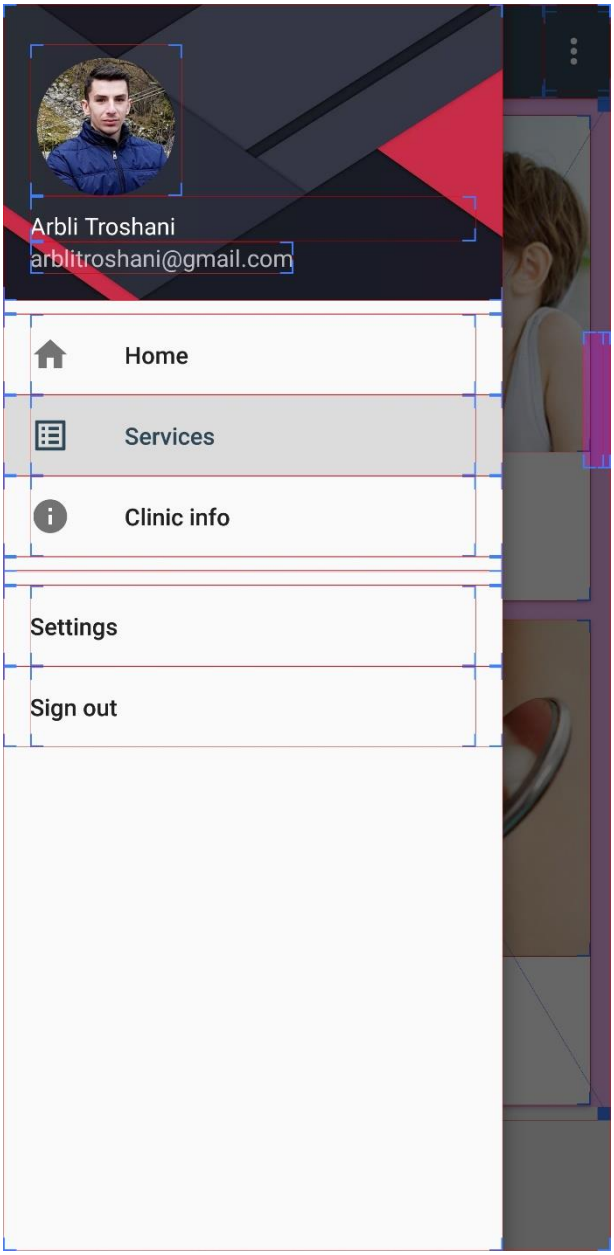
4.8.3. Home (Doctor). User Profile



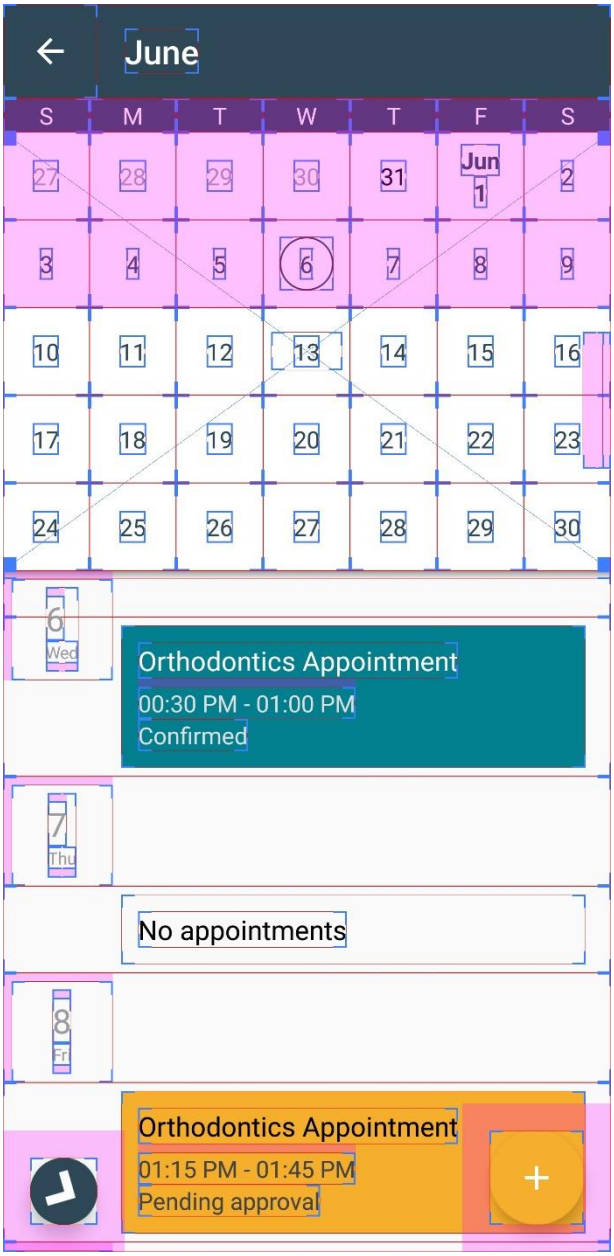


4.8.4. Services. Navigation Drawer



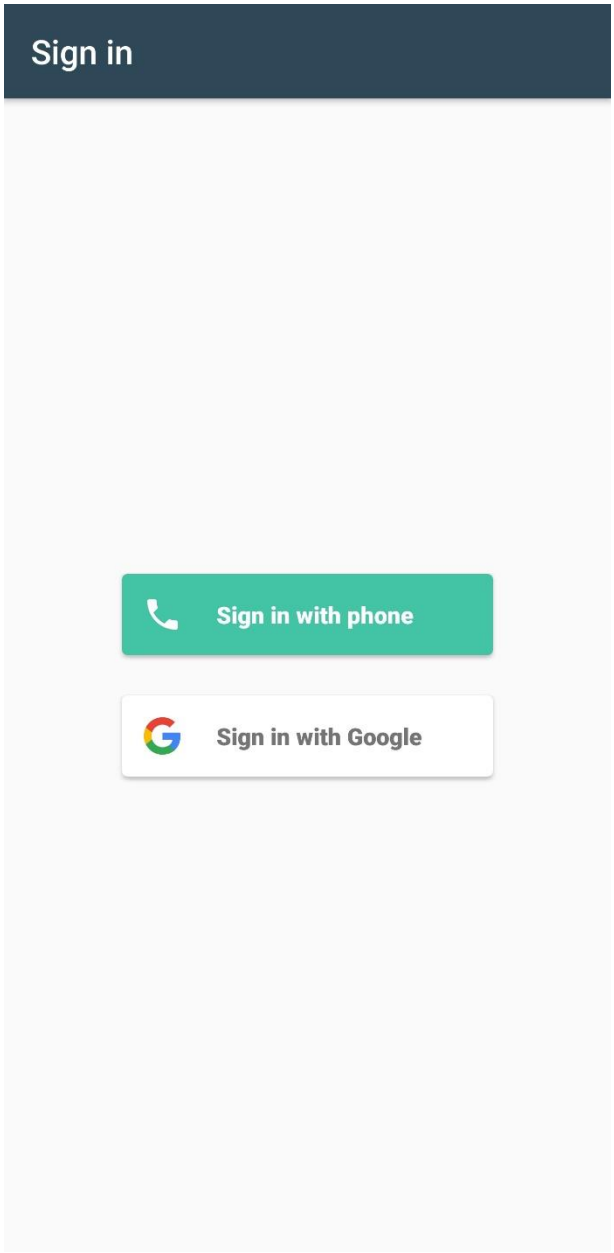


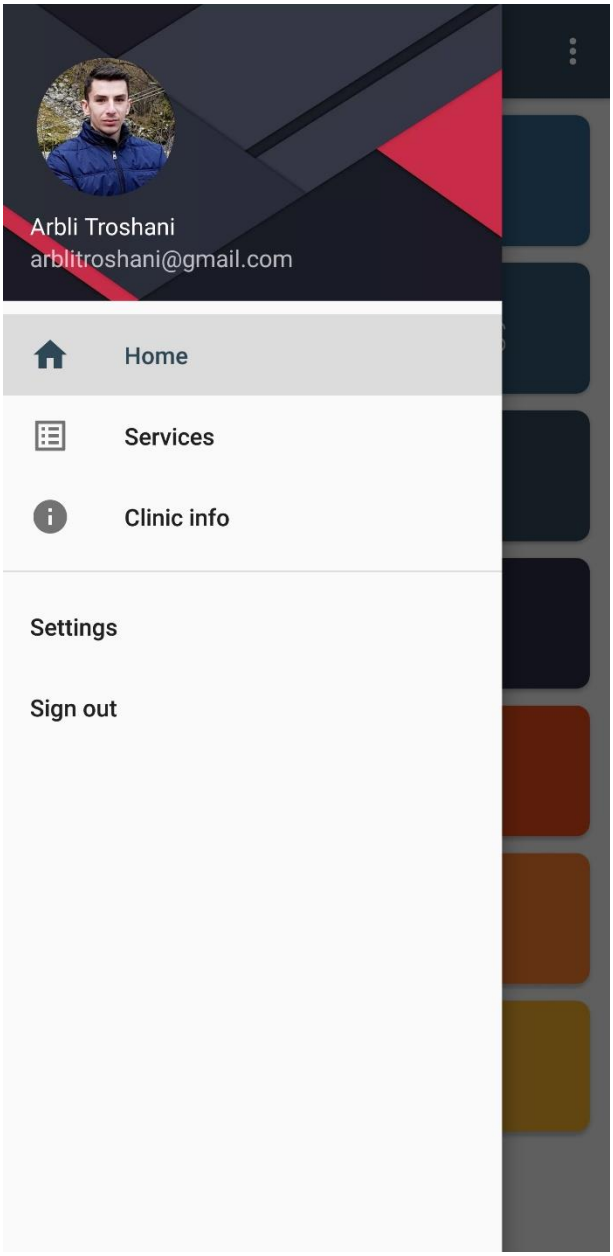
4.8.5. Appointments



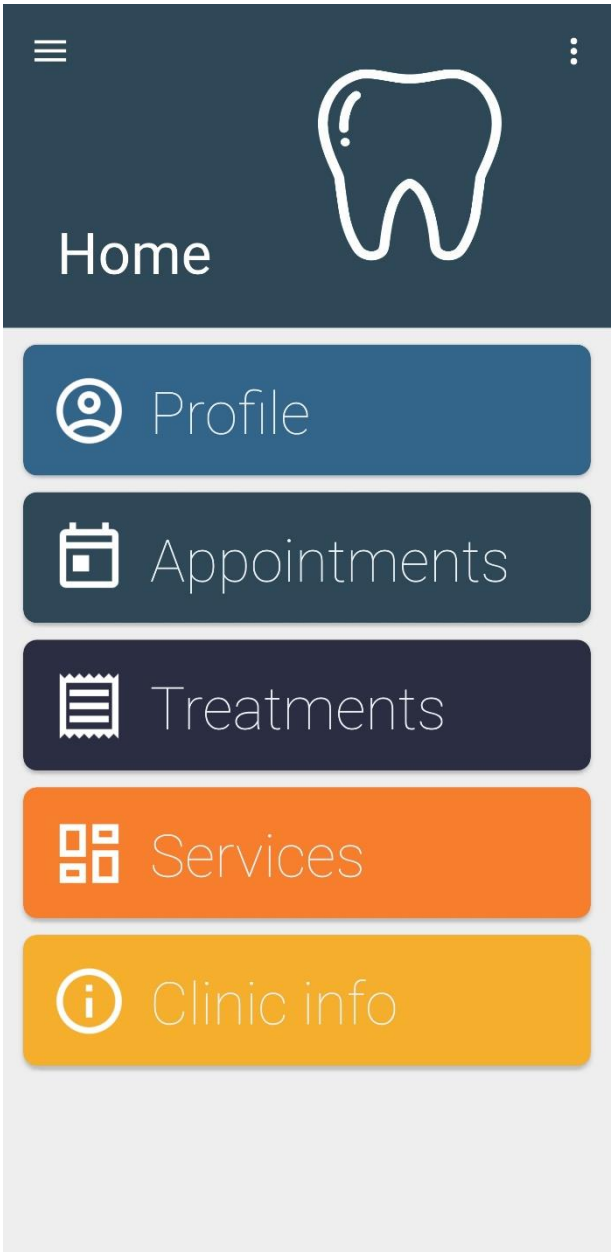
4.9. Mobile app – Screenshots

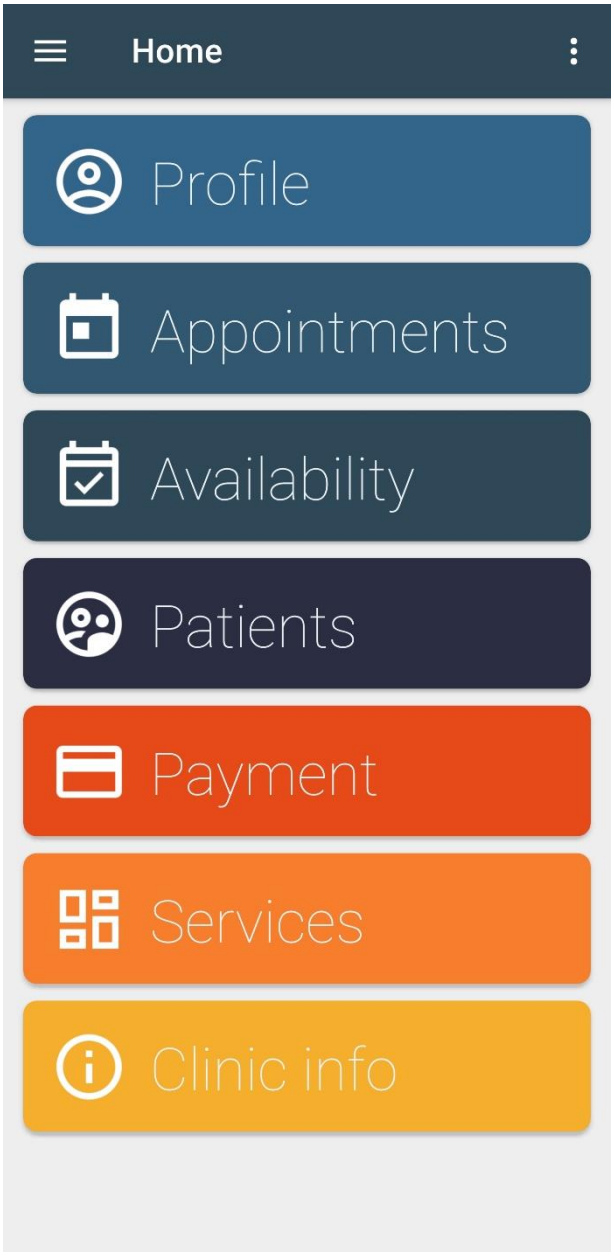
4.9.1. Sign In. Navigation Drawer.



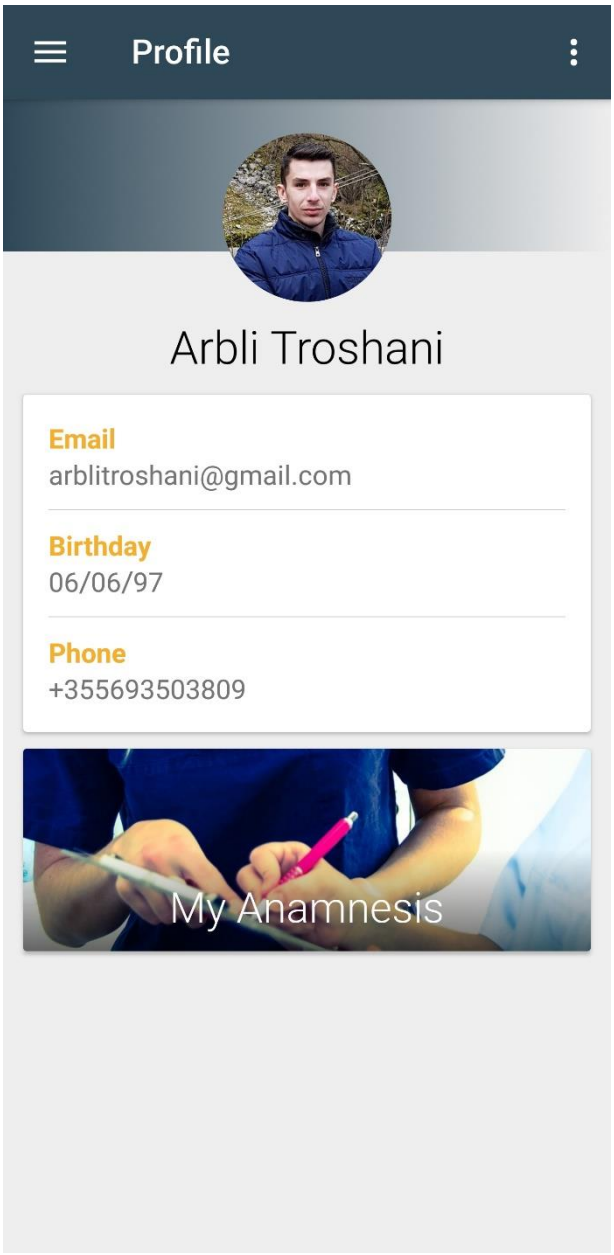


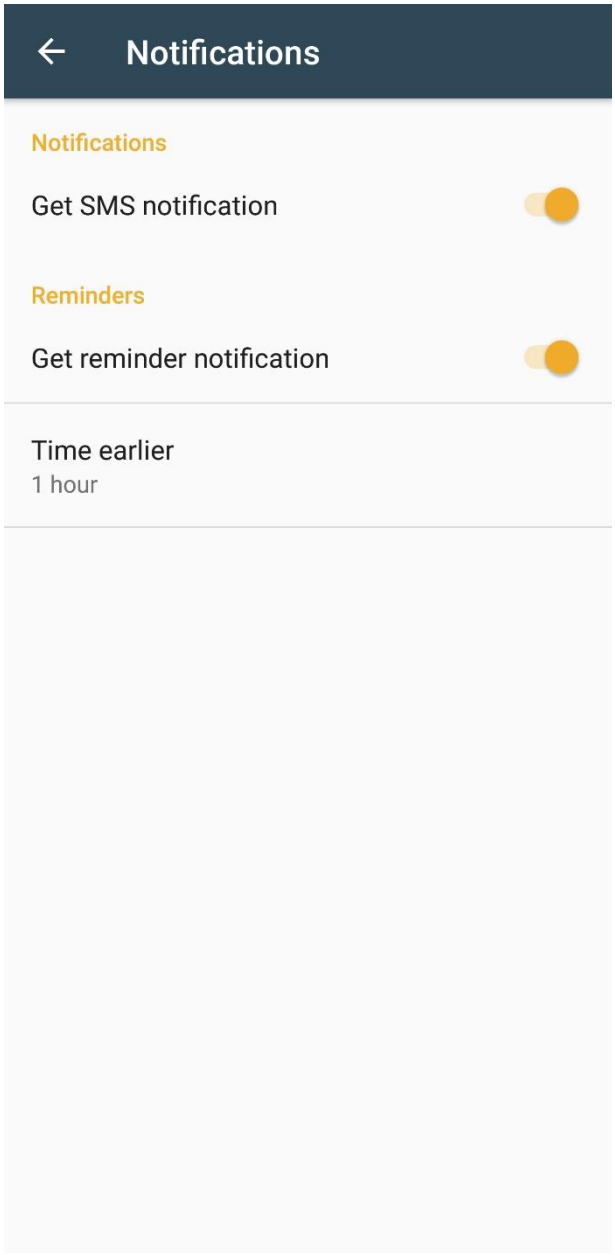
4.9.2. Home (User and Doctor)



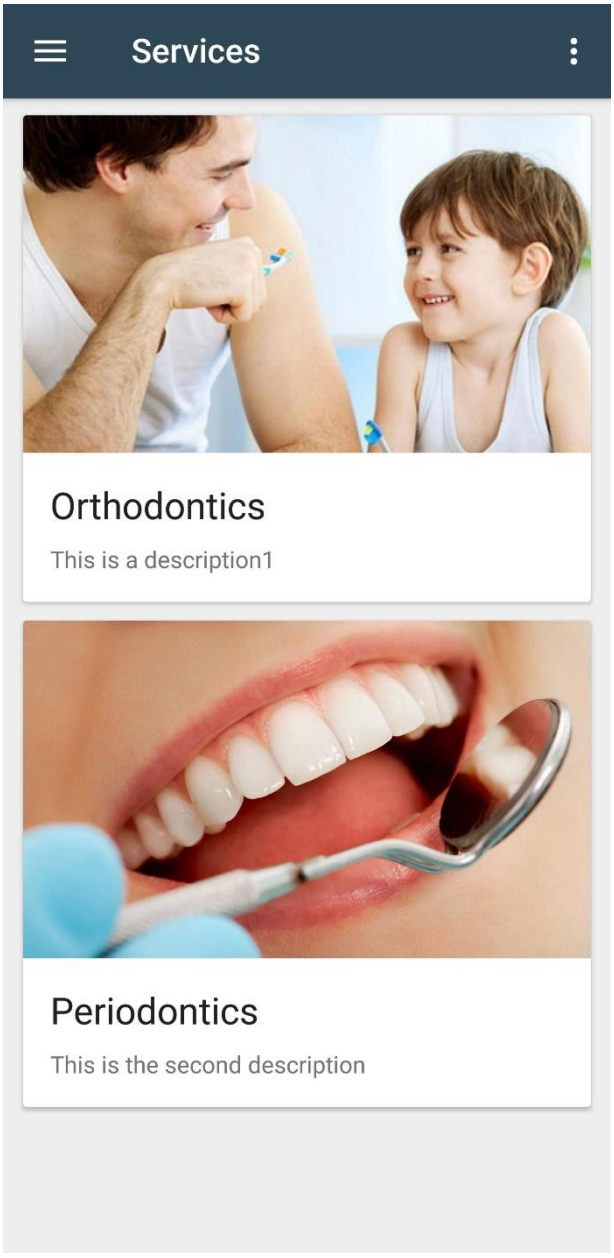


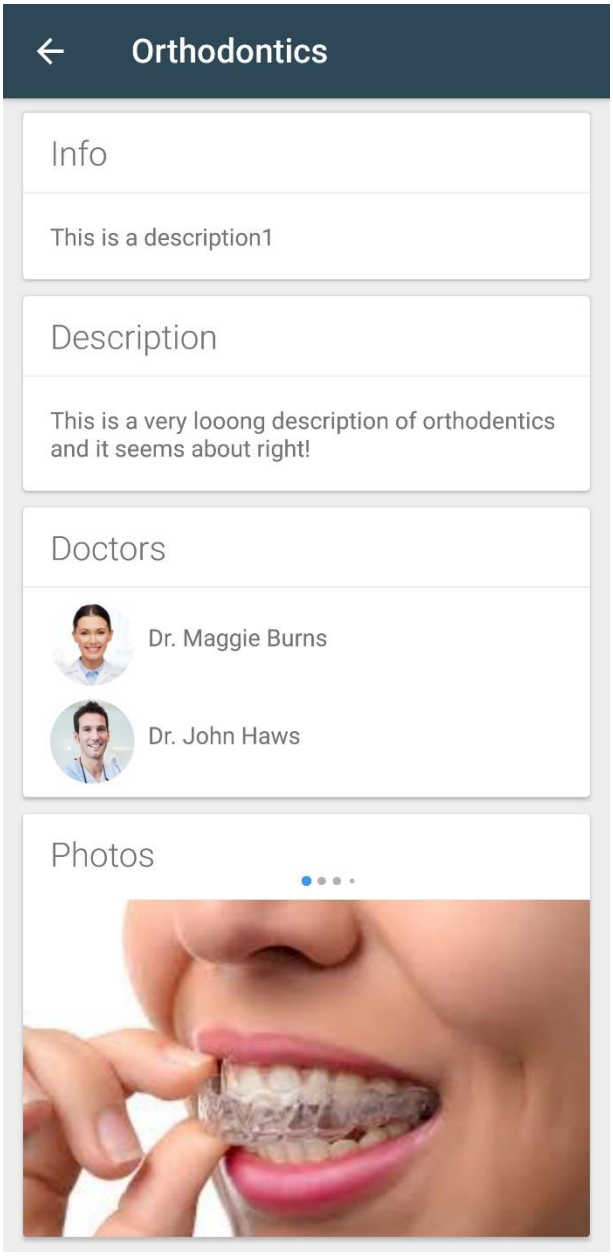
4.9.3. User Profile. Settings



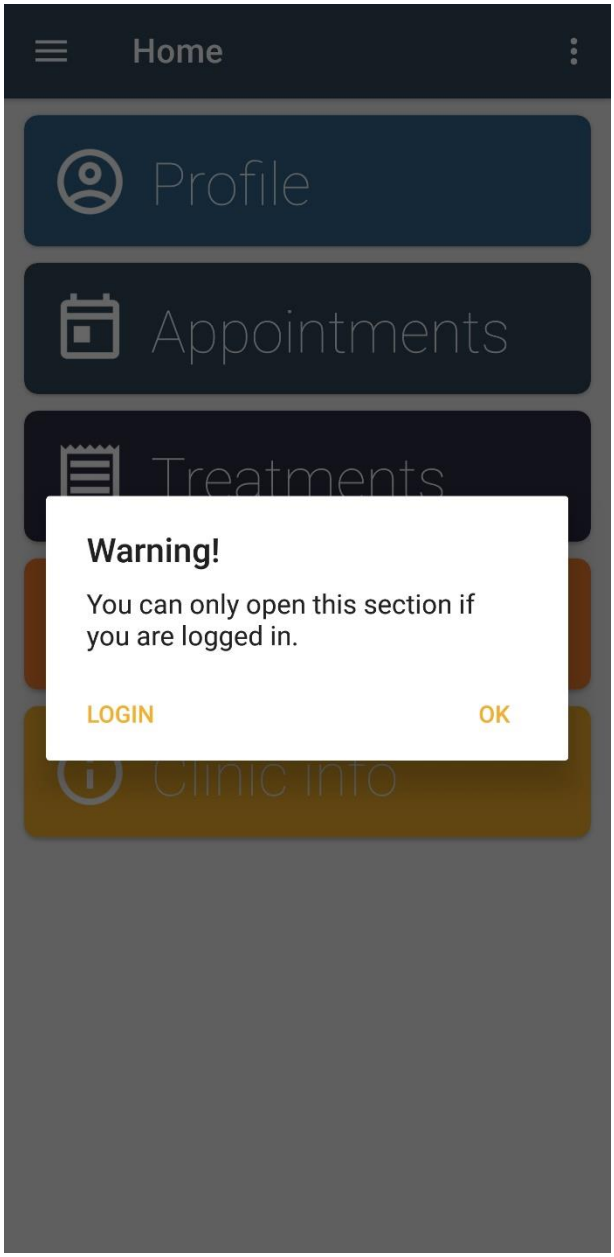


4.9.4. Services. Service Details

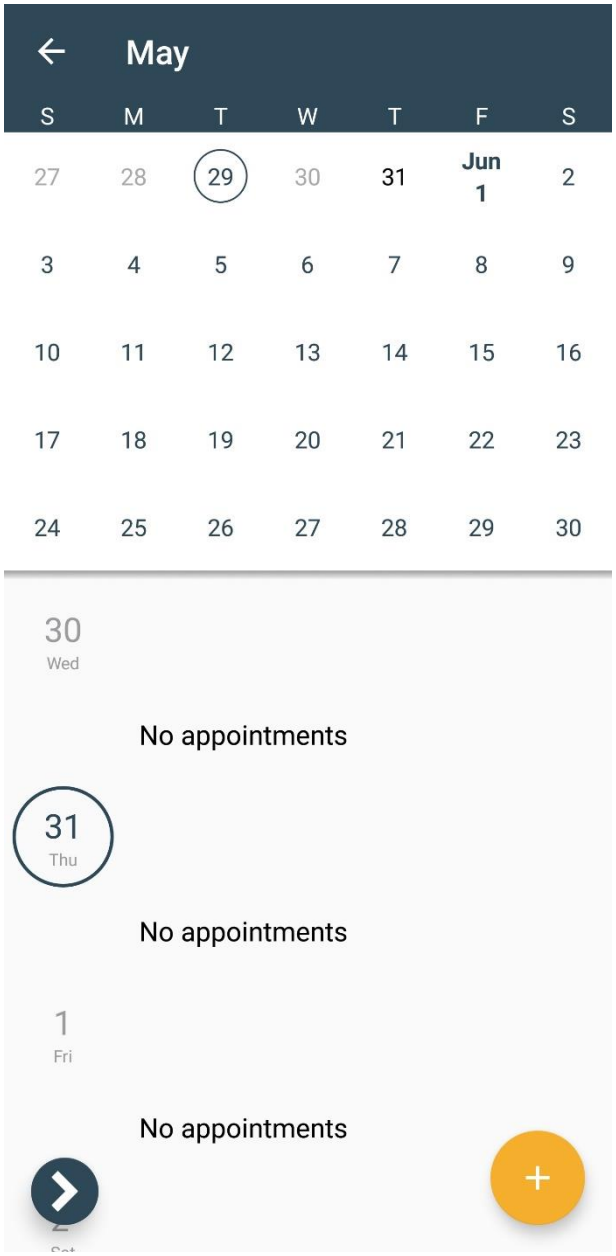


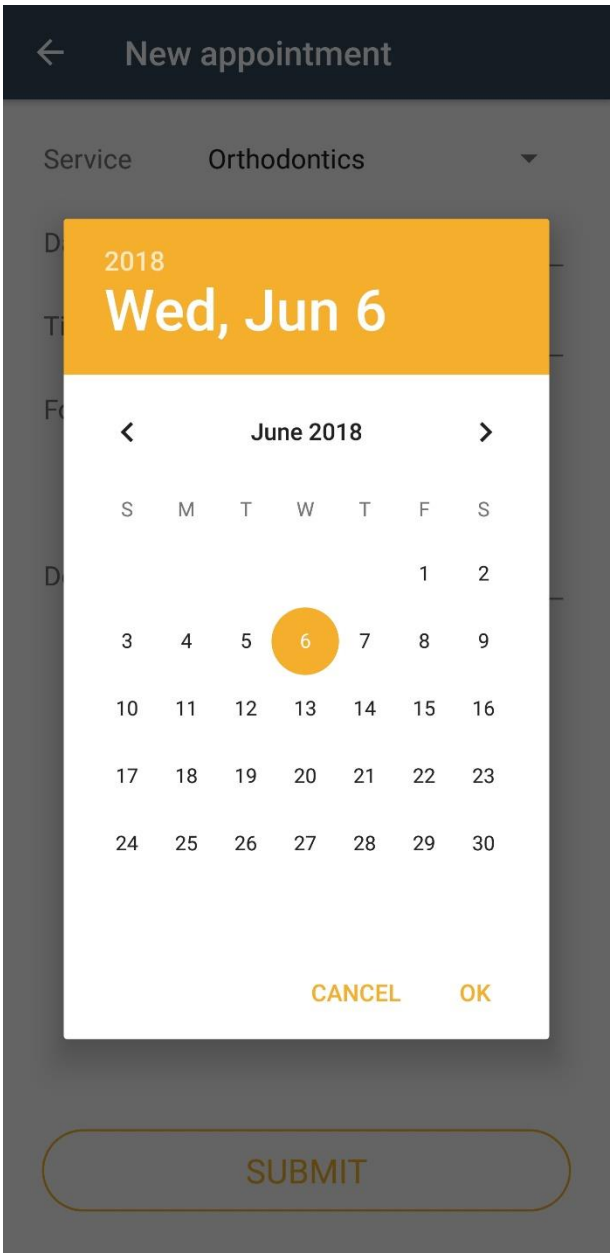


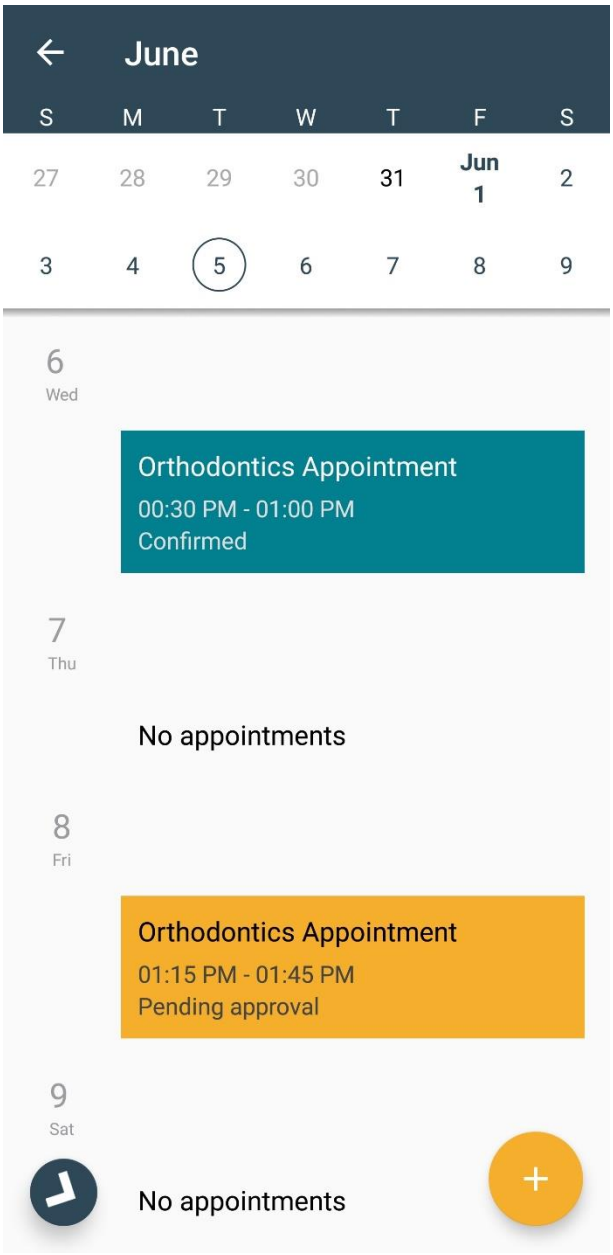
4.9.5. Login Warning. ID scan.

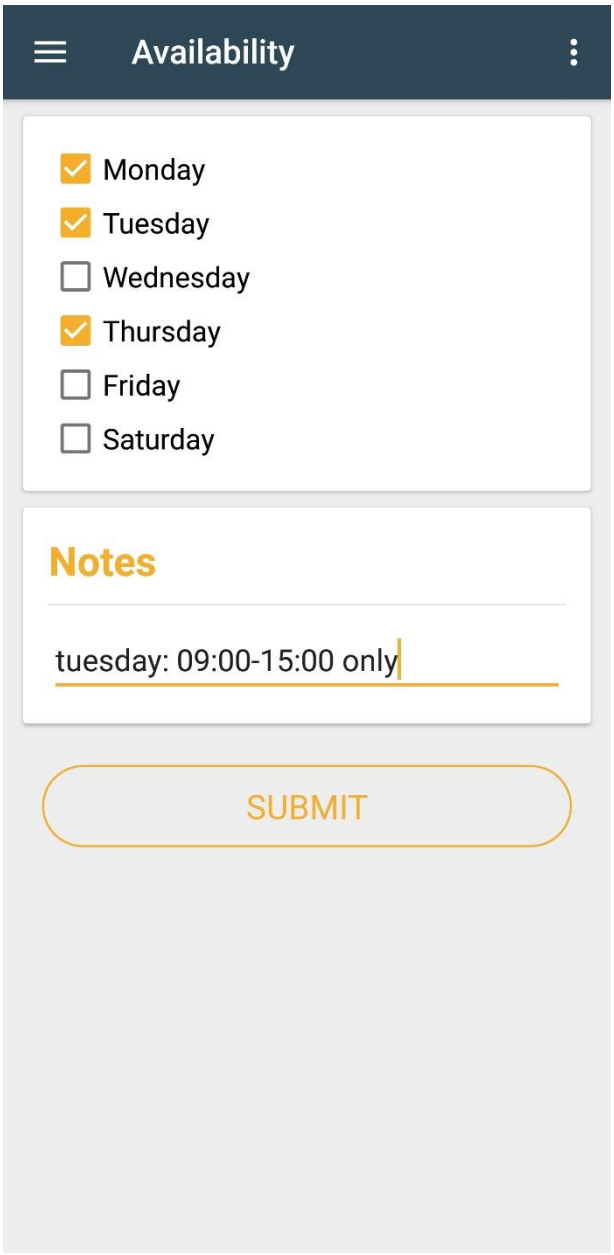


4.9.6. Appointments. Availability.

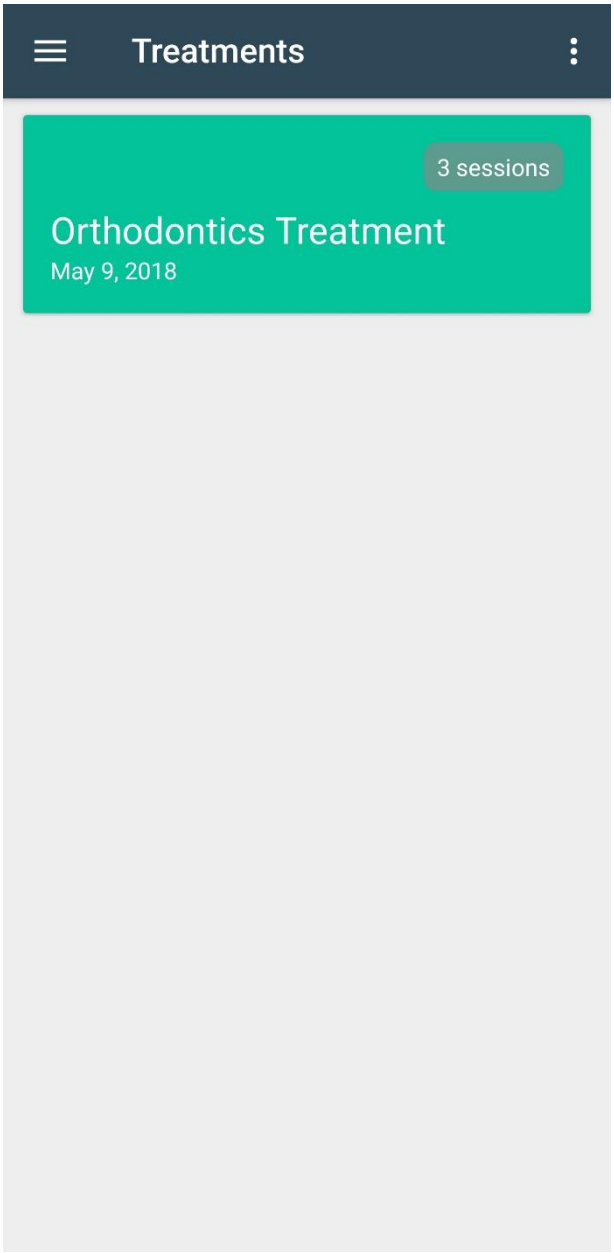





June
S M T W T F S
27 28 29 30 31 Jun 1 2
3 4 5 6 7 8 9
6 Wed
Orthodontics Appointment
00:30 PM - 01:00 PM
Confirmed
7 Thu
No appointments
8 Fri
Orthodontics Appointment
01:15 PM - 01:45 PM
Pending approval
9 Sat
No appointments

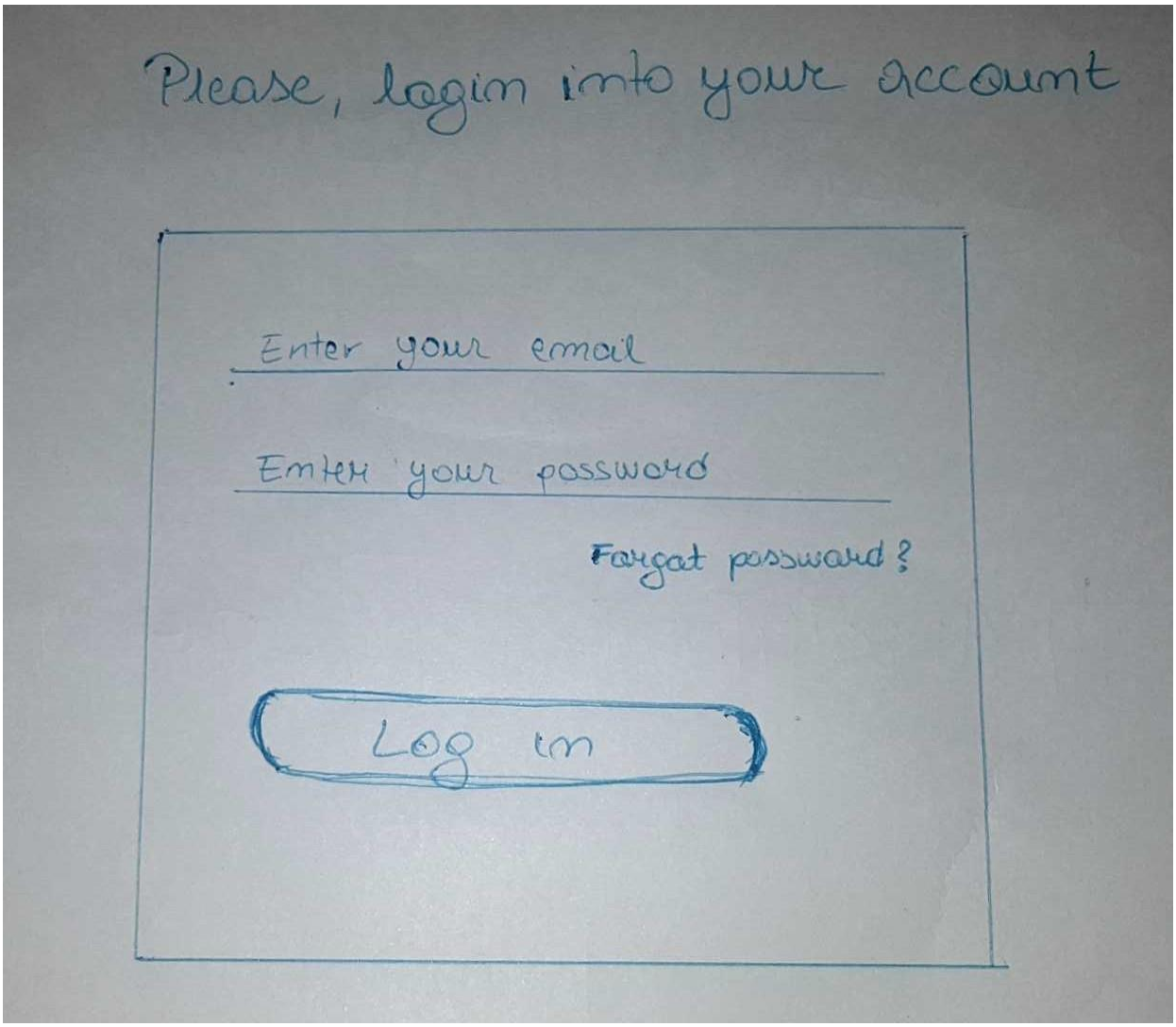

Availability
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Notes
tuesday: 09:00-15:00 only
SUBMIT

4.9.7. Treatments.

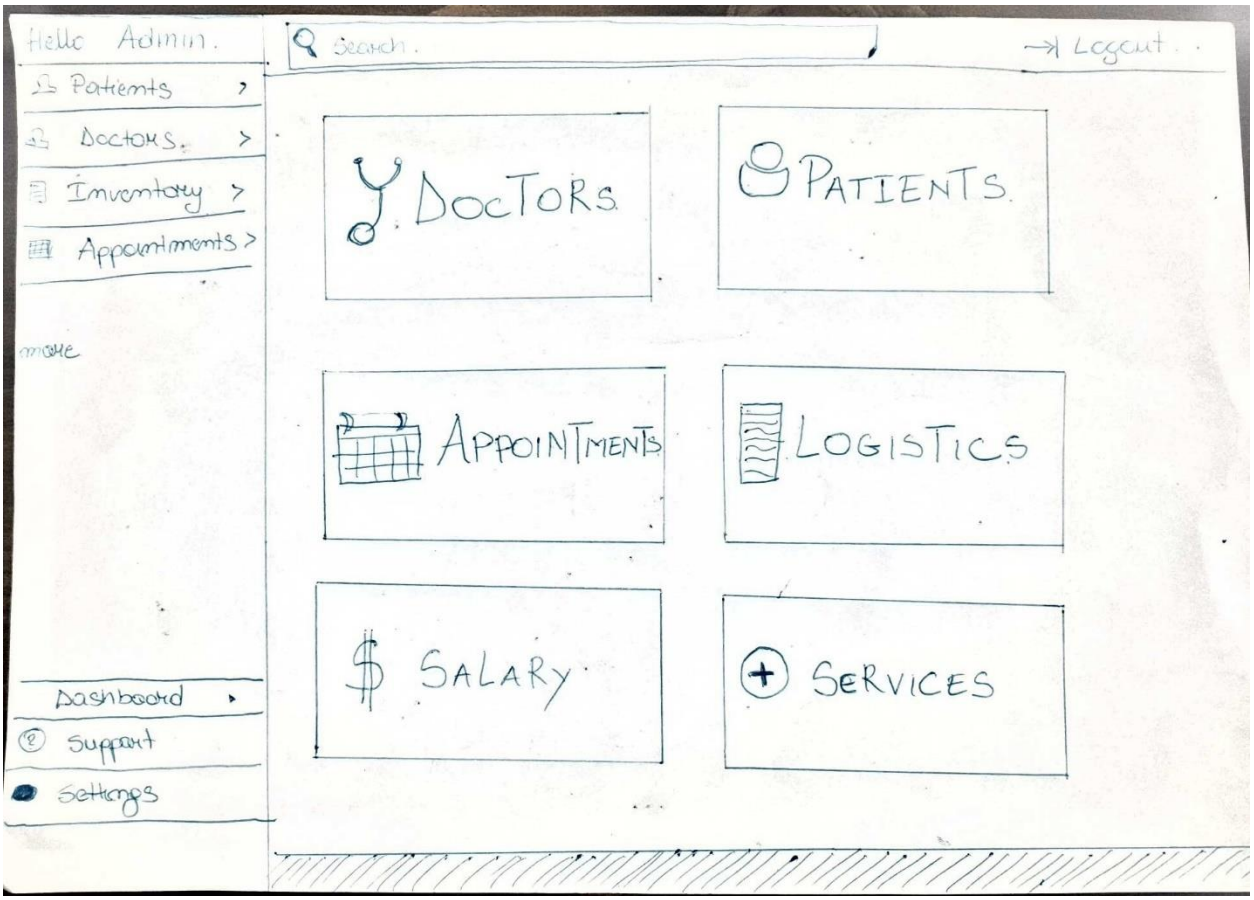


4.10. Web app – Sketches

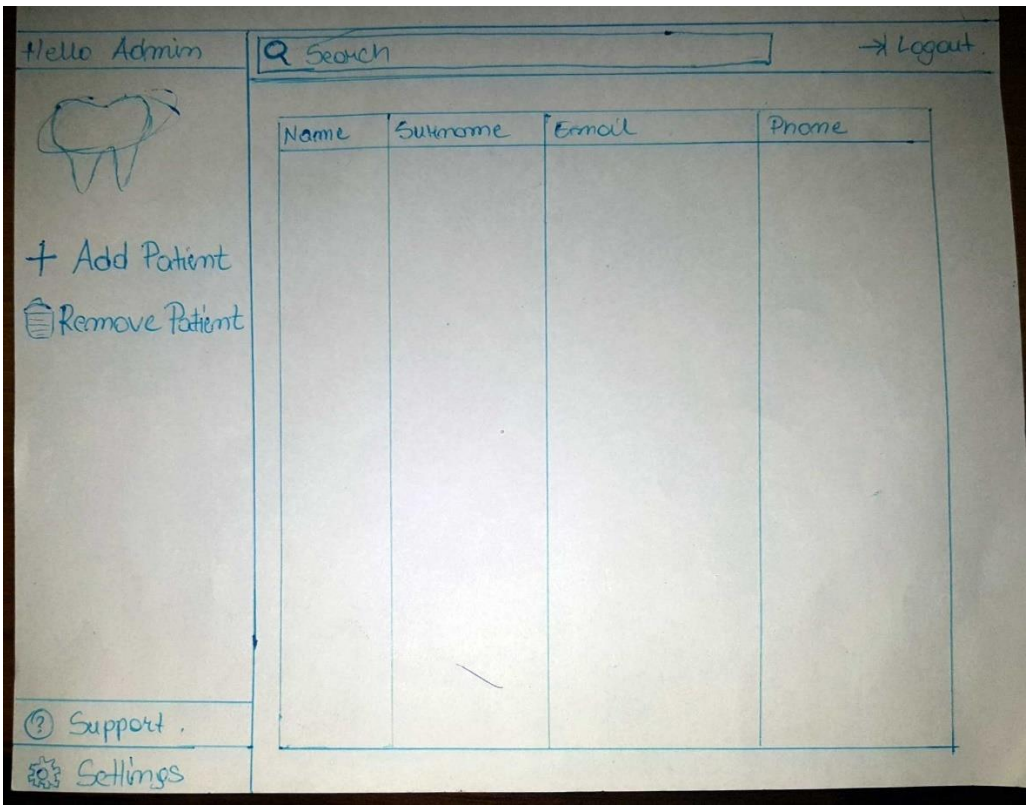
4.10.1. Admin Login



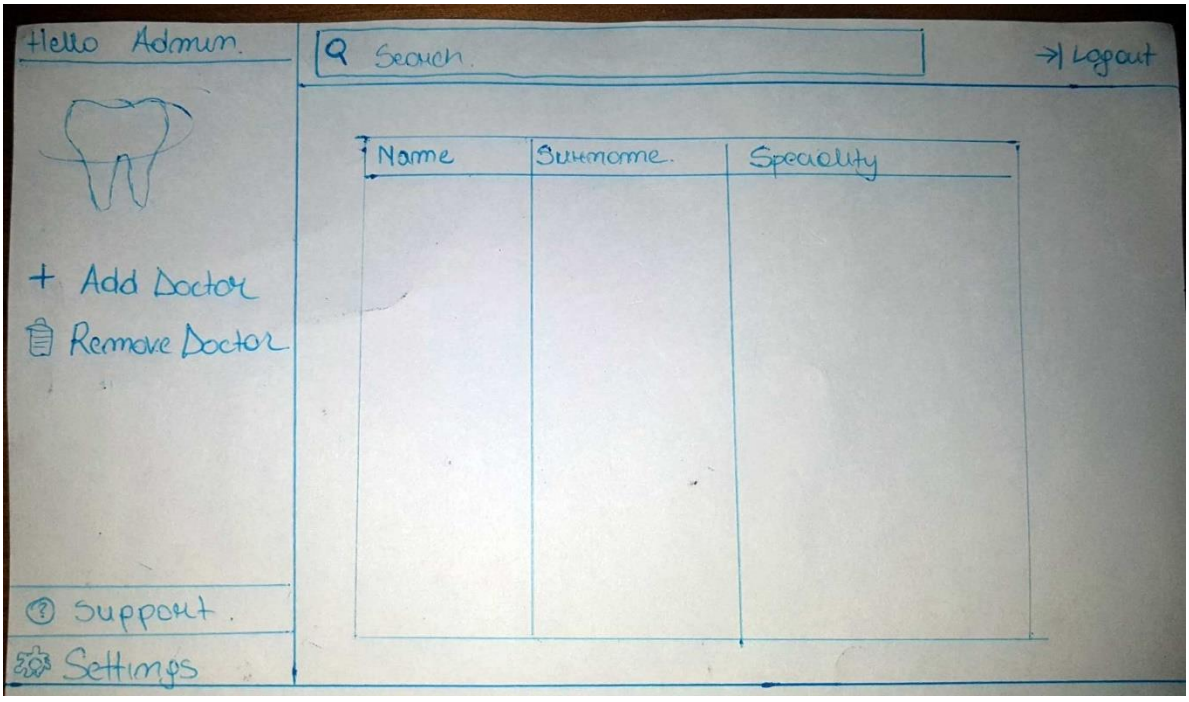
4.10.2. Admin Panel



4.10.3. Manage Patients

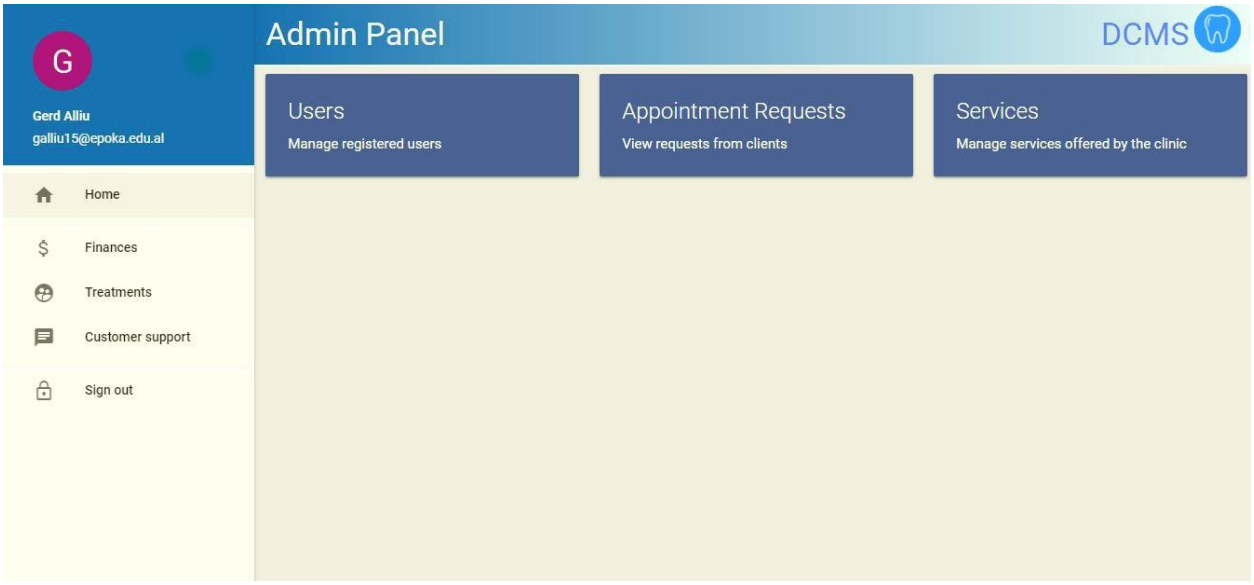


4.10.4. Manage Doctors

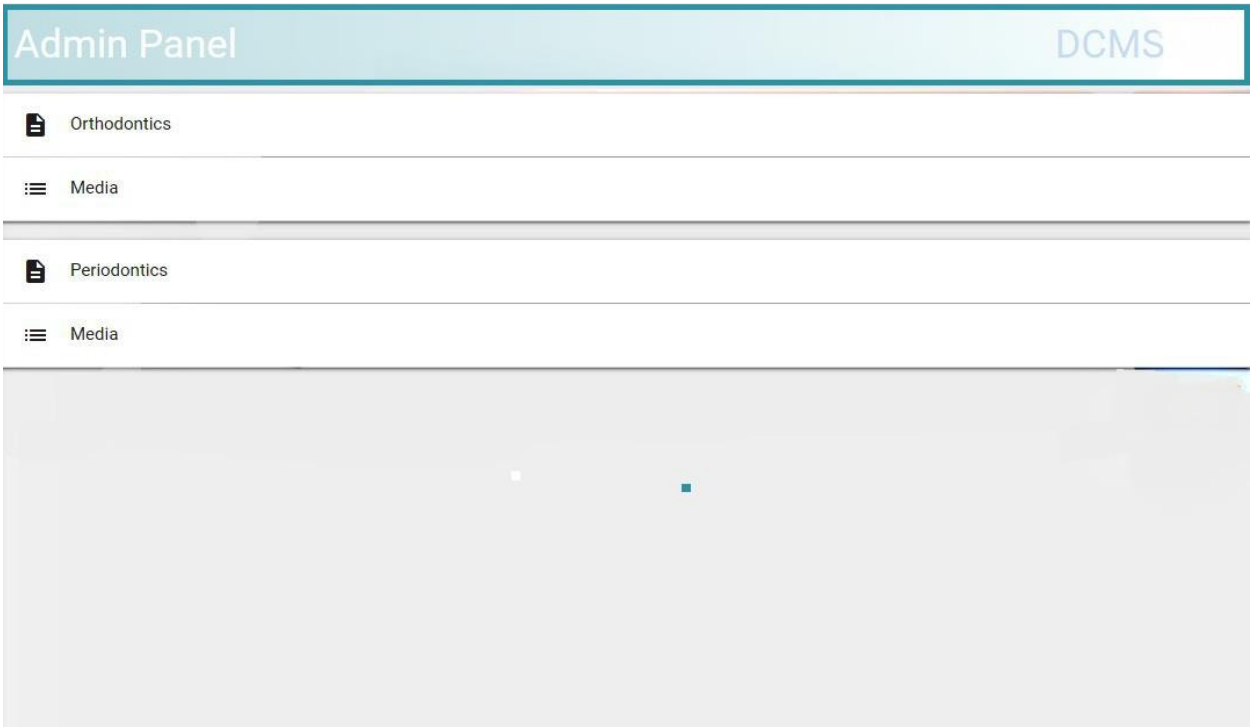


4.11. Web app – Detailed Design

4.11.1. Home

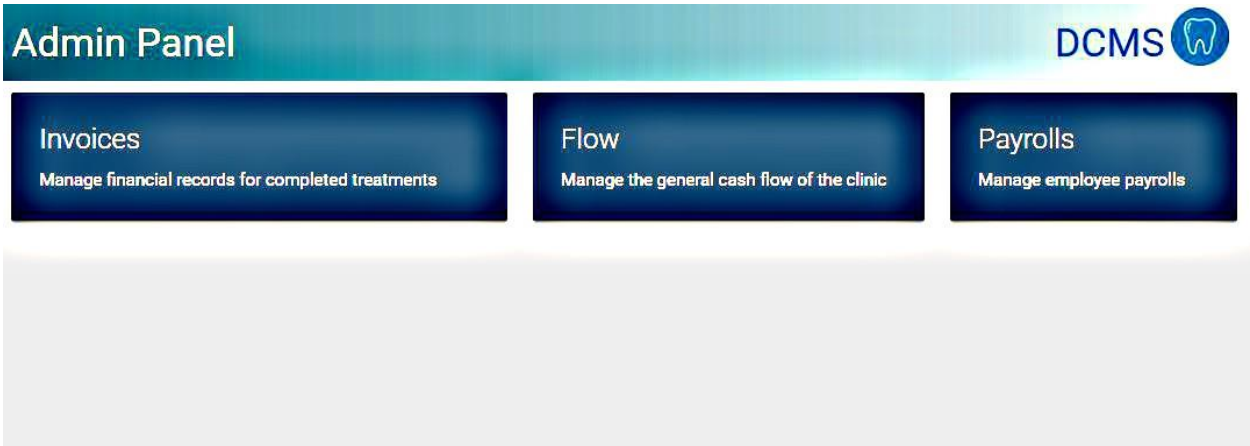


4.11.2. Services

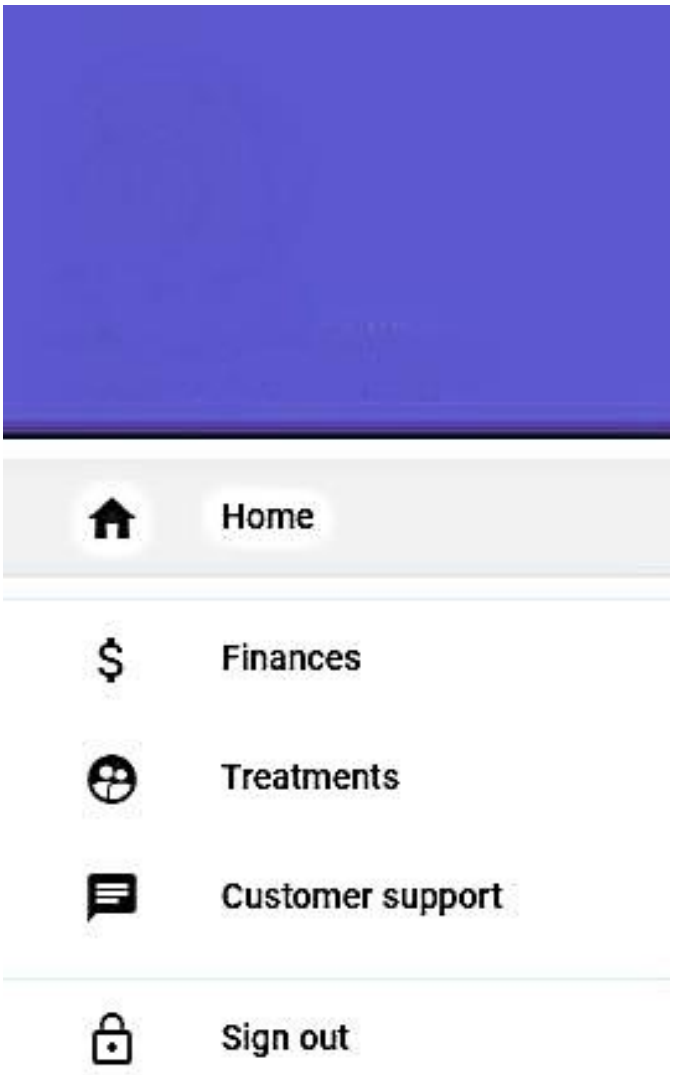


4.11.3. Users

4.11.4. Finance

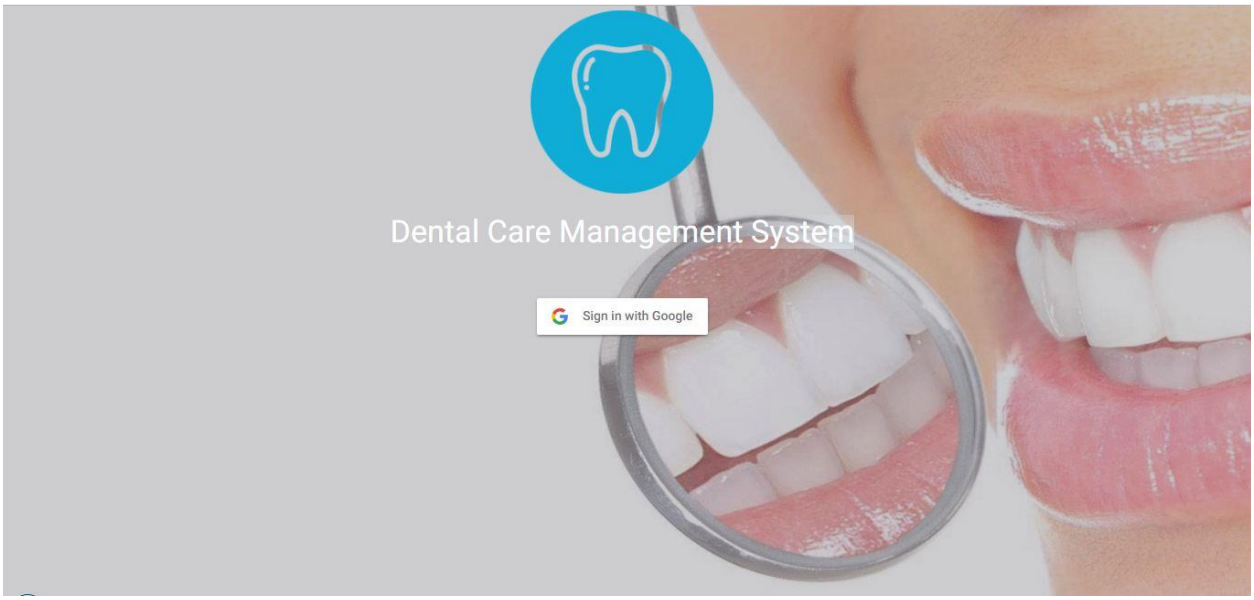


4.11.5. Navigation

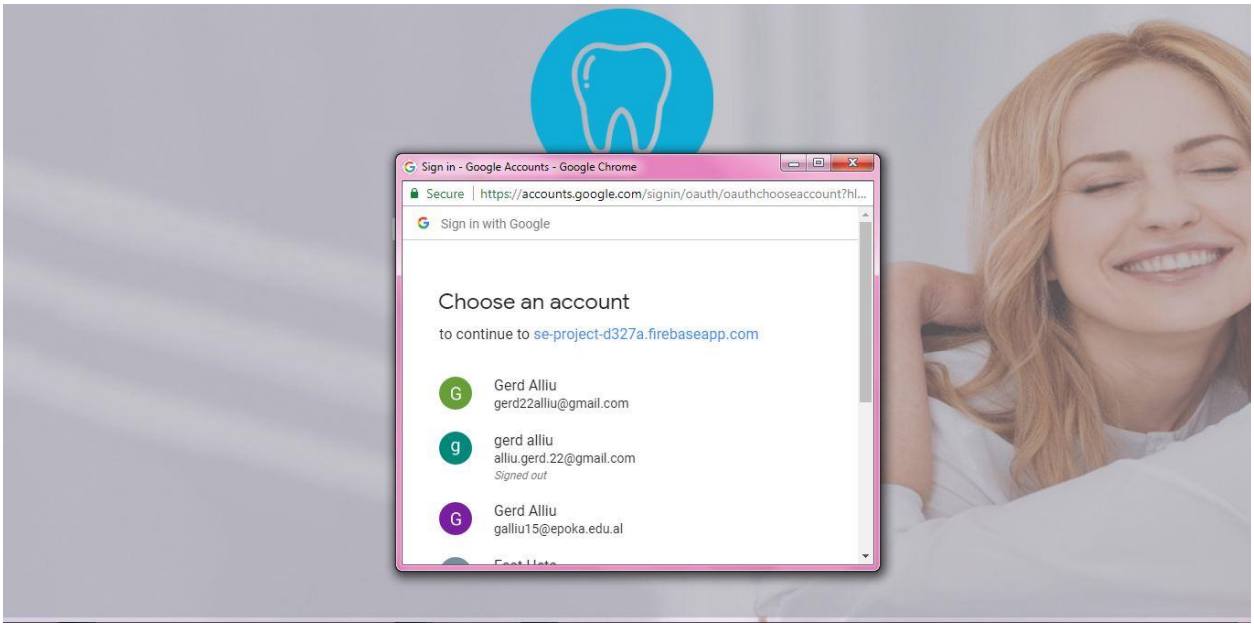


4.12. Web app - Screenshots

4.12.1. Login

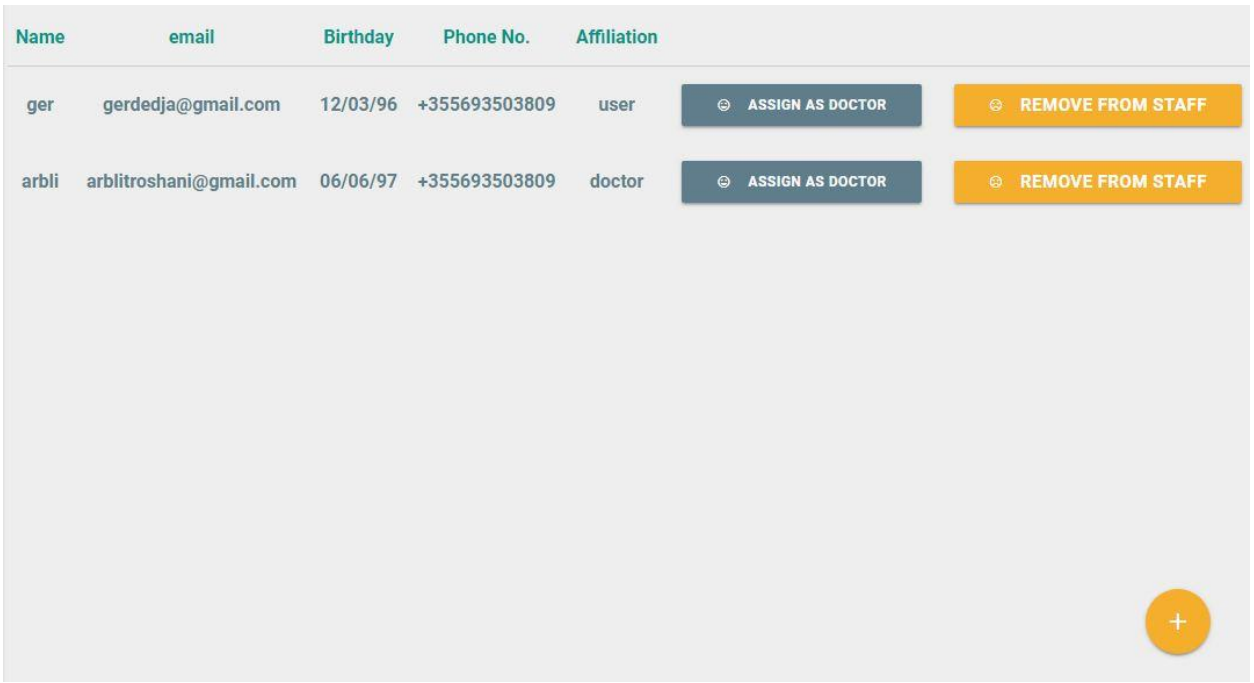


4.12.2. Popup window

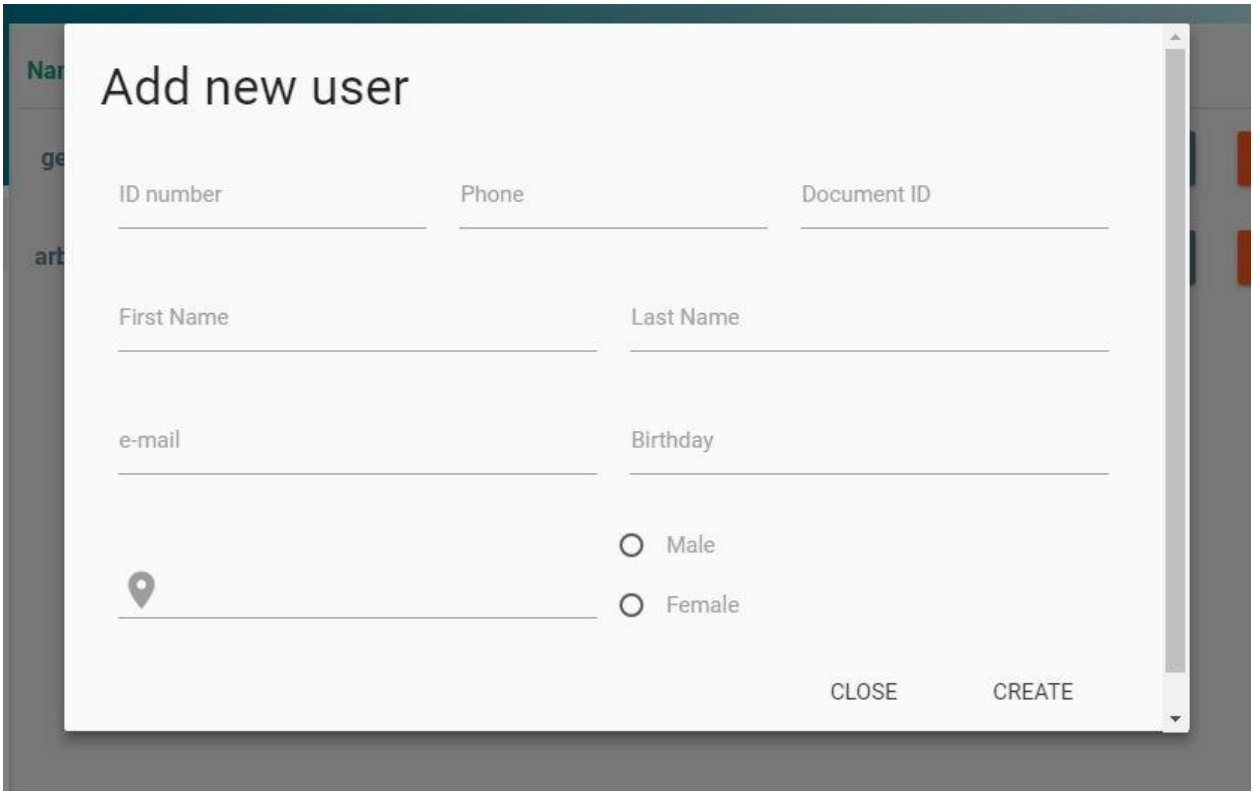


4.12.3. Home

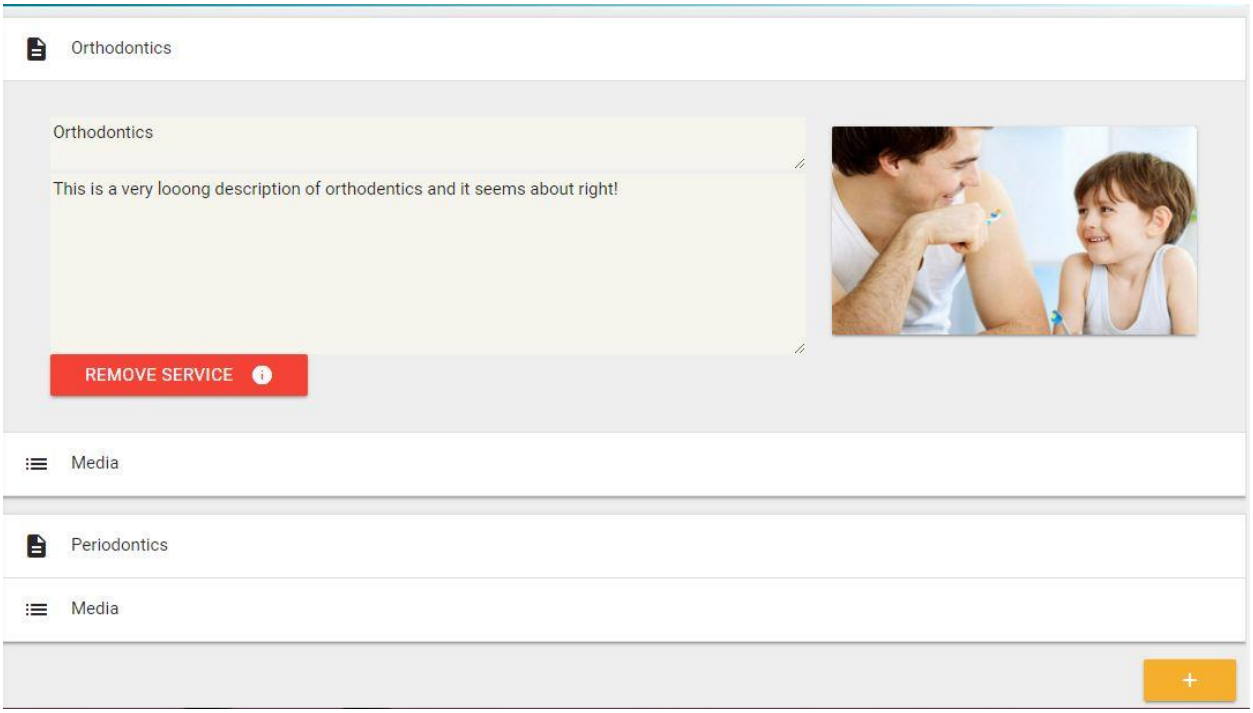
4.12.4. Users



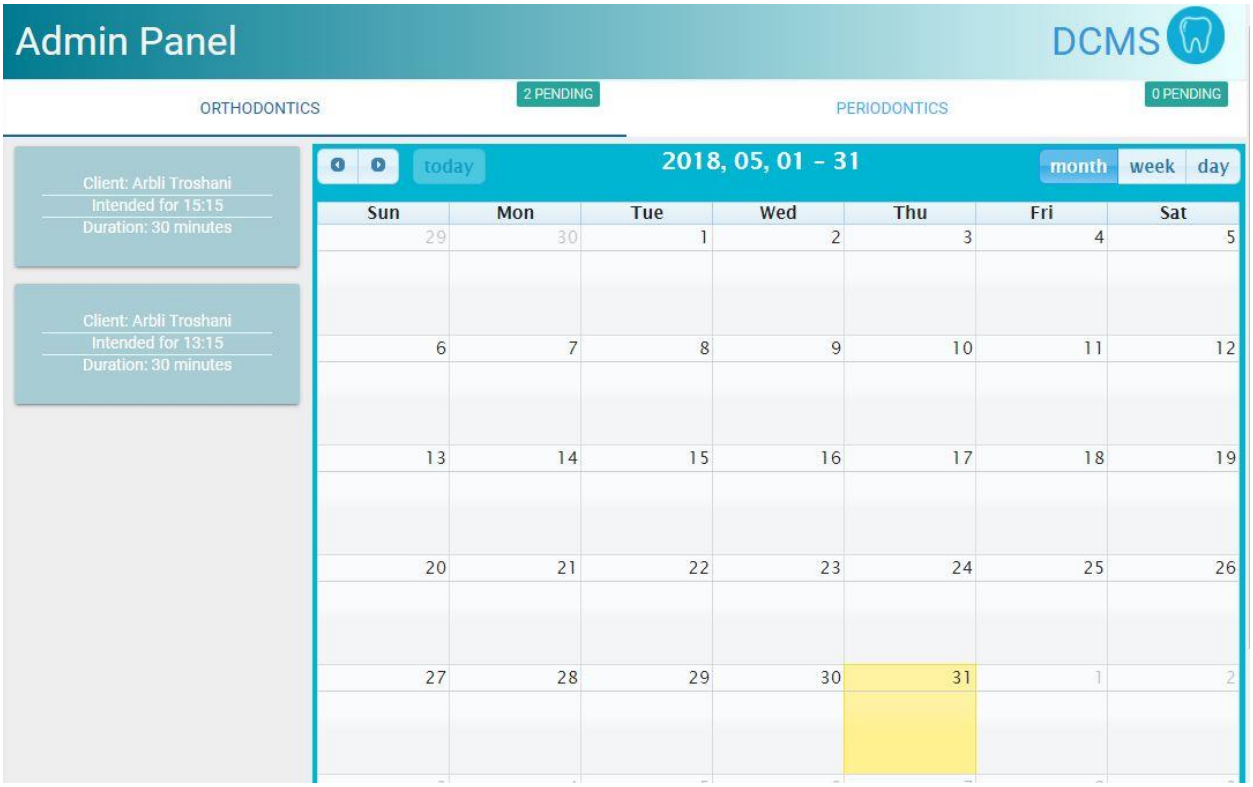
4.12.5. Add User



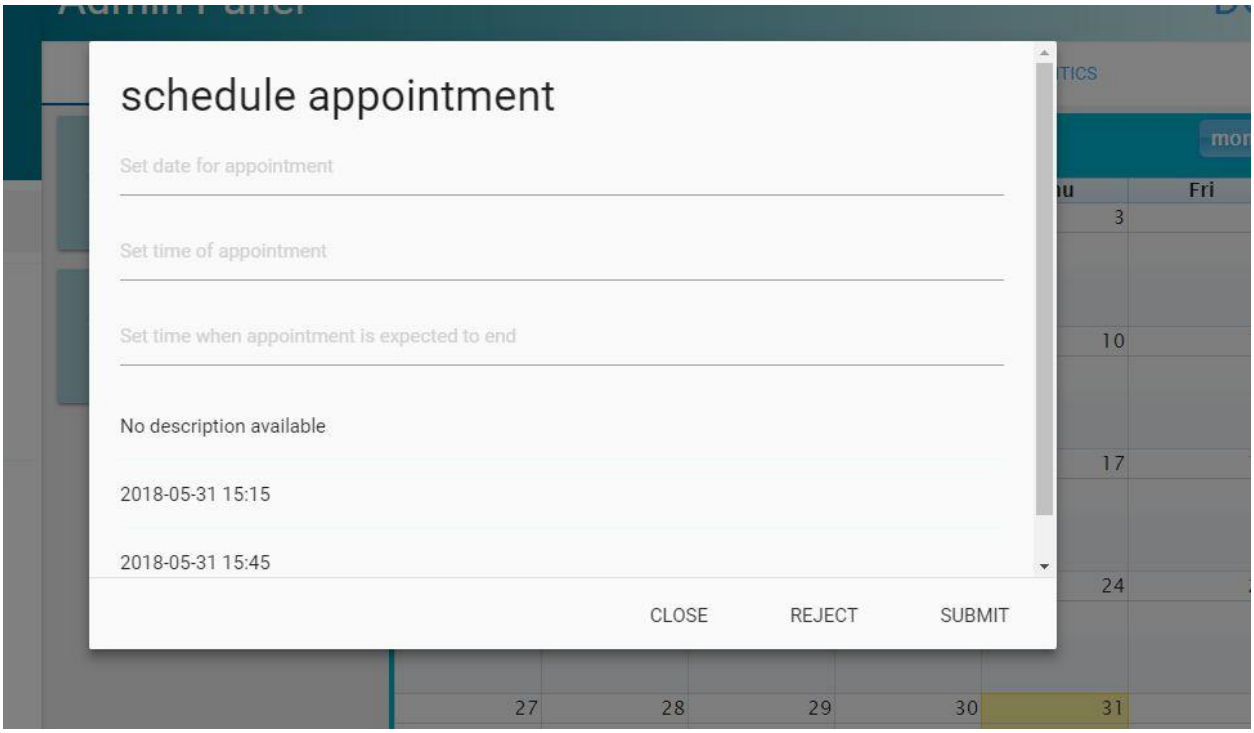
4.12.6. Services



4.12.7. Appointments



4.12.8. Schedule appointment



5. Implementation

5.1. General product overview.

As a basis for the implementation, we have followed the Firebase-recommended specifications, along with the public documentation and API offered by the platform.
To describe the system infrastructure, the following points are made clear:

- The architectural design of the system consists of using a hybrid MVC and PAC design, suited to the scalability and the workload on the system. This means that the design doesn't follow neither MVC nor PAC too strictly, but it makes use of the best of both designs.
- In an MVC architecture, the View is expected to have direct access to the Model, usually through templating. However, this is not possible in the current design, as it would violate security requirements which are upheld through the usage of Middleware components.
- In an PAC architecture, the communication between Presenter and Controller is disallowed, and the Presenter components is updated through the Controller, which serves backend-generated code and has all the routing information. This is the case in this implementation, however unlike traditional PAC, the current implementation uses only one triad of components, where each component is designed as in MVC.

This implementation is generally known as a Hybrid MVC-PAC architecture or HMP.

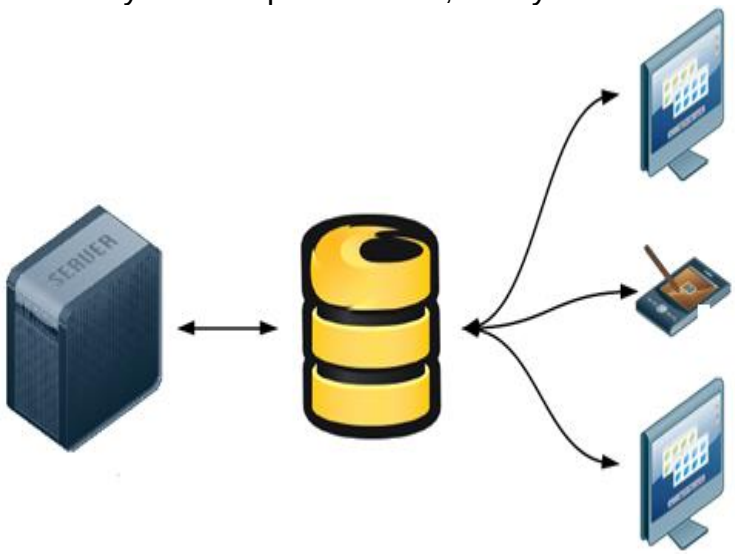
5.2. NodeJS and Firebase

Based on the requirements of the product and the general usage perspective of the system, the approach used to make the implementation scalable, flexible and computationally effective consists of using a customized back-end server, programmed in NodeJS and which can, always, connect to the Firebase system.
Using this pattern provides us with more flexibility, especially when it comes to three main aspects:

1. Integrating third party APIs such as an SMS sending module.
2. Handling advanced authentication requirements, such as manually controlled validations and invalidations of JWT (JSON Web Tokens), which are the standard of authentication for the Firebase.
3. Backend organization and distribution of responsibilities in a modular manner. This means that we can achieve a more refined architectural design of the software, according to our own requirements, and not necessarily follow the general back-endless paradigm offered by Firebase.

In terms of scalability, the distribution of responsibilities between components without affecting querying and database modelling provides us with refactorable units of code and an environment which is easy to alter, whenever needed.

From a systematic point of view, the system has the following structure



In this configuration, Firebase is handling most of the heavy lifting of scale and model duties whereas the server implements the business logic and authentication mechanisms. On the other side, Firebase serves as a conduit between Cloud Firestore, Custom server and Mobile Clients.

5.3. Code Snippets – Web app

5.3.1. *Authentication:*

```
// The start method will wait until the DOM is loaded.

let socket = io('http://localhost:9911'); // send to the index
// process all https requests here
socket.on('connect', function (data) {

    var ui = new firebaseui.auth.AuthUI(firebase.auth());

    ui.start('#firebaseui-auth-container', getUiConfig());


    firebase.auth().onAuthStateChanged(function (user) {

        if (user) {
            console.log(user.toString());
            user.getIdToken(false).then(function (idToken) {

                socket.emit('login_attempt', idToken);
                $('#firebaseui-auth-container').hide();
                $('#loadingDiv').show(200);
                socket.on('redirect', function (destination) {

                    console.log(destination);

                    window.location.href = destination;

                });
            });
        }

    });


});
```

5.3.2. *Routing:*

```
app.get('/', (req, res) => {

    // if user is not logged

    res.redirect(/login/);

});


app.get('/register', (req, res) => {

    fs.readFile('../register.html/', 'UTF-8', (err, data) => {

        if (err) console.log(err);

        res.send(data);
    });

});

app.get('/login', (req, res) => {

    permitted = false;
    fs.readFile('../public/login/login.html/', 'UTF-8', (err, data) =>
{

        if (err) console.log(err);

        res.send(data);
    });

});
```

5.3.3. *Middlewares:*

```
module.exports = function(app){
    app.use('/logo', express.static(__dirname + '/../images/logo'));
    app.use('/android.gif', express.static(__dirname +
'/../images/android.gif'));
    app.use('/favicon', express.static(__dirname +
'/../images/favicon.ico'));
    app.use('/adder', express.static(__dirname +
'/../images/addImage.png'));
    app.use('/dental', express.static(__dirname +
'/../images/dental.gif'));
    app.use('/login.js', express.static(__dirname +
'/login/login.js'));
    app.use('/index.js', express.static(__dirname +
'/home/index.js'));
    app.use('/util', express.static(__dirname + '/home/util.js'));
    app.use('/login.css', express.static(__dirname +
'/login/login.css'));
    app.use('/home.css', express.static(__dirname +
'/home/home.css'));
    app.use('/materialize.css', express.static(__dirname +
'/../css/materialize.css'));
    app.use('/qtip.css', express.static(__dirname +
'/../css/qtip.css'));
    app.use('/materialize.js', express.static(__dirname +
'/../javascript/materialize.js'));
    app.use('/qtip.js', express.static(__dirname +
'/../javascript/qtip.js'));
    app.use('/getSocket', express.static(__dirname +
'/../node_modules/socket.io-client/dist/socket.io.js'));

}
```

5.4. Code Snippets – Android app

5.4.1. Reminder - BroadcastReceiver

```
public class ReminderReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        Intent notificationIntent = new Intent(context, AppointmentsActivity.class);

        TaskStackBuilder stackBuilder = TaskStackBuilder.create(context);
        stackBuilder.addParentStack(AppointmentsActivity.class);
        stackBuilder.addNextIntent(notificationIntent);

        PendingIntent pendingIntent =
                stackBuilder.getPendingIntent( requestCode: 0, PendingIntent.FLAG_UPDATE_CURRENT);

        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
        int minutesBefore = Integer.parseInt(prefs.getString( s: "reminder_before",  s1: "60"));

        MyNotification notification = new MyNotification(
                        pendingIntent,
                        contentTitle: "You have an appointment in " + minutesBefore + " minutes!",
                        contentText: ":)",
                        context);

        notification.send(MyNotification.NOTIFICATION_REMIND);
    }
}
```

5.4.2. SessionsFragment

```
    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        recyclerView.setHasFixedSize(true);
        layoutManager = new LinearLayoutManager(getActivity());
        recyclerView.setLayoutManager(layoutManager);

        db.collection( s: "treatments")
                .document(getArguments().getString(TREATMENT_ID_KEY))
                .collection( s: "sessions")
                .addSnapshotListener((snapshot, e) -> {
                    if (e != null) return;
                    if (snapshot != null) {
                        myDataset = snapshot.toObjects(Session.class);
                        adapter = new SessionsAdapter(myDataset);
                        recyclerView.setAdapter(adapter);
                    }
                });
    }
```

5.4.3. Add new appointment

```
db.collection( s: "appointments")
        .add(new FirebaseAppointmentCalendarEvent(
                context: this,
                etDescription.getText().toString(),
                Constants.Appointments.STATUS_PENDING,
                spinner.getSelectedItem().toString(),
                isForSelf: true,
                myCalendar.getTimeInMillis(),
                durationMinutes: 30
        ))
        .addOnSuccessListener(documentReference -> {
            bSubmit.doResult( isSucceed: true);
            Intent returnIntent = new Intent();
            returnIntent.putExtra( name: "result", Constants.Appointments.RESULT_OK);
            new Handler().postDelayed(() -> {
                setResult(Activity.RESULT_OK, returnIntent);
                finish();
            }, delayMillis: 600);
        })
        .addOnFailureListener(e -> bSubmit.reset());
```

5.4.4. User sign in

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == RC_SIGN_IN) {
        invalidateOptionsMenu();
        IdpResponse response = IdpResponse.fromResultIntent(data);
        if (resultCode == RESULT_OK) {
            FirebaseUserMetadata metadata = auth.getCurrentUser().getMetadata();
            if (metadata.getCreationTimestamp() == metadata.getLastSignInTimestamp()) {
                // New user
                Intent i = new Intent( packageContext: MainActivity.this, IdCardScanActivity.class);
                startActivity(i);
            } else {
                // Existing user
                showSnackbar("Welcome back");
                FirebaseFirestore.getInstance()
                        .collection( s: "users")
                        .whereEqualTo( s: "uid", auth.getCurrentUser().getUid())
                        .get()
                        .addOnCompleteListener(task -> {
                            for (QueryDocumentSnapshot document : task.getResult()) {
                                Utility.setNationalIdSharedPreference( context: MainActivity.this, document.getId());
                                Utility.setLoggedInUser( context: MainActivity.this, document.toObject(User.class));
                                replaceFragment(R.id.nav_home);
                            }
                            refreshNavigationDrawer();
                        });
            }
            updateRegistrationToken();
            drawer.closeDrawer(GravityCompat.START);
            replaceFragment(R.id.nav_home);
```

5.5. Code Snippets – Cloud Functions

5.5.1. Sending welcome email

```
'use strict';

const functions = require('firebase-functions');
const nodemailer = require('nodemailer');
const admin = require('firebase-admin');
admin.initializeApp();

const APP_NAME = 'Dental Clinic Management System';

const mailTransport = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: functions.config().gmail.email,
    pass: functions.config().gmail.password,
  },
});

exports.sendWelcomeEmail = functions.auth.user().onCreate((user) => {
  const email = user.email;
  const displayName = user.displayName;
  return sendWelcomeEmail(email, displayName);
});

function sendWelcomeEmail(email, displayName) {
  const mailOptions = {
    from: `${APP_NAME} <noreply@firebase.com>`,
    to: email,
  };
  mailOptions.subject = `Welcome to ${APP_NAME}!`;
  mailOptions.text = `Hey ${displayName || ''}! Welcome to ${APP_NAME}. I hope
    you will enjoy our service.`;
  return mailTransport.sendMail(mailOptions).then(() => {
    return console.log('New welcome email sent to:', email);
  });
}
```

5.6. Testing

To test the application's stability, we had to perform several tests on the custom backend. These tests included the handling of Exceptions, Memory Leaks and Load Testing.
For a start, the tools used to perform testing are
1. Express Framework's built in module called SuperTest
2. An HTTP request generator called Postman.
Using the SuperTest module, we were able to conclude that all predefined routes work correctly and there are no authentication problems such as redirecting or premature session expiration.
Using the Postman module, we were able to conclude that the server can comfortably handle up to 10000 concurrent requests to different routes.
Furthermore, these tests revealed that there were no possible security breaches, especially when serving code from the server side, since the code is sent through an encrypted communication channel as prescribed by the WebSockets standard.
In cases when the server serves static files such as images or scripts, the risks of altering these files are alienated by default, due to the usage of Middleware components offered by the Express Framework.

5.6.1. *Snippets*

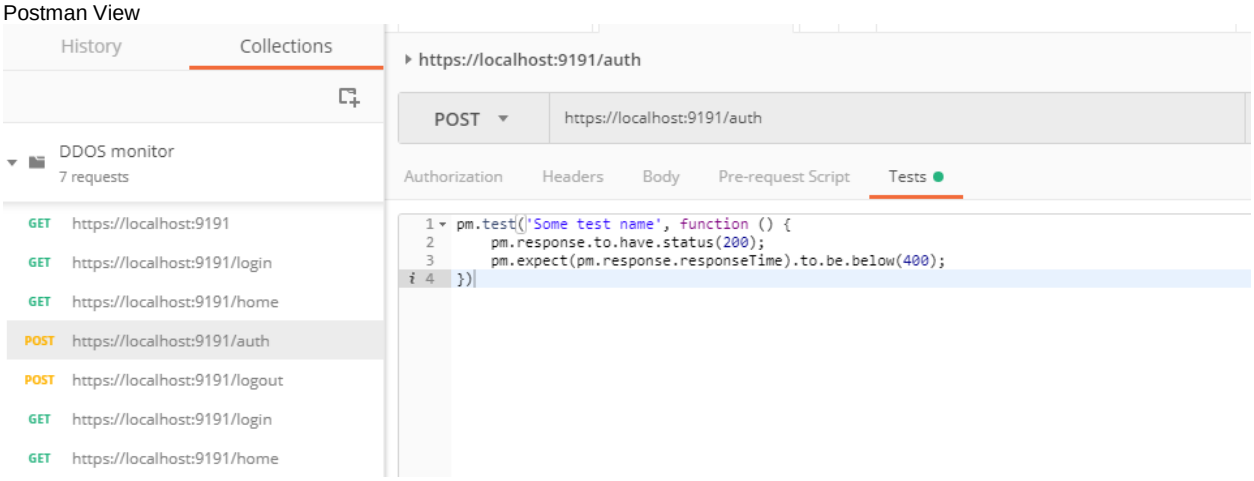
Route unit testing for Exceptions & Leaks

```
const request = require('supertest');
const express = require('express');

const app = express();

app.get('/login, function(req, res) {
  res.status(200).json({ feedback: 'found' });
});

request(app)
  .get('/logi)
  .expect('Content-Type', /json/)
  .expect('Content-Length', '15')
  .expect(200)
  .end(function(err, res) {
    if (err) throw err;
  });

describe('POST /home, function() {
  it('responds with json', function(done) {
    request(app)
      .post('/home)
      .send({status: 'can validate JWT'})
      .set('Accept', 'application/json')
      .expect(200)
      .end(function(err, res) {
        if (err) return done(err);
        done();
      });
  });
});
```

Load Testing

```
GET https://localhost:9191/login
pm.test('login test', function () {
    pm.response.to.have.status(200);
    pm.expect(pm.response.responseTime).to.be.below(200);
})

GET https://localhost:9191/home
pm.test('home test', function () {
    pm.response.to.have.status(200);
    pm.expect(pm.response.responseTime).to.be.below(400);
})

POST https://localhost:9191/auth
pm.test('Authentication test', function () {
    pm.response.to.have.status(200);
    pm.expect(pm.response.responseTime).to.be.below(400);
})
```

Postman View



5.7. Technology stack

- Cloud Firestore and Realtime Database for data persistence
- Cloud Storage for storing files
- Cloud Messaging and Cloud Functions for sending notifications and other backend triggered operations in the database
- Twilio for SMS
- Express framework for Node.JS
- Web Sockets for asynchronous communication in the web app

5.8. Open Source Libraries used

5.8.1. Android app

- Tibolte/AgendaCalendarView for the calendar
- BlinkID/blinkid-android for extracting data from ID cards
- firebase/FirebaseUI-Android for simpler authentication and storage
- bumptech/glide for image loading and caching
- chrisbanes/PhotoView full screen image viewer, with zooming support
- JakeWharton/butterknife view binding using annotation processing
- rbro112/Android-Indefinite-Pager-Indicator indefinite pager indicator for RecyclerViews
- Someonewow/SubmitButton
- google/gson convert Java objects to JSON
- facebook/Shimmer shimmer effect on text and other views

5.8.2. Web app

- FullCalendar.io for the calendar
- Socket.io for Realtime backend-frontend communication
- firebase/FirebaseUI-Web for simpler authentication and JWT validation
- JQuery-UI for frontend functionality enhancement
- MaterializeCSS for the main UI design

5.9. Versioning

- Git and GitHub for version control
- Gradle for Android building and versioning

5.10. License

5.11. Instructions of use

5.11.1. Android app

The android application comes in a compressed format: “apk”.
To install the “apk” file, you should enable the installation of apps from unknown sources in Settings -> Security.

No further configuration is needed to use the application. You can login using your mobile phone number (similar flow as in “WhatsApp” application), or you can use a Google account (preferred).

5.11.2. Web App

The web application is accessible on the internet, using port 9191 by default.

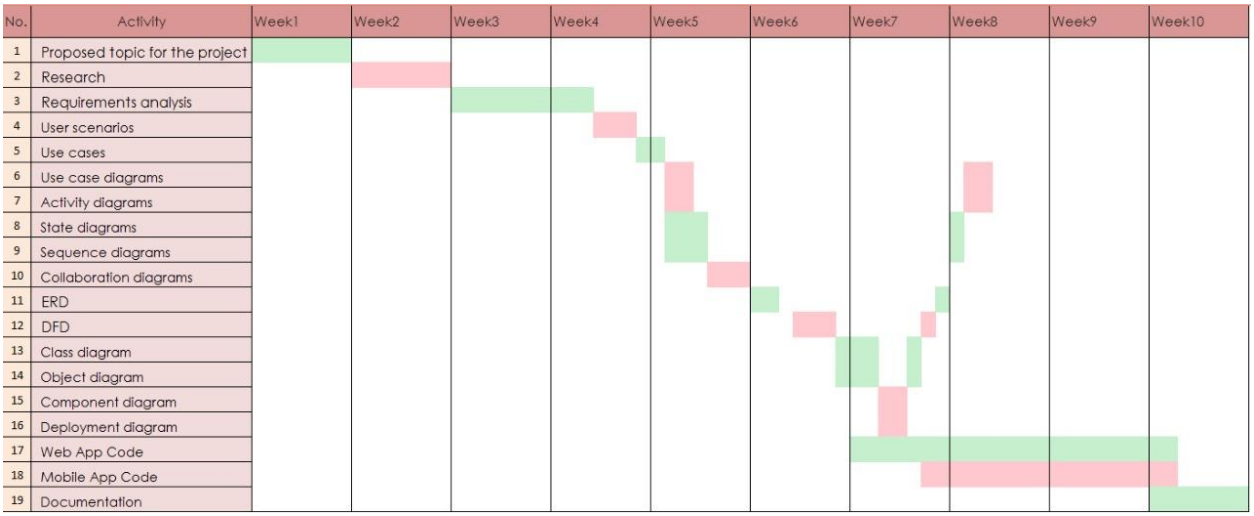
To gain access to the system, one must be identified as an administrator.

The registration as an admin is performed manually for security reasons. After the registration, the admin user can access the system through their google account.

6. Project Management (SPM)

The distribution of tasks over time for the realization of the software is described as follows:

6.1 Gantt Chart



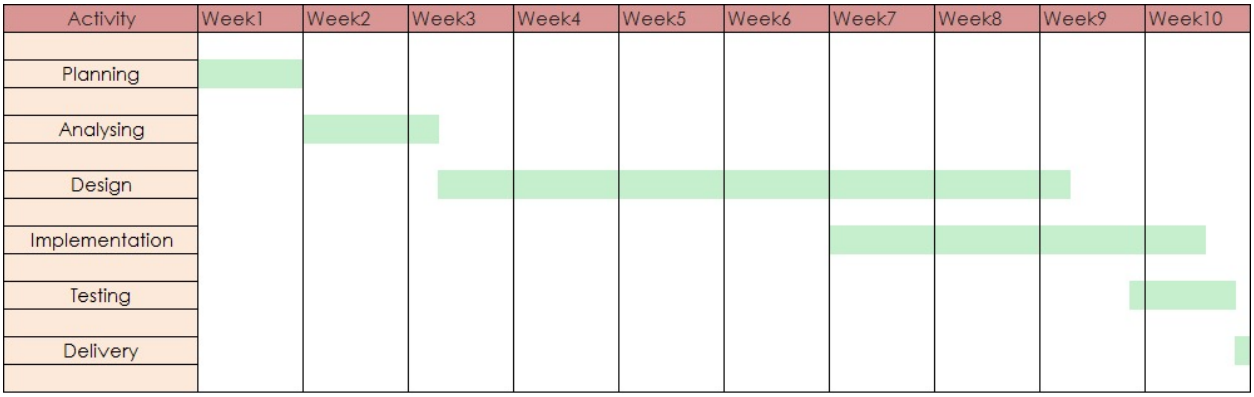
6.2 Task distribution Chart

To accomplish all tasks, possibly before the due dates, the distribution of tasks among team members was as follows:

No.	Activity	Duration (Days)	Dependencies	Worked
1	**Proposed topic for the project**	**7**		All members
	Team organization			
2	**Research**	**7**	1	All members
	Meet with clients			
	Determine technologies			
3	**Requirements analysis**	**10**	2	All members
	Functional			
	Non-Functional			
	Domain			
	Software analysis	**5**		All members
4	User scenarios	3	3	
5	Use cases	2	3,4	
	Behavioral Diagrams	**15**		
6	Use case diagrams	6	5	All members
7	Activity diagrams	6	6	Arbli Troshani
8	State diagrams	6	6	Gerd Alliu
9	Sequence diagrams	6	7,8	Gerd Alliu
10	Collaboration diagrams	3	9	Enxhi Ferhati
11	**ERD**	3	6	Arbli Troshani
12	**DFD**	4	6	Egi Gjevori
	Structural Diagrams	**12**		
13	Class diagram	4	10	Egi Gjevori
14	Object diagram	4	13	Enxhi Ferhati
15	Component diagram	2	14	Arbli Troshani
16	Deployment diagram	2	15	Egi Gjevori
	Implementation	**25**		
17	Web App Code	10	10	Gerd Alliu, Enxhi Ferhati
18	Mobile App Code	15	10,17	Arbli Troshani _Egi Gjevori
	Review	**7**		
19	Documentation		17,18	All members

6.3 Software development lifecycle

To achieve faster prototyping, we organized our work based on the Agile software development lifecycle. Using this approach, we were able to spend time on several tasks simultaneously, while repeatedly making improvements on the design, implementation and performance aspects. This is also shown in the diagram below.



In addition, the Agile model helped us achieve more in a shorter time, as it provided us with intuitive self-organization and distribution of responsibilities, as well as a strong and direct communication pattern.

From the perspective of flexibility, it has been highly evident that taking an agile approach in development helped us refactor parts of the design as well as of the implementation, improving the performance and overall quality of the product. This was done based on repeated inquiry to the potential customer, which would verify the software in terms of design and functionality.

It is also worth mentioning that, given the dynamic nature that the product is expected to have to suit ever changing requirements, our work model was well suited for responding to changes.

As per the implementation of this model, the following can be said:

- The Scrum Master was elected periodically, based on the current state of the overall cycle.
- There have been three major sprints, where each sprint targeted one milestone aspect of the overall product.
- There has been daily reporting and discussion, either in person or virtually, in which each team member would express their concerns and current state of work.
- There has been a weekly review of the design.
- The implementation has been performed incrementally, with group members switching tasks and attention from one aspect to the other, guaranteeing higher quality code.

6.4 Network Diagram

To visualize the planning process and the critical sections of the development, the following diagram has been used.