DocCare

Software Architecture Document

Version 1.3

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 30/11/2023 | 1.0 | The first version of Software Architecture Document | DocCare Team |
| 1/12/2023 | 1.1 | Adding Introduction, Architectural Goals and Constraints, Use Case model | DocCare Team |
| 1/12/2023 | 1.1 | Update on Logical View Diagram  User model and User Service and Controllers Storage Service and Storage Controllers | Phat Tran |
| 1/12/2023 | 1.1 | Adding Diagram for Logical View: Appointment Model + Service, Specialization Model + Service, Authentication Controller | Phuc Chu |
| 1/12/2023 | 1.1 | Adding Diagram for Logical View:  Prescription Model + Service,  Working Shift Model + Service,  Category Model + Service | Nghia Nguyen |
| 1/12/2023 | 1.1 | Adding Diagram for Logical View: Statistics Model + Service,  Absent Request Model + Service,  Admin Controller + Service  Customer Controller | Quoc Tran |
| 1/12/2023 | 1.1 | Adding Diagram for Logical View:  Intake Model + Service,  Medicine Model + Service,  Doctor + Receptionist Controller | Vinh Nguyen |
| 2/12/2023 | 1.2 | Review all parts | DocCare Team |
| 14/12/2023 | 1.3 | Update project architecture, deployment plan, and implementation view | Phat Tran |

Table of Contents

[**1. Introduction 4**](#_heading=h.gjdgxs)

[**2. Architectural Goals and Constraints 4**](#_heading=h.30j0zll)

[**3. Use-Case Model 5**](#_heading=h.1fob9te)

[**4. Logical View 5**](#_heading=h.3znysh7)

[4.1 Package: Controllers 7](#_heading=h.wpmc4yn6ogj9)

[4.1.1. Administrator 7](#_heading=h.yvi7g6x2f8fr)

[4.1.2. Customer 8](#_heading=h.43t040atk6ig)

[4.1.3 Receptionist 8](#_heading=h.ftd8nguej69r)

[4.1.4 Doctor 9](#_heading=h.ybef4lrr862v)

[4.2 Package: Services 12](#_heading=)

[4.2.1. User 12](#_heading=h.bggc7rcd06k9)

[4.2.2. Storage 12](#_heading=h.fhwuz4p4gkq0)

[4.2.3 Administration 13](#_heading=h.dfu9vekh0my8)

[4.2.4 Intake 14](#_heading=h.hs0afksg80b4)

[4.2.5 Medicine 14](#_heading=h.s8j9ol6smwdw)

[4.2.6 Specialization 15](#_heading=h.5dn8ekgcvp2x)

[4.2.7 Category 16](#_heading=h.q64gyz9cqjy)

[4.2.8 Appointment 17](#_heading=h.y775nwp4ss5p)

[4.2.9 Statistics 18](#_heading=h.r5i8muen44jr)

[4.2.10 Absent Request 19](#_heading=h.fq9sog2n2hlh)

[4.2.11 Prescription 20](#_heading=h.r80gxfdgxuyr)

[4.2.12 Working Shift 21](#_heading=h.z7jp1ud5jlyd)

[4.3 Package: Models 21](#_heading=h.sd3svggh7nl1)

[4.3.1 Medicine 21](#_heading=h.e7xsum1s7zot)

[4.3.2 Intake 22](#_heading=h.ljp22xaduqys)

[4.3.4 Appointment 23](#_heading=h.r3b4c8al4lme)

[4.3.4 Specialization 23](#_heading=h.k7c42dr90tzy)

[4.3.5 Absent Request 24](#_heading=h.vpwxdydcmtve)

[4.3.6 Working Shift 24](#_heading=h.5nsk8k57ifw8)

[4.3.7 Category 24](#_heading=h.f0wapwoqw1ke)

[4.3.8 Prescription 25](#_heading=h.h6vh9ocbfqoj)

[4.3.9 Statistic 25](#_heading=h.pshb9xgehpev)

[**5. Deployment 25**](#_heading=h.tyjcwt)

[**6. Implementation View 25**](#_heading=h.3dy6vkm)

Software Architecture Document

# Introduction

This document describes the overall architecture design, structure of the DocCare mobile application. It serves as a blueprint for the system's development and provides a shared understanding among stakeholders.

1. Purposes:
   1. Provides a clear and concise description of the system's architecture to all stakeholders, including developers, testers, project managers, and customers.
   2. Serves as a reference point for the development team, ensuring that all components are designed and implemented in a consistent manner.
   3. Provides a framework for understanding the impact of proposed changes on the system's architecture.
2. Scope: This document describes architectural goals and constraints, use-case models, architectural views, deployment strategy, and implementation view.

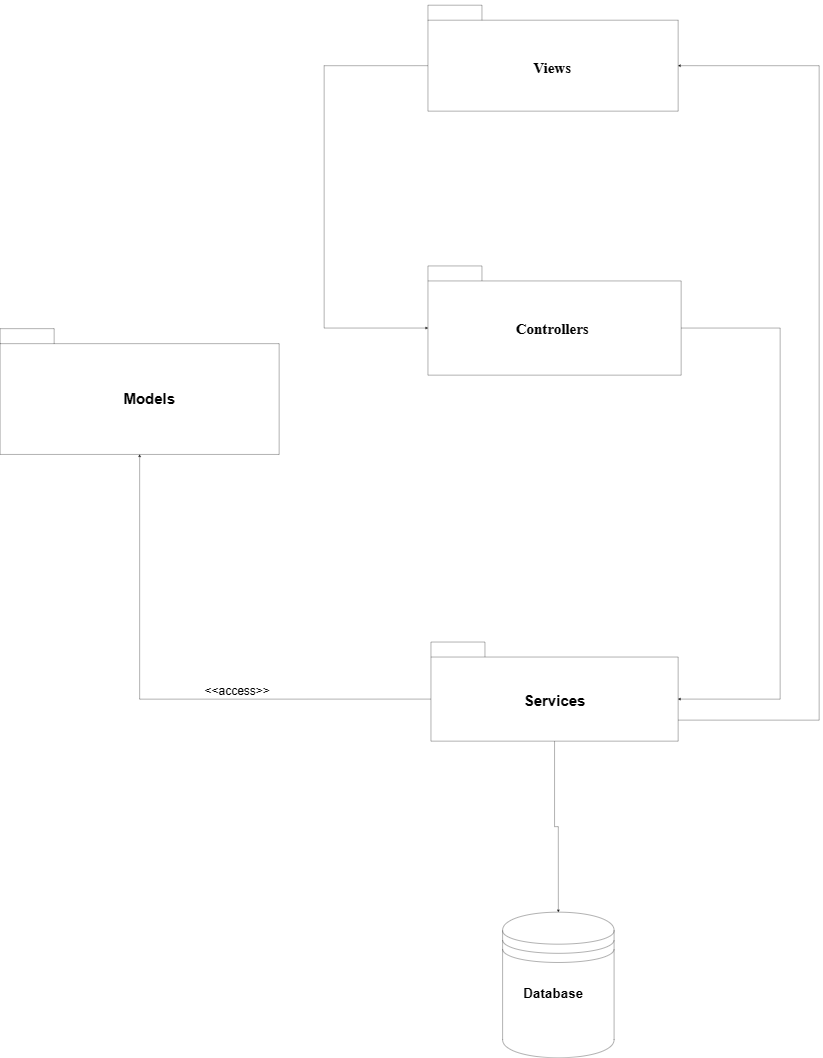
# Architectural Goals and Constraints

* The application is developed using the Flutter framework.
* Support on Android mobile phones only.
* The architecture must follow Flutter recommended pattern: Clean Architecture.

# Use-Case Model

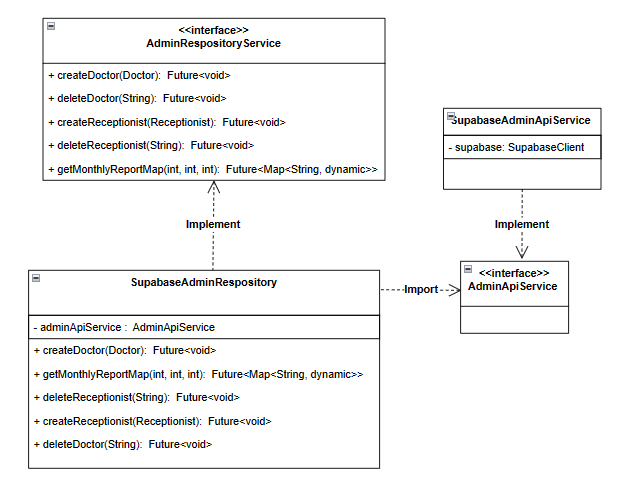


# Logical View



## Package: Controllers

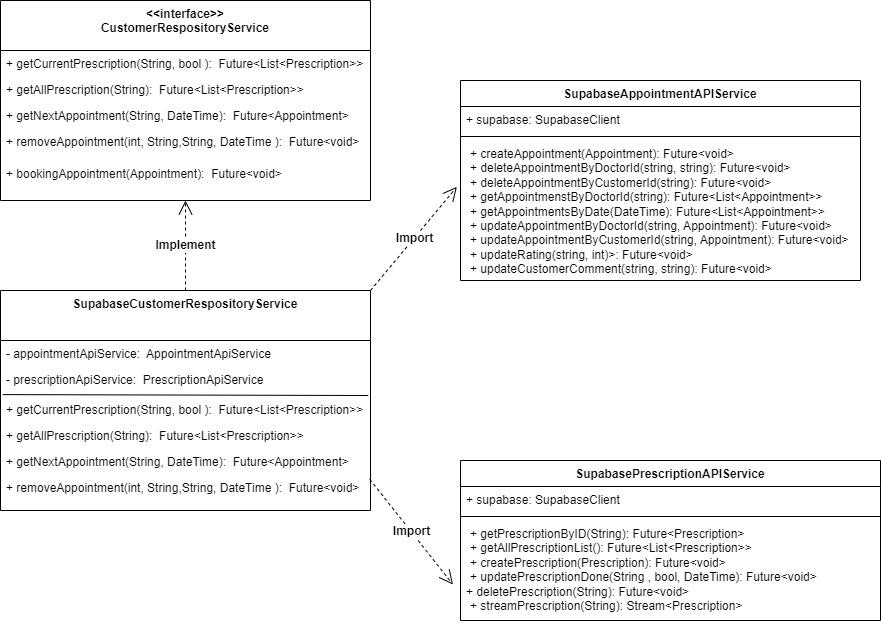
### **4.1.1. Administrator**



**AdminRespositoryService**: provides methods for admin add and remove staff.

**SupabaseAdminRespository**: Implement the above interface.

### **4.1.2. Customer**



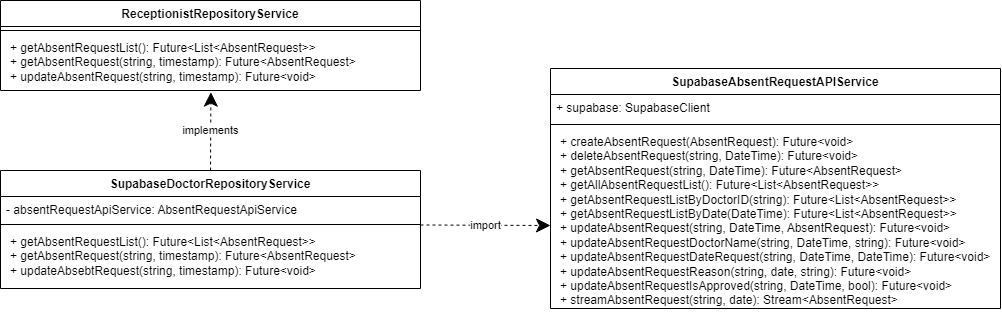
**CustomerRespositoryService:** provides methods for customers to get the current prescription, get all the prescriptions and booking, remove and get the next appointment.

**SupabaseCustomerRespositoryService**: Implement the above interface.

**SupabaseAppointmentAPIService**: providing methods to work with appointments.

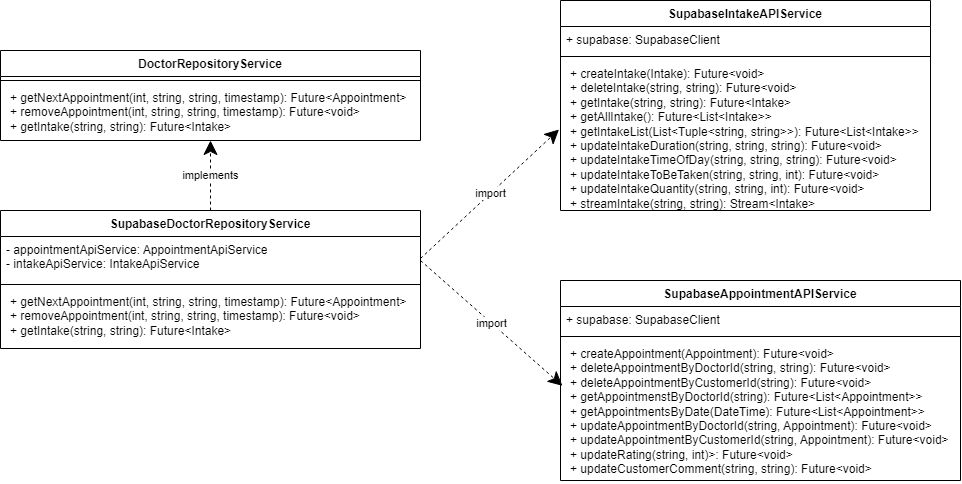
**SupabasePrescriptionAPIService**: providing methods to work with Prescription.

### **4.1.3 Receptionist**



* **ReceptionistRepositoryService** provides methods for receptionists to retrieve and update related information about their field of work.
* **SupabaseAbsentRequestApiService** provides methods to work with absent requests tables in the database, in order to retrieve, update, or stream data.

### **4.1.4 Doctor**

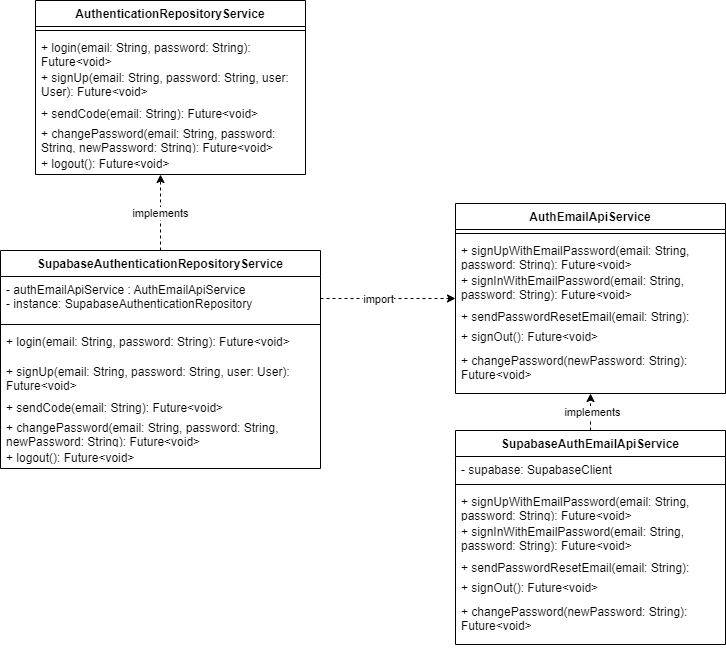


**DoctorRepositoryService** provides methods for doctor to create or get intake information from a patient,

get their appointment schedule or sent an absent request, waiting for approval.

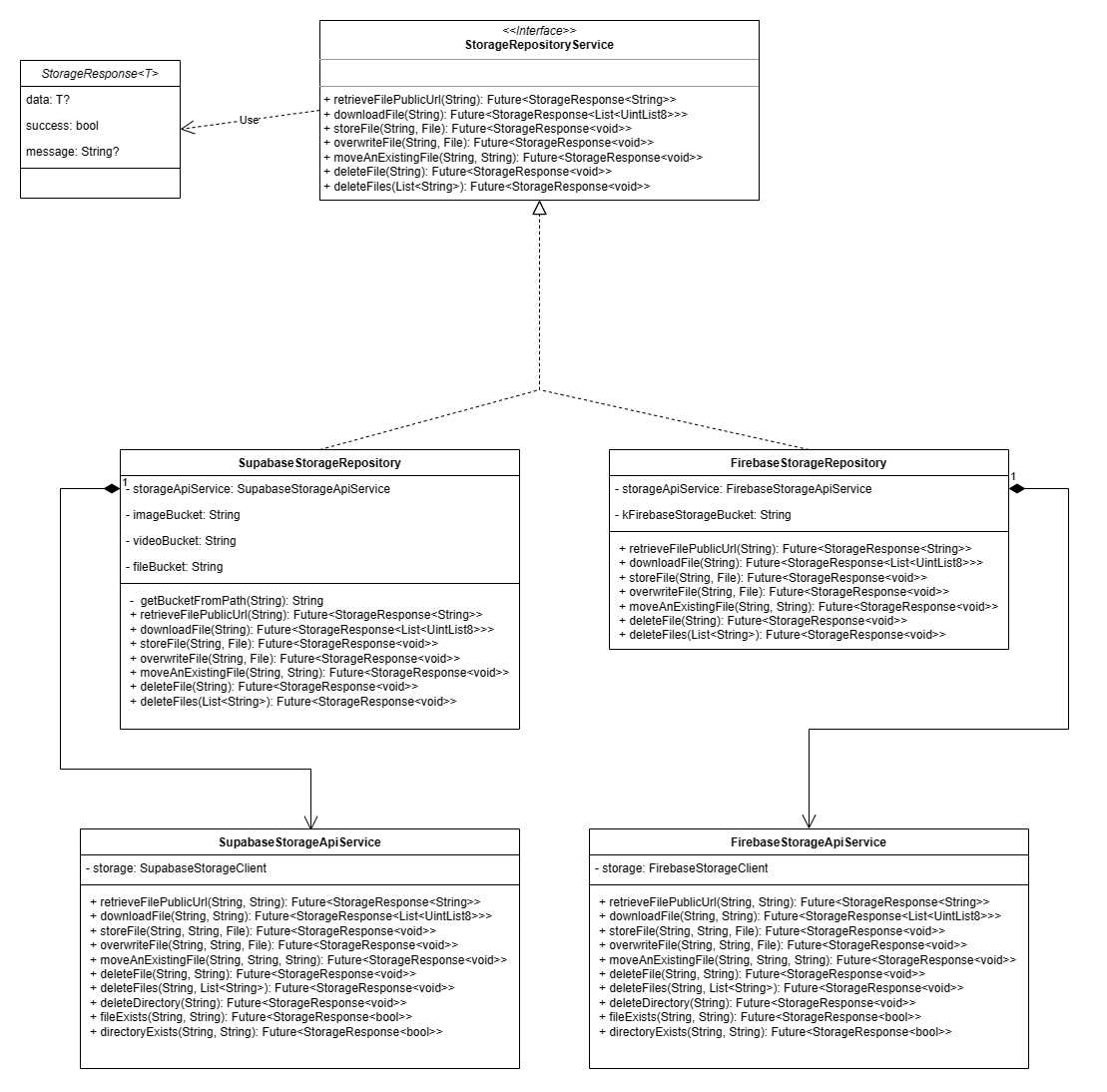
* **SupabaseAppointmentApiService** provides methods to work with appointment scheduling
* **SupabaseIntakeApiService** provides methods to work with create, retrieve or update information of intakes.

**4.1.5 Authentication**

****

* **AuthenticationRepositoryService** provides methods for clients to authenticate to the application.
* **AuthEmailApiService** provides methods to work with email like sign up, sign in, send email,....
* **SupabaseAuthenticationRepositoryService, SupabaseAuthEmailApiService** are the implementation of the above classes.

**4.1.6 Storage**

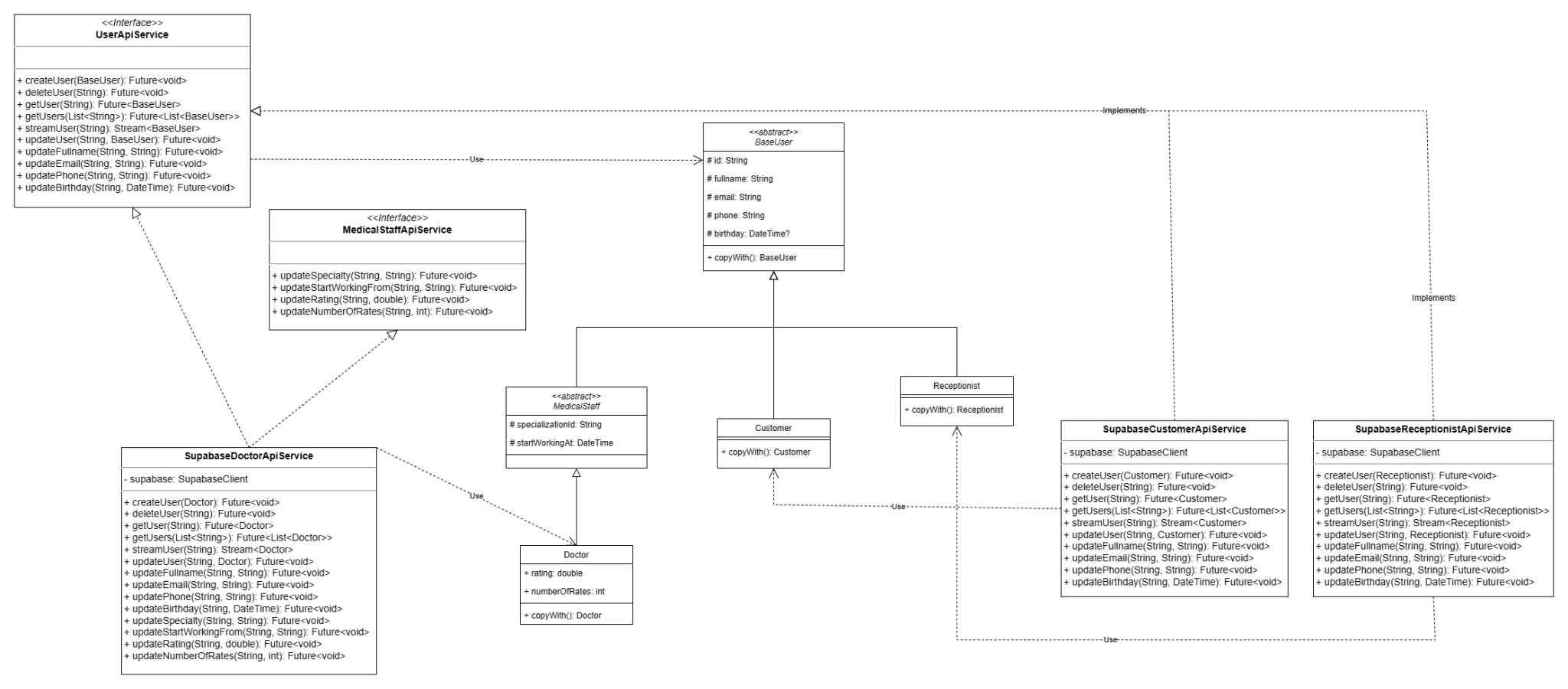
****

**StorageRepositoryService** interface defines methods which facilitate the client to interact with the data stored in the server. It provides the essential functionality which each storage repository must have. Its methods now do not need the bucket name any more as it is delegated to its concrete versions to decide the name.

**SupabaseStorageRepository, FirebaseStorageRepository** are the two concrete implementations of the above interface. Each version has its own service to call when implementing the methods of the interface. The service lives inside each version and disappears when the repository is disposed; therefore the relationship is composite.

## Package: Services

### **4.2.1. User**

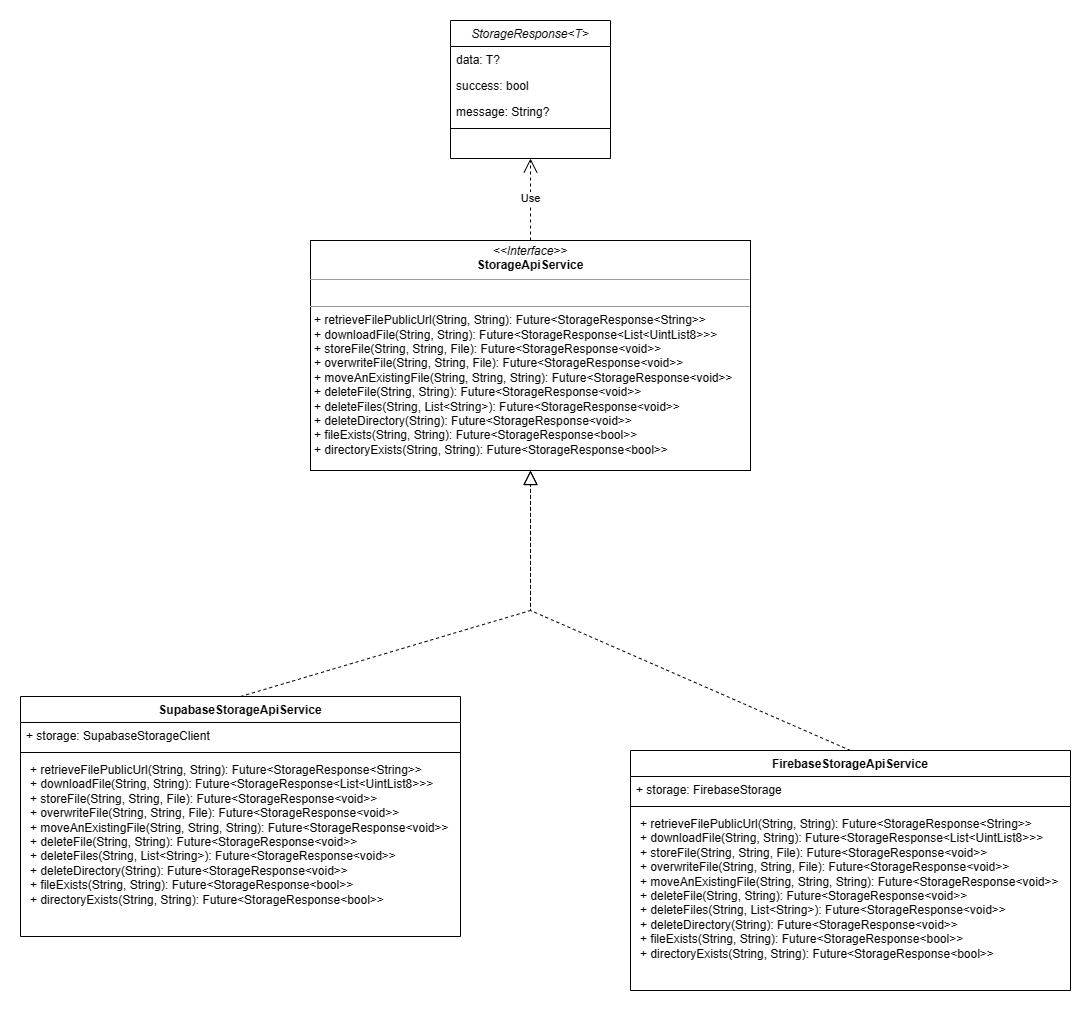
****

This module includes the service interface for the CRUD operations with **BaseUser** data type and its subclasses:

1. **UserApiService**: The interface includes methods to create, update each attribute of the model, methods to retrieve a snapshot of the BaseUser, or multiple users, and methods to stream data of the user. All methods require at least an id of the user to function correctly.
2. **MedicalStaffApiService**: this interface adds additional methods to deal with the data model of doctors, nurses (medical staff).
3. **SupabaseCustomerApiService, SupabaseReceptionistApiService**: these concrete implementations of the **UserApiService** interface are customized to communicate with Supabase database servers. They contain the client of Supabase to use the provided A.P.I from Supabase.
4. **SupasbaseDoctorApiService**: this implements **MedicalStaffApiService** and **UserApiService** and serves the same purposes with the above 2 classes.

This module depends on the models of the user.

### **4.2.2. Storage**



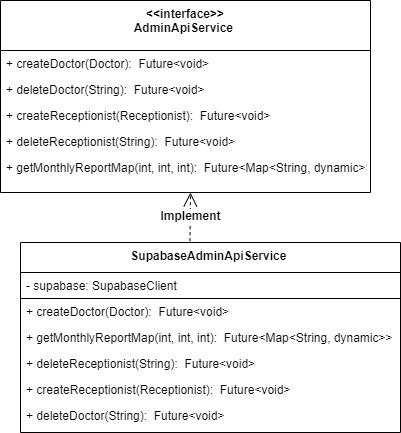
This module includes a data class to model the response of the API calls from the client to the server of any storage in use, the interface defining essential methods for the controllers to store, delete, retrieve images, files, videos from the storage, and its concrete implementation for Supabase and Firebase.

Each method requires the bucket name and the path of the desired files to retrieve, delete or remove the corresponding files.

We have to implement 2 versions of storage service to meet the risk management strategy listed in the Project Plan document.

### 

### **4.2.3 Administration**



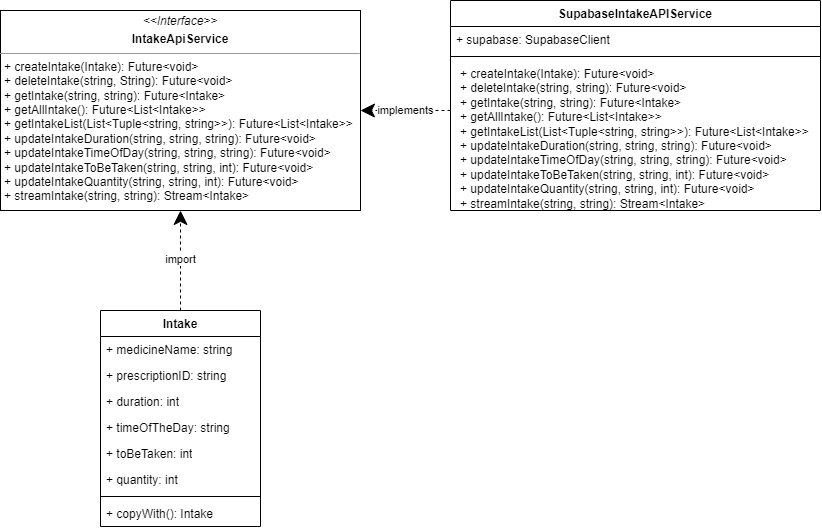
**AdminApiService :** The interface includes methods to add and remove doctors and receptionists.

**SupabaseIntakeApiService:** It’s a concrete implementation of the **AdminApiService** interface, used to

communicate with Supabase database servers. They contain the client of Supabase to use the provided A.P.I

from Supabase.

### **4.2.4 Intake**



**IntakeApiService:** The interface includes methods to create, retrieve, and update attributes of the intake

model. It also contains a method to stream data of that intake. All methods require a prescription ID and a

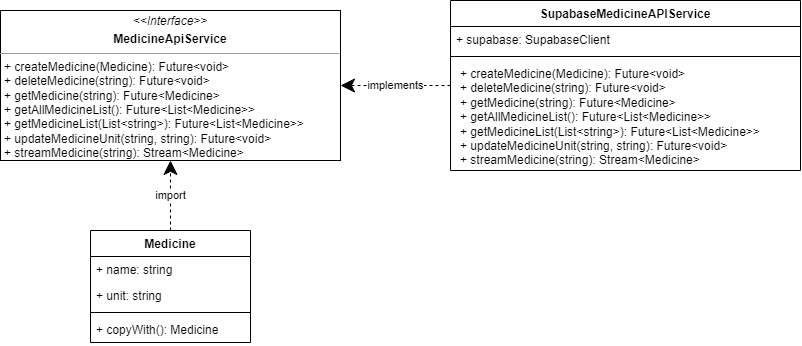
medicine name to work.

**SupabaseIntakeApiService:** It’s a concrete implementation of the **IntakeApiService** interface, used to

communicate with Supabase database servers. They contain the client of Supabase to use the provided A.P.I

from Supabase.

### **4.2.5 Medicine**



**MedicineApiService:** The interface includes methods to create, retrieve, and update attributes of the

medicine model. It also contains a method to stream data of that medicine. All methods require a

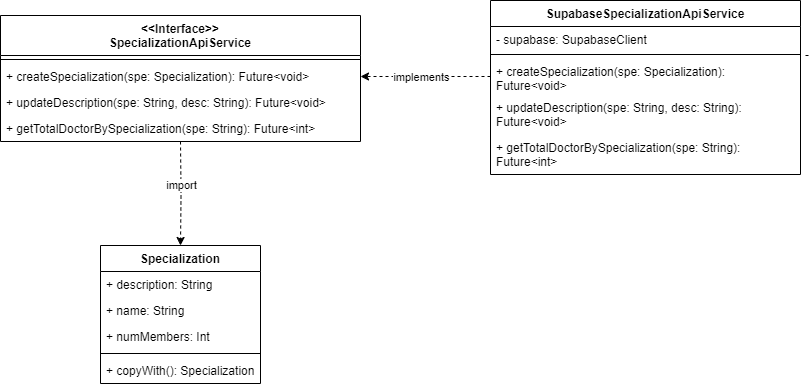
medicine name to work.

**SupabaseMedicineApiService:** The concrete implementation of the **MedicineApiService** interface, used

to communicate with Supabase database servers. They contain the client of Supabase to use the provided

A.P.I from Supabase.

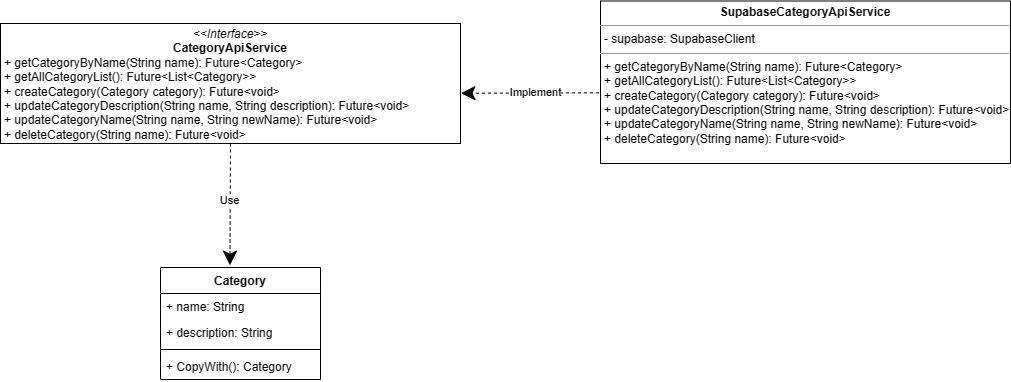
### **4.2.6 Specialization**



* **Specialization:** this is the data class
* **SpecializationApiService:** this is an interface contains methods for the class Specialization to work with the database:
  + createSpecialization: receive an object of type Specialization and add it to the database
  + getTotalDoctorBySpecialization: receive an object of type String, which is the name of the specialization, and return the number of doctors with that specialization
  + updateDescription: receive the specialization’s name as String and the description of it, update it to the database.
* **SupabaseSpecializationApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

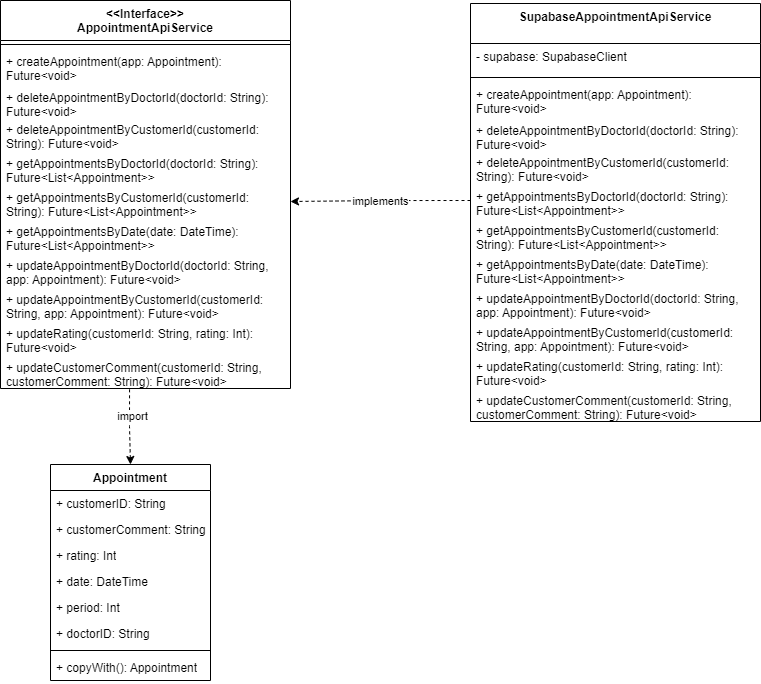
### 

### **4.2.7 Category**



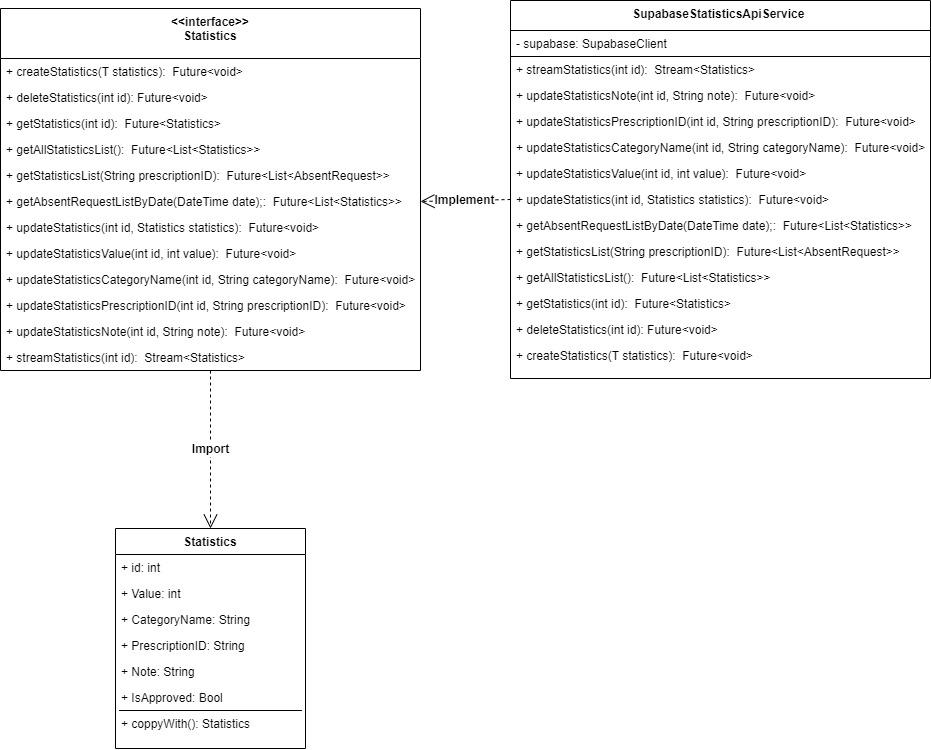
* **Category**: this is the data class
* **CategoryApiService:** this is an interface contains methods for the class Category to work with the database:
  + getCategoryByName: receive category name of type String, and return a list of all categories.
  + getAllCategoryList: receive the list of all Category from the database
  + createCategory: receive an object of type category and add it to the database
  + updateCategoryDescription: receive name and description of type string and update that category description corresponding to name
  + updateCategoryName: receive name and newName of type string and update that category newName corresponding to name
  + deleteCategory: receive category name of type String and delete category having that name.
* **SupabaseCategoryApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

### **4.2.8 Appointment**



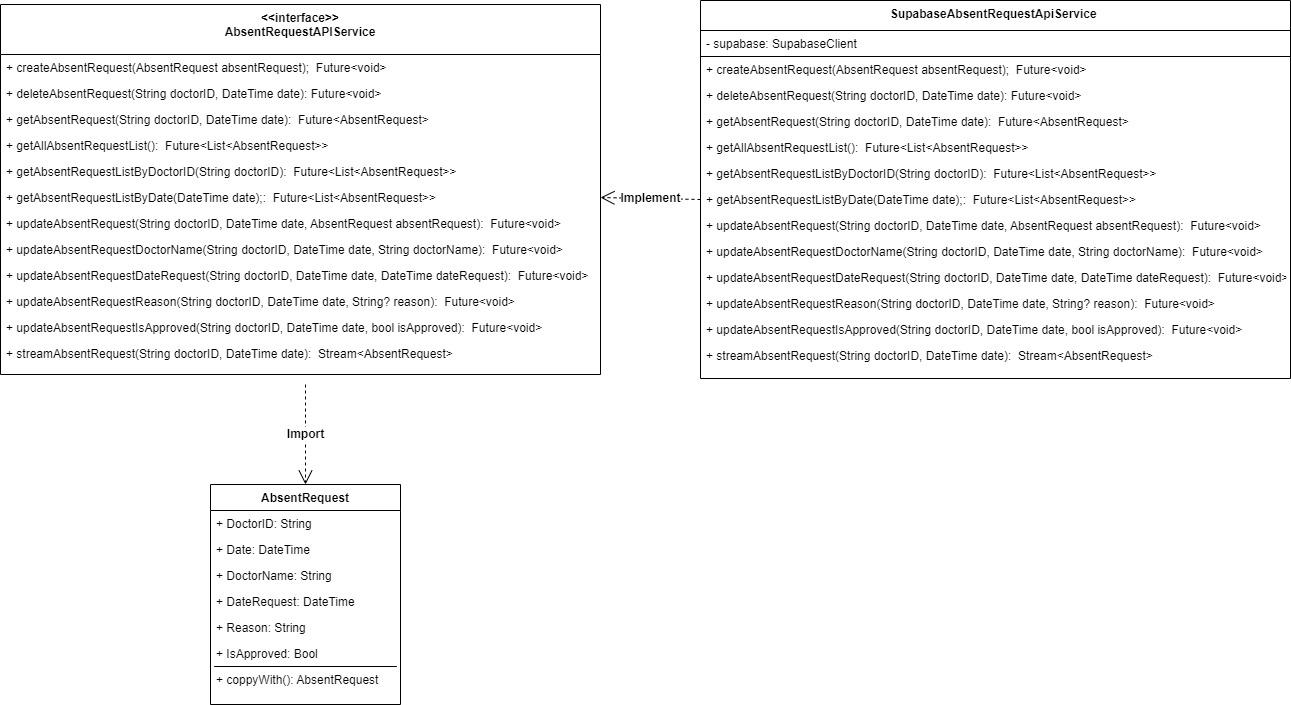
* **Appointment**: this is the data class
* **AppointmentApiService:** this is an interface contains methods for the class Appointment to work with the database:
  + createAppointment: receive an object of type appointment and add it to the database
  + deleteAppointmentByDoctorId: receive an object of type String, which is the ID of the doctor, and delete all appointments of that doctor.
  + deleteAppointmentByCustomerId: receive an object of type String, which is the ID of the customer, and delete all appointments of that customer.
  + getAppointmentsByDoctorId: receive an object of type String, which is the ID of the doctor, and return a list of all appointments of that doctor.
  + getAppointmentsByCustomerId: receive an object of type String, which is the ID of the customer, and return a list of all appointments of that customer.
  + getAppointmentsByDate: receive an object of type DateTime, which is date, and return a list of all appointments of that date.
  + updateAppointmentByDoctorId: receive an object of type String, which is the ID of the doctor and an object of type Appointment, update that appointment with new fields to the database.
  + updateAppointmentByCustomerId: receive an object of type String, which is the ID of the customer and an object of type Appointment, update that appointment with new fields to the database.
  + updateRating: receive an object of type String, which is the ID of the customer and an integer from 1-5, update the rating of the appointment of that customer.
  + updateCustomerComment: receive an object of type String, which is the ID of the customer and a String which is the comment made by that customer, update it to the appointment.
* **SupabaseAppointmentApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

### **4.2.9 Statistics**



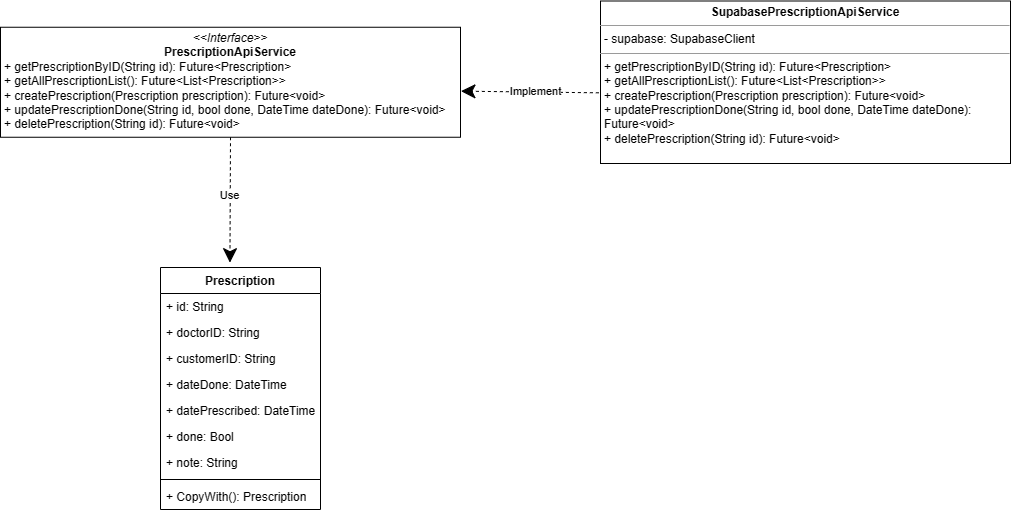
* **Statistics**: this is the data class as mentioned below
* **StatisticsApiService:** this is an interface contains methods for the class Statistics to work with the database:
  + **createStatistics:** Adds a new statistics entry to the database.
  + **deleteStatistics:** Deletes the statistics entry identified by the provided 'id'.
  + **getStatistics:** Retrieves the statistics entry identified by the provided 'id'.
  + **getAllStatisticsList:** Returns a list containing all statistics entries from the database.
  + **getStatisticsList:** Retrieves a list of statistics entries associated with the provided 'prescriptionID'.
  + **updateStatistics:** Updates the statistics entry identified by the provided 'id' with the new 'statistics' data.
  + **updateStatisticsValue:** Updates the 'value' field of the statistics entry identified by the provided 'id'.
  + **updateStatisticsCategoryName:** Updates the 'categoryName' field of the statistics entry identified by the provided 'id'.
  + **updateStatisticsPrescriptionID:** Updates the 'prescriptionID' field of the statistics entry identified by the provided 'id'.
  + **updateStatisticsNote:** Updates the 'note' field of the statistics entry identified by the provided 'id'.
  + **streamStatistics:** Returns a stream providing real-time updates for the statistics entry identified by 'id'.
* **SupabaseStatisticsApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

### **4.2.10 Absent Request**



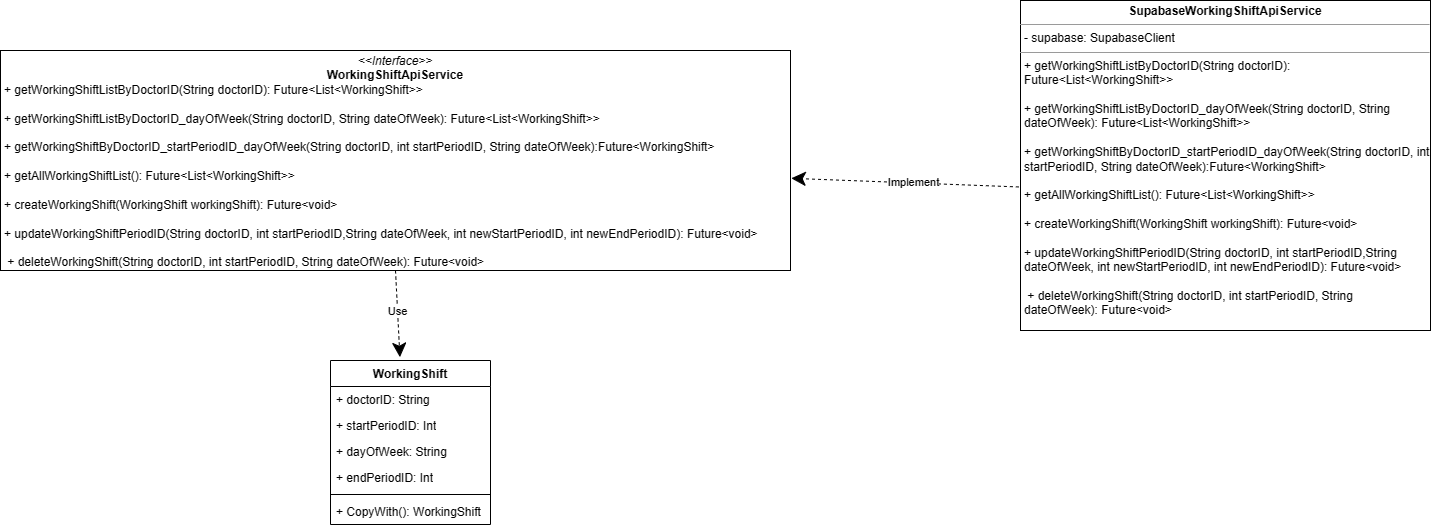
* **AbsentRequest**: this is the data class as mentioned below
* **AbsentRequestApiService:** this is an interface contains methods for the class AbsentRequest to work with the database:
  + **createAbsentRequest**: receive an object of type AbsentRequest and add it to the database
  + **deleteAbsentRequest**: Receive doctorID and date of the request, then delete the appropriate AbsentRequest.
  + **getAbsentRequest**: Receive doctorID and date of the request, and return the appropriate AbsentRequest
  + **getAllAbsentRequestList**: Return the list of all Prescription from the database
  + **getAbsentRequestListByDoctorID**: Retrieves a list of AbsentRequest objects filtered by the provided doctorID.
  + **getAbsentRequestListByDate**: Retrieves a list of AbsentRequest objects filtered by the provided date.
  + **updateAbsentRequest**: Receives a doctorID, date, and an AbsentRequest object, updating the respective AbsentRequest entry in the database with the provided details.
  + **updateAbsentRequestDoctorName**: Updates the doctorName field of the AbsentRequest entry identified by doctorID and date in the database with the provided doctorName.
  + **updateAbsentRequestDateRequest**: Updates the dateRequest field of the AbsentRequest entry identified by doctorID and date in the database with the provided dateRequest.
  + **updateAbsentRequestReason**: Updates the reason field of the AbsentRequest entry identified by doctorID and date in the database with the provided reason.
  + **updateAbsentRequestIsApproved**: Updates the isApproved field of the AbsentRequest entry identified by doctorID and date in the database with the provided isApproved value.
  + **streamAbsentRequest**: Returns a stream that provides updates or changes to the AbsentRequest identified by doctorID and date in real-time from the database.
* **SupabaseAbsentRequestApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

### **4.2.11 Prescription**



* **Prescription**: this is the data class as mentioned below
* **PrescriptionApiService:** this is an interface contains methods for the class Prescription to work with the database:
  + getPrescriptionByID: receive Prescription id of type String, and return a list of all Prescriptions.
  + getAllPrescriptionList: receive the list of all Prescription from the database
  + createPrescription: receive an object of type Prescription and add it to the database
  + updatePrescriptionDone: receive id of type string, done of type bool and dateDone of type DateTime and update that Prescription Done state corresponding to name
  + deletePrescription: receive Prescription id of type String and delete Prescription having that id.
* **SupabaseCategoryApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

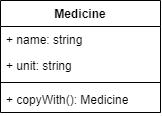
### **4.2.12 Working Shift**



* **WorkingShift**: this is the data class as mentioned below
* **WorkingShiftApiService:** this is an interface contains methods for the class WorkingShift to work with the database:
  + getWorkingShiftListByDoctorID: receive Prescription id of type String, and return a list of all Prescriptions.
  + getWorkingShiftListByDoctorID\_dayOfWeek: receive doctor id type string and get all working shift list of that doctor
  + getWorkingShiftByDoctorID\_startPeriodID\_dayOfWeek: receive doctor id type string, start of period of type int, date of week type string and return that working shift record.
  + getAllWorkingShiftList: receive the list of all Working Shift from the database
  + createWorkingShift: receive an object of type Working Shift and add it to the database
  + updateWorkingShiftPeriodID: receive doctor id of type string, start of period of type int, date of week type string and newStartPeriodID and newEndPeriodID of type int and update that record having first three attributes with newStartPeriodID and newEndPeriodID
  + deleteWorkingShift: receive Working Shift doctor id of type String, start of period of type int, date of week type string and delete Working Shift having those data.
* **SupabaseCategoryApiService:** this class will implement the mentioned services above using the functions provided by Supabase.

## Package: Models

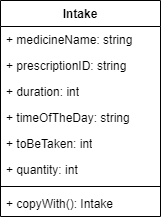
### **4.3.1 Medicine**



**Medicine:** This class contains information about every medicine shown in the application. Every

medicine has a name and a unit to measure the doses. The copyWith() method is for cloning.

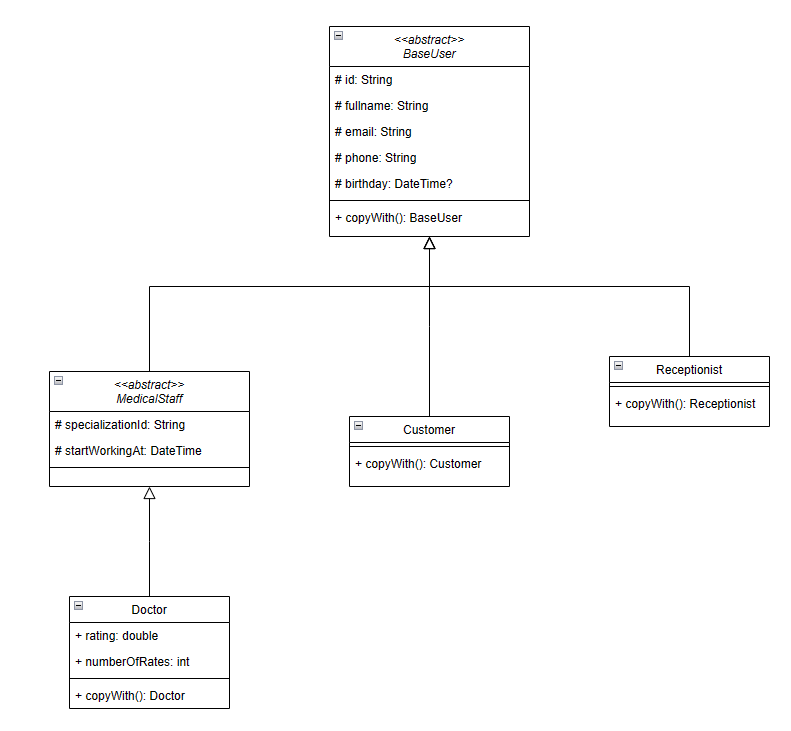
### **4.3.2 Intake**



**Intake:** The class model contains information about a patient intake. Every intake has to have a medicine   
 name and the ID of the prescription. The optional fields are the duration, the time of the day to take the

medicine, and the amount to be taken.

**4.3.3 User**



**BaseUser:** this abstract class contains all the essential information of a user, and:

* copyMethod:
  + Type: Function
  + Arguments: all of the attributes of the current class
  + Return type: either BaseUser or subclass of BaseUser

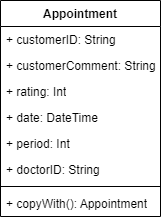
**MedicalStaff**: This class inherits **BaseUser**, all the medical staff including doctors, nurses must subclass it to create a concrete class. It contains additional id for querying the specialization information and the start working date of the staff.

**Doctor:** This is the data class for Doctor in our application.

**Customer:** This is the data class for Customer(patient) in our application.

**Receptionist:** This is the data class for Receptionist in our application.

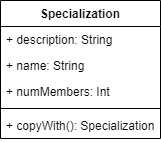
### **4.3.4 Appointment**



**Appointment:** this is the data class for storing the appointment details of each patient. The copyWith()

works as a way to modify the attributes in the object Appointment.

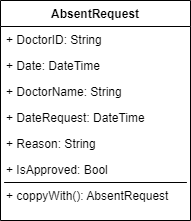
### **4.3.4 Specialization**



**Specialization:** this is the data class for the specialization of all the doctors. The copyWith()

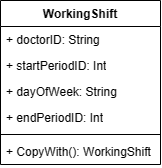
works as a way to modify the attributes in the object Specialization.

### **4.3.5 Absent Request**



**AbsentRequest:** this is the data class for storing the Absent request of doctors.

### **4.3.6 Working Shift**



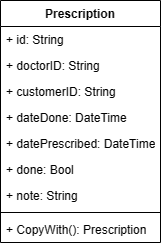
**WorkingShift:** this is the data class for the WorkingShift of all the doctors including doctor ID, start Period ID, the day of week (monday, tuesday, …) and end Period ID describing the doctor schedule.

### **4.3.7 Category**



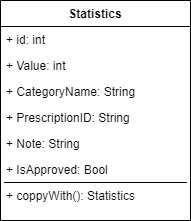
**Category:** this is the data class for the Category of all the scalable health characteristics include name and description for each type.

### **4.3.8 Prescription**



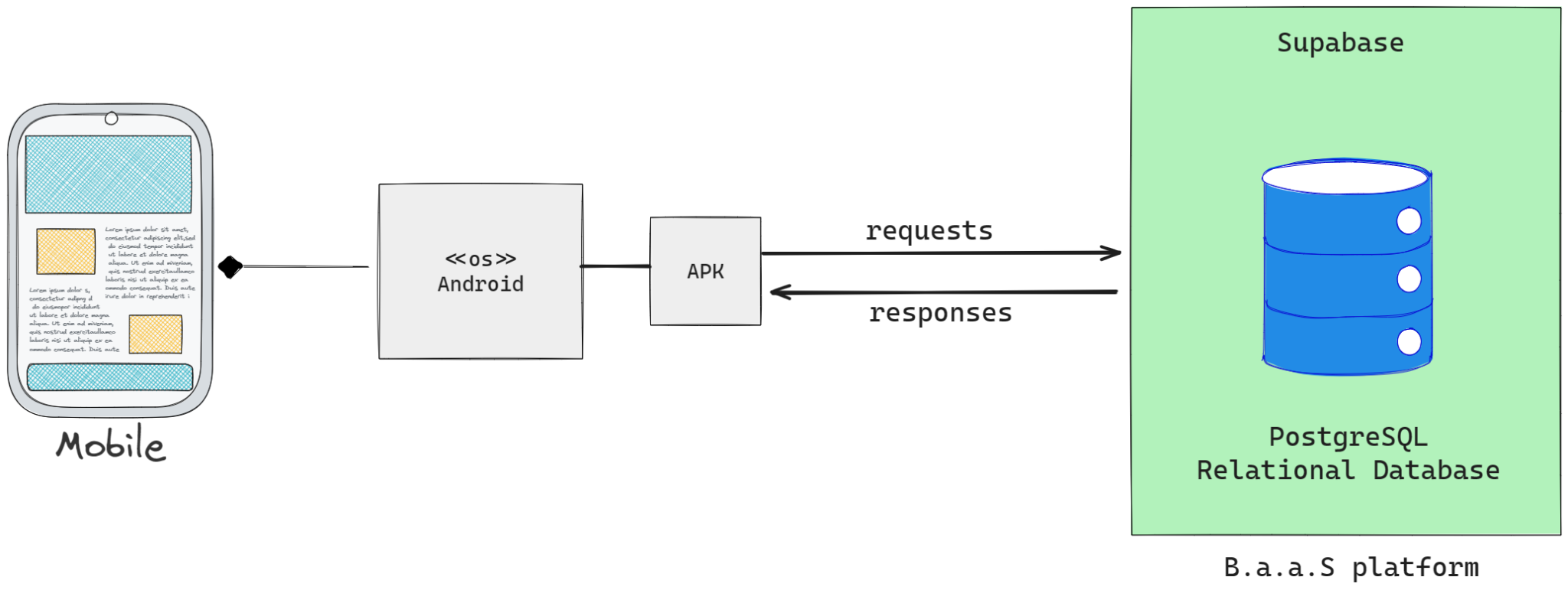
**Prescription:** this is the data class for the Prescription of Customer(Patients) gained from Doctor. This table would include ID of prescription, Doctor ID, customer ID, the date done for totally expired with prescription, and date prescribed marked as done for prescribing, done and note for extra information

### **4.3.9 Statistic**



**Statistics:** this is the data class for storing the medical statistics value such as heart rate, blood pressure, etc.. .

# Deployment

**

# Implementation View

The project folder structure is organized as below with src being the root folder:

src/

—-------packages/

--- **models/**

---------user/

—-----------base\_user.dart

--------------doctor/

—-------------------doctor.dart

--------------customer/

—-------------------customer.dart

--------------receptionist/

—-------------------receptionist.dart

---------statistics/

—-------------------statistics.dart

---------absent\_request/

—-------------------absent\_request.dart

---------appointment/

—-------------------appointment.dart

---------category/

—-------------------appointment.dart

---------intake/

—-------------------intake.dart

---------medicine/

—-------------------medicine.dart

---------prescription/

—-------------------prescription.dart

---------specialization/

—-------------------specialization

---------working\_shift/

—-------------------working\_shift.dart

--- **services/**

---------authentication/

—-------------------authentication\_api\_service.dart

—-------------------supabase\_authentication\_api.dart

---------storage/

—-------------------storage\_api\_service.dart

—-------------------storage\_api/

—---------------------------supabase\_storage\_api\_service.dart

—---------------------------firebase\_storage\_api\_service.dart

---------user/

—-----------user\_api\_service.dart

—-----------medical\_staff\_api\_service.dart

--------------doctor/

—-------------------supabase\_doctor\_api\_service.dart

--------------customer/

—-------------------supabase\_doctor\_api\_service.dart

--------------receptionist/

—-------------------supabase\_receptionist\_api\_service.dart

---------statistics/

—-----------statistics\_api\_service.dart

—-----------supabase\_statistics\_api\_service.dart

---------absent\_request/

—-----------absent\_request\_api\_service.dart

—-----------supabase\_absent\_request\_api\_service.dart

---------appointment/

—-----------appointment\_api\_service.dart

—-----------supabase\_appointment\_api\_service.dart

---------category/

—-----------category\_api\_service.dart

—-----------supabase\_category\_api\_service.dart

---------intake/

—-----------intake\_api\_service.dart

—-----------supabase\_intake\_api\_service.dart

---------medicine/

—-----------medicine\_api\_service.dart

—-----------supabase\_medicine\_api\_service.dart

---------prescription/

—-----------prescription\_api\_service.dart

—-----------supabase\_prescription\_api\_service.dart

---------specialization/

—-----------specialization\_api\_service.dart

—-----------supabase\_specialization\_api\_service.dart

---------working\_shift/

—-----------working\_shift\_api\_service.dart

—-----------supabase\_working\_shift\_api\_service.dart

--- **controllers/**

----------authentication/

--------------auth\_repository\_service.dart

--------------auth\_repository/

—-------------------supabase\_auth\_repository.dart

----------customer/

--------------customer\_repository\_service.dart

--------------customer\_repository/

—-------------------supabase\_customer\_repository.dart

----------receptionist/

--------------receptionist\_repository\_service.dart

--------------receptionist\_repository/

—-------------------supabase\_receptionist\_repository.dart

----------doctors/

--------------doctor\_repository\_service.dart

--------------doctor\_repository/

—-------------------supabase\_doctor\_repository.dart

----------admin/

--------------admin\_repository\_service.dart

--------------admin\_repository/

—-------------------supabase\_admin\_repository.dart

----------storage/

--------------storage\_repository\_service.dart

--------------storage\_repository/

—-------------------supabase\_storage\_repository.dart

--- **views/**

----------widgets/

----------screens/

—-----------users/

—-----------doctors/

—-----------receptionists/

—-----------admin/