Senior Design Project

SurePa

Low-Level Design Report

Kaan Ateşel, İsmet Alp Eren, Kerem Alemdar, Eylül Çağlar, Emine Ezgi Saygılı

Supervisor: Prof. Selim Aksoy

Jury Members: Dr. Shervin Arashloo and Dr. Hamdi Dibeklioglu

Contents

1. Introduction

Forgetfulness is a crucial problem when it starts to occur often in daily lives. Forgetting something unnecessary is good for your health and helps you learn and remember new and vital things. However, forgetting things like taking medicines, drinking water, walking, and any other daily routines complicates and adversely affects the lives of those particular people.

Forgetfulness targets elders primarily. Occasional lapses in short-term memory are a normal part of the aging process, such as forgetting which medicine to take and their daily doses. So, many older adults need new ways to remember to take their medications and know which medicine to take.

People with certain diseases, for instance, tumors, blood clots, and thyroid, also suffer from forgetting in their daily lives. While having a disease, daily routines like not forgetting to take medicines, drinking water becomes more critical than usual. In the search for finding a solution to these problems, we came up with SurePa. It is an application for mainly offering solutions to not taking medicine problems resulting from forgetfulness targeting primarily older adults, diabetic people, and other people. In addition, different from other applications, SurePa aims to provide their users with some image-recognition-based solutions to their problems. Such solutions aim to ease the life of these patients by providing intuitive ways for remembering and tracking their medication use. SurePa consists of convenient tools for anyone who carries a mobile phone to help its users undergo a smooth recovery process.
In this report, a high-level design of the SurePa application will be provided. This high-level design firstly includes the purpose, design goals, definitions, and overview of the system. Then the current software architecture will be included. Afterward, proposed software architecture, subsystem services, consideration of various factors in engineering design will be discussed. Lastly, teamwork details will be provided in this high-level design report.

1.1. Object design trade-offs

Usability vs. Functionality: The main goals of SurePa are to provide an easy-to-understand and easy-to-use application, especially to elderly patients. Excessive functionality can not always bring a good result, but it can confuse the elderly which will result in either loss of connection between the user and the application due to the confusion or mistakes in the usage of the application which could result in health demerits in a health-related application such as SurePa. Therefore, our design tends to prefer usability over functionality.

Rapid Development vs. Robustness: Project is planned to finish within 2 semesters, therefore, we have limited time. At this time our main focus will be to develop a functionally working application. At the end of this short period, the application could need further optimizations and bug fixes.

Scalability vs. Efficiency: Since SurePa's targeted users are those who take medicines and this means a lot of people, the server and database should be able to process a large amount of data and can be used concurrently. In this case, efficiency can be in the second place because it is not really important how fast you reach your data or how fast you see your medicine schedule since once they've fetched the information could be stored locally and the medicine schedule does not change frequently. Thus, SurePa needs to be scalable over being efficient.

1.2. Interface documentation guidelines

A sample class is described as follows:

Class Name: Description of the class

Attributes: attributeName: type

Methods: methodName(args): return type | Explanation of methods

"Class Name" will be the name of the referred class and "Attributes" will be used consts, states, or lets. The "Methods" section will indicate what are the methods declined in this class and what are their purposes.

1.3. Engineering standards

In this Low-Level Design Report, the UML standards will be used to represent classes. In addition, the IEEE standards will be used for citations for used references.

1.4. Definitions, acronyms, and abbreviations

- **HTTP**: Hypertext Transfer Protocol
- **FB**: Firebase

- **API**: Application Programming Interface
- **UI**: User Interface
- **DB**: Database
- **UML**: Unified Modeling Language
- **IEEE**: Institute of Electrical and Electronics Engineers
- **Caregiver:** A type of user whose account is connected with the patient’s account.
- **CG:** Caregiver
- **Medication Schedule:** A schedule that shows which pill will be taken and when it will be taken.
- **Image Recognition:** Computer applications that can identify objects from an image or video. It is a context of computer vision.

2. Packages

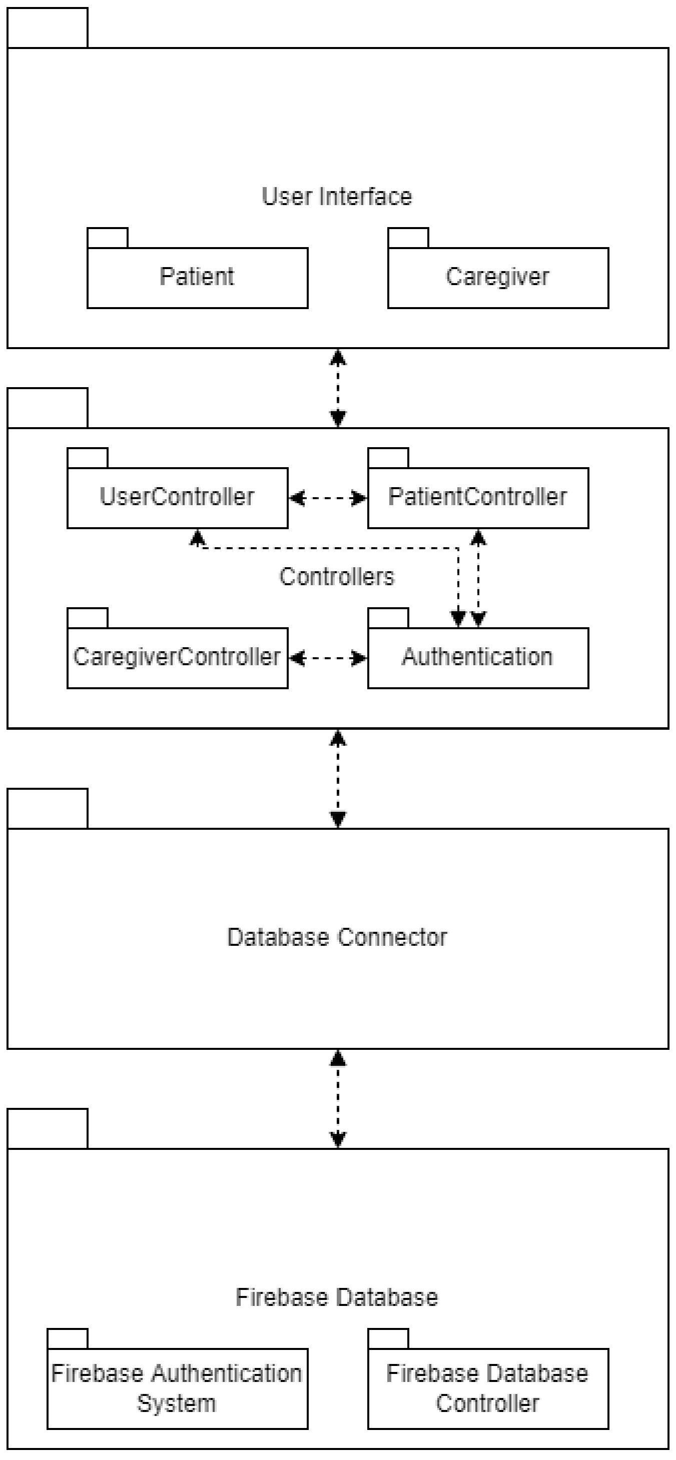

Figure 1: Packages

User Interface:

- Patient: This subpackage refers to the user interface of ‘patient’ type users.
- Caregiver: This subpackage refers to the user interface of caregiver type users.

Controllers:

- UserController: This subpackage contains all user-related firebase controllers.
- PatientController: This subpackage contains all patient-related firebase controllers.
- CaregiverController: This subpackage contains all caregiver-related firebase controllers.
- Authentication: This subpackage contains all authentication-related firebase controllers.

Database Connector: This package contains firebase connection methods.

Firebase Database:

- Firebase Authentication System: This subpackage contains firebase's own methods.
- Firebase Database Controller: This subpackage contains the firebase's own methods.

3. Class Interfaces

3.1. Controller

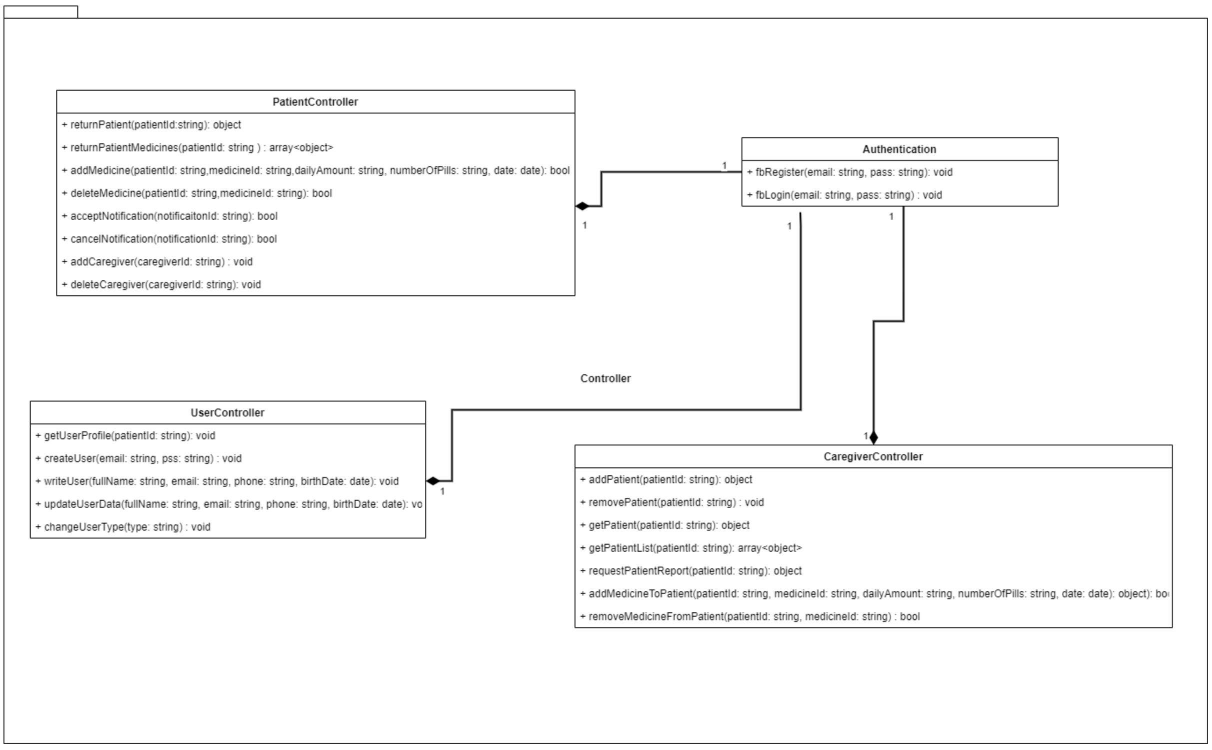
PatientController
+ returnPatient(patientId:string): object
+ returnPatientMedicines(patientId: string) : array<object>
+ addMedicine(patientId: string,medicineId: string,dailyAmount: string, numberOfPills: string, date: date): bool
+ deleteMedicine(patientId: string,medicineId: string): bool
+ acceptNotification(notificaitonId: string): bool
+ cancelNotification(notificationId: string): bool
+ addCaregiver(caregiverId: string) : void
+ deleteCaregiver(caregiverId: string): void
Authentication
+ fbRegister(email: string, pass: string): void
+ fbLogin(email: string, pass: string) : void
1
1
1
1
Controller
UserController
+ getUserProfile(patientId: string): void
+ createUser(email: string, pss: string) : void
+ writeUser(fullName: string, email: string, phone: string, birthDate: date): void
+ updateUserData(fullName: string, email: string, phone: string, birthDate: date): vo
+ changeUserType(type: string) : void
1
1
CaregiverController
+ addPatient(patientId: string): object
+ removePatient(patientId: string) : void
+ getPatient(patientId: string): object
+ getPatientList(patientId: string): array<object>
+ requestPatientReport(patientId: string): object
+ addMedicineToPatient(patientId: string, medicineId: string, dailyAmount: string, numberOfPills: string, date: date): object): bo
+ removeMedicineFromPatient(patientId: string, medicineId: string) : bool

Figure 2: Class Diagram for Controller

Table 1: Patient Controller

PatientController	This class enables us to write and read patients' data from the firebase database.	
Methods	returnPatient(patientId: string): object	It returns every data related to the corresponding patient id.
	returnPatientMedicines(patientId: string) : array<object>	It returns an array of medicine objects related to given patient id.
	addMedicine(patientId: string, medicineId: string, dailyAmount: string, numberOfPills: string, date: date): bool	This method adds medicine to the patient's medicine list.
	deleteMedicine(patientId:string medicineId: string): bool	This method deletes medicine from the patient's medicine list.
	acceptNotification(notificationId: string): bool	When the user clicks, the "I've taken my medicine" button in the notification, accept notification method updates the database accordingly to inform the caregiver.
	cancelNotification(notificationId: string): bool	When the user clicks the "I've not taken my medicine" button in the notification, the cancel notification method updates the database accordingly to inform the caregiver.
	addCaregiver(caregiverId: string)	This method links the current patient with the given caregiver.
	deleteCaregiver(caregiverId: string)	This method deletes the current chosen caregiver from the

		patients’ caregiver list and updates the related caregivers’ account about the deletion.

Table 2: User Controller

UserController	This class enables us to write and read users' data.	
Methods	getUserProfile(patientId: string)	It returns every credential.
	createUser(email: string, pss: string)	It creates a user and registers it to the firebase auth.
	writeUser(fullName: string, email: string, phone: string, birthDate: date)	After the user successfully created, it creates a new document in firebase firestore.
	updateUserData(fullName?: string, email?: string, phone?: string, birthDate?: date)	Updates the user’s credentials.
	changeUserType(type: string)	Changes the type of the user. Caregiver to patient and patient to caregiver.

Table 3: Caregiver Controller

CaregiverController	This class enables us to write and read users’ data	
Methods	addPatient(patient): Object	It writes new patients to the database and returns the newly added patient.
	removePatient(patientId): voidl	It deletes the patient with the chosen id and returns nothing. It also updates the corresponding patients’

		caregiver list.
	getPatient(patientId): Object	It returns the patient with a given Id.
	getPatientList(patientId): Array<object>	It returns the patients list of the corresponding caregiver.
	requestPatientReport(patientId): object	It creates a pdf report with chosen patient data.
	addMedicineToPatient(patientId: string, medicineId: string, dailyAmount: string, numberOfPills: string, date: date): object	It adds new medicine to the patient with a chosen id.
	removeMedicineFromPatient(patientId:string medicineId: string): bool	It removes the medicine from the patient with chosen id.

Table 4: Authentication

Authentication	This class handles login and register functions.	
Methods	fbRegister(email: string, pass : string) : void	It deletes the patient with the given id and returns nothing.
	fbLogin(email: string, pass: string) : void	It returns the patient with chosen id.

3.2. Database Connector

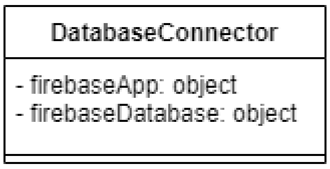

Figure 3: Class Diagram for Database Connector

Table 5: Database Connector

DatabaseConnector	This class makes the corresponding connection between the firebase database and the application.
Attributes	firebaseApp: object firebaseDatabase: object

3.3. User Interface

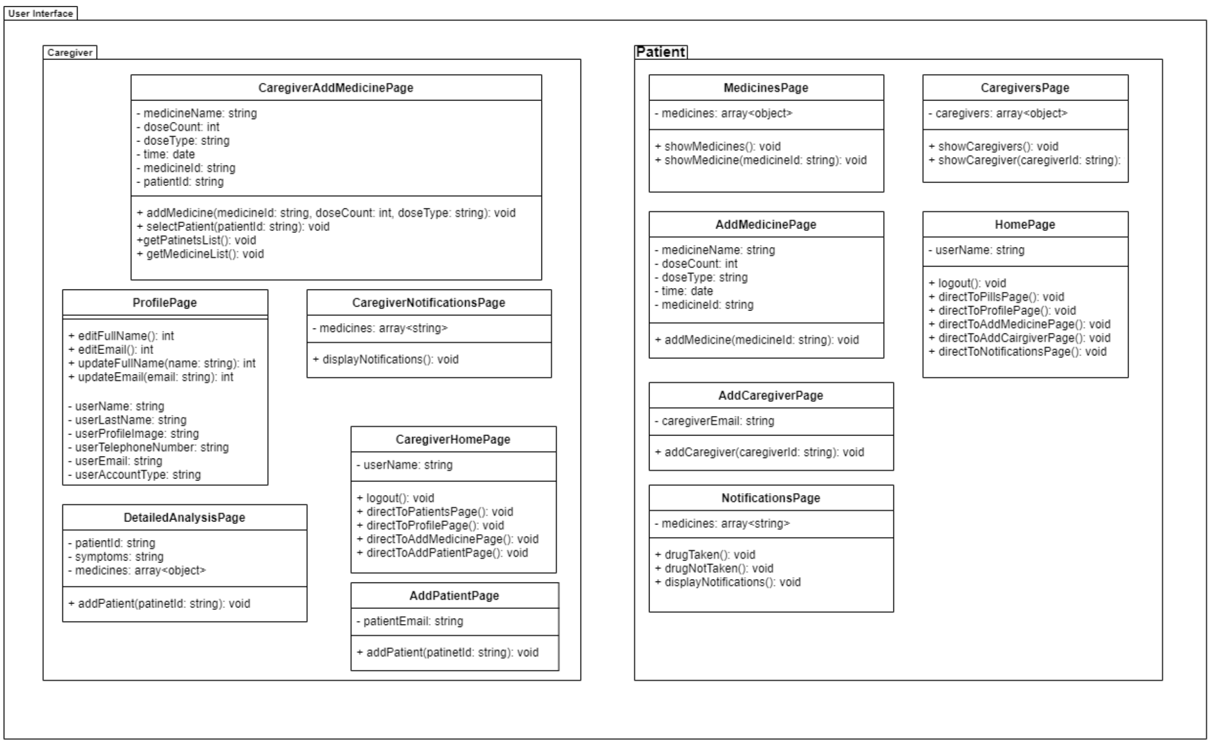
User Interface
Caregiver
CaregiverAddMedicinePage
- medicineName: string
- doseCount: int
- doseType: string
- time: date
- medicineId: string
- patientId: string
+ addMedicine(medicineId: string, doseCount: int, doseType: string): void
+ selectPatient(patientId: string): void
+getPatinetsList(): void
+ getMedicineList(): void
ProfilePage
+ editFullName(): int
+ editEmail(): int
+ updateFullName(name: string): int
+ updateEmail(email: string): int
- userName: string
- userLastName: string
- userProfileImage: string
- userTelephoneNumber: string
- userEmail: string
- userAccountType: string
CaregiverNotificationsPage
- medicines: array<string>
+ displayNotifications(): void
CaregiverHomePage
- userName: string
+ logout(): void
+ directToPatientsPage(): void
+ directToProfilePage(): void
+ directToAddMedicinePage(): void
+ directToAddPatientPage(): void
DetailedAnalysisPage
- patientId: string
- symptoms: string
- medicines: array<object>
+ addPatient(patinetId: string): void
AddPatientPage
- patientEmail: string
+ addPatient(patinetId: string): void
Patient
MedicinesPage
- medicines: array<object>
+ showMedicines(): void
+ showMedicine(medicineId: string): void
CaregiversPage
- caregivers: array<object>
+ showCaregivers(): void
+ showCaregiver(caregiverId: string):
AddMedicinePage
- medicineName: string
- doseCount: int
- doseType: string
- time: date
- medicineId: string
+ addMedicine(medicineId: string): void
HomePage
- userName: string
+ logout(): void
+ directToPillsPage(): void
+ directToProfilePage(): void
+ directToAddMedicinePage(): void
+ directToAddCairgiverPage(): void
+ directToNotificationsPage(): void
AddCaregiverPage
- caregiverEmail: string
+ addCaregiver(caregiverId: string): void
NotificationsPage
- medicines: array<string>
+ drugTaken(): void
+ drugNotTaken(): void
+ displayNotifications(): void

Figure 4: Class Diagram for User Interface

3.3.1. Patient

Table 6: Home Page

HomePage	Displays the home page when a patient-user is logged in.	
Attributes	userName: string	
	logout(): void	It ends the session for the user
	directToPillsPage(): void	Directs to the pill list page.
	directToProfilePage(): void	Directs to profile page.

Methods	directToAddMedicinePage(): void	Directs to add medicine page.
	directToAddCaregiverPage():void	Directs to add caregiver page.
	directToNotificationsPage(): void	Directs to notifications page.

Table 7: Profile Page

ProfilePage	Displays the profile of the current user or selected patient	
Attributes	userName: string userLastName: string userProfileImage: string (URL) userTelephoneNumber: string userEmail: string userAccountType: string	
Methods	editFullName(): int	This method is called when the user clicks the edit button to open the edit full name popup.
	editEmail(): int	This method is called when the user clicks the edit button to open the edit email popup.
	updateFullName(name: String): int	In the edit full name popup if the user clicks the update button this function is called to update the user name.
	updateEmail(email: String): int	In the edit email popup if the user clicks the update button this function is called to update the user email.

Table 8: Caregivers Page

CaregiversPage	Displays the caregiver's page where the user could see all of their caregivers.	
Attributes	caregivers: array<object>	This array will store caregiver objects which include name, and type such as neighbor,

		child, nurse, etc..
Methods	showCaregivers(): void	This method is used to display all caregivers.
	showCaregiver(caregiverId: string): void	This method is used to display the specified caregiver of a patient.

Table 9: Medicines Page

MedicinesPage	Display the medicines page to list all medicines of the corresponding patient.	
Attributes	medicines: array<object>	This array will store medicines objects which include the name and dose of the medicine such as tablet/ampul/gram/capsule/mg/ml.
Methods	displayMedicines(): void	This method is used to display all medicines of a patient.
	displayMedicine(medicineId: string): void	This method is used to display the specified medicine of a patient.

Table 10: Add Medicine Page

AddMedicinePage	Displays the add medicine page which will be used to add new medicine to the schedule.	
Attributes	medicineName: string doseCount: int doseType: string time: date medicineId: string	
Methods	addMedicine(medicineId: string, doseCount: int,	This method is used to add medicine to the database to the user's schedule.

	doseType: string): void	

Table 11: Add Caregiver Page

AddCaregiverPage	Displays the add caregiver page which will be used to add a new caregiver to the patient.	
Attributes	caregiverEmail: string	
Methods	addCaregiver(caregiverId : string): void	This method will send an invitation to the CG if the email exists in the database.

Table 12: Notifications Page

NotificationsPage	Displays the notifications of the medicines.	
Attributes	medicines: array<string>	
Methods	drugTaken(): void	This method will notify the patient’s caregiver that the patient has taken the drug.
	drugNotTaken(): void	This method will notify the patient’s caregiver that the patient has not yet taken the drug.
	displayNotifications(): void	This method displays all notifications.

3.3.2. Caregiver

Table 13: Caregiver Home Page

CaregiverHomePage	Displays the home page when a caregiver user is logged in.

Attributes	userName: string	
Methods	logout(): void	It ends the session for the user.
	directToPatientsPage(): void	Directs to the patient list page.
	directToProfilePage(): void	Directs to the profile page.
	directToAddMedicinePage(): void	Directs to the add medicine page(caregiver).
	directToAddPatientPage(): void	Directs to add a patient page.

Table 14: Patients Page

PatientsPage	Displays the patients page where the user could see all of their patients.	
Attributes	patients: array<object>	This array will store the patient's objects which include names.
Methods	showPatients(): void	This method is used to display all patients.
	directToDetailedInformationPage(patientId: string): void	This method is used to direct to the detailed analysis page of a certain patient.

Table 15: Caregiver Add Medicine Page

CaregiverAddMedicinePage	This page lets the caregiver add medicine for the selected patient.
Attributes	medicineName:string time: date doseCount: int doseType: string

	medicineId: string patientId: string	
Methods	addMedicine(medicineId: string, doseCount: int, doseType: string): void	This method is used to add medicine to the patient’s schedule.
	selectPatient(patinetId: string): void	This method is used to select a patient to add medicine for.
	getPatientsList(): void	This method is used to get the patients of the current caregiver.
	getMedicineList(): void	This method is used to get the medicines list.

Table 16: Add Patient Page

AddPatientPage	This page lets the caregiver add new patients.	
Attributes	patientEmail: string	
Methods	addPatient(patinetId: string): void	This method is used to assign a new patient to the caregiver.

Table 17: Detailed Analysis Page

DetailedAnalysisPage	Displays the detailed analysis page of a certain patient which includes their medicine taking an on-time percentage and their medical information/ health records.	
Attributes	patientId: string symptoms: string medicines: array<object>	
Methods	displayPatientDetails(): void	This method is used to display patients' details.

Table 18: Caregiver Notifications Page

CaregiverNotificationsPage	Displays the notifications of the medicines.	
Attributes	medicines: array<string>	
Methods	displayNotifications(): void	This method displays all notifications.

4. Glossary

- FB: Firebase cloud services by Google including storage [1].
- React-Native: React-Native is a TypeScript-based open-source web application framework developed by Facebook engineers.

5. References

[1] “Firebase & Google Cloud,” *Google*. [Online]. Available: https://firebase.google.com/firebase-and-gcp. [Accessed: 28-Feb-2022].