Bilkent University

Department of Computer Engineering

Senior Design Project

HelpingHands

Low-Level Design Report

Mustafa Anıl Taşdan, Metehan Gürbüz, Rola Younis, Hassam Abdullah

SUPERVISOR: Dr. İbrahim Körpeoğlu

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

Feb 28, 2022

1. Introduction

As mankind progressed through the ages, the technology we possessed and our understanding of science progressed alongside us. Medicine was one of the fields that saw the most progression. Many medical conditions that may have resulted in fatalities in the past have simple solutions to them today. One big development that proved crucial to the field of medicine were blood transfusions. The first human-to-human blood transfusion was performed in 1795.[1] Since then, blood transfusions have saved countless lives, but the amount of daily blood donations is still very little. Blood transfusions are used for many medical conditions, such as anemia or cancer, but more importantly they are crucial for blood loss resulting from accidents.[2] As donated blood can only be stored for 42 days, and with only a few percent of the population donating their blood[3], a steady supply of blood donations is needed every day. This can be further proven by the fact that Bilkent University periodically sends requests for emergency blood donations. If our community consisting of 14.000 people needs to send this many requests, it is easy to grasp the grimness of the situation when we enlarge the scale to the whole population of Turkey.

This is where we, as HelpingHands, want to come in. We want to provide a mobile based solution for connecting recipients and donors. We want to provide a platform where people in need, or hospitals, can put requests for types of blood donations, and donors fitting the provided criteria can be alerted by notifications, so that they can contact the recipients easily.

HelpingHands is a mobile application that aims to provide a platform for connecting blood donors and recipients. This is a difficult task because there is not a streamlined platform for a task like this and people use all sorts of vectors, such as e-mails, social media or emergency announcements. While HelpingHands does not aim to completely replace these tools, it aims to provide a high quality platform so these tools are made redundant for this task.

The application is a mobile based application, with two parts, client and server. For the mobile client, we will use React Native, which eliminates the need for using two different native languages for Android and IOS development. For the server, we will use a Node.js framework called Express.js. We will pair it with a MongoDB No-SQL database and finally deploy it to AWS.

1.1 Object Design Trade-offs

1.1.1 Functionality vs. Usability

Usability is more important to HelpingHands than functionality. To keep the app as basic as possible, we minimize needless and sophisticated UI functionalities. The UI is designed to stick to the application's main features in order to provide a positive user experience.

1.1.2 Security vs. Cost

Users' personal information, such as phone numbers and email addresses, will be collected and stored by HelpingHands. We must ensure that this information is kept safe at all times. The importance of data security will outweigh the cost. We use encrypted databases for security.

1.1.3 Compatibility vs Extensibility

HelpingHand is designed as cross-platform mobile software. It will have both android and Ios variants since it is made using react native. This brings some minor nuances in the deploying of our application to both platforms. But we are using React native for both the platforms, regardless.Using the same language and development tool for the mobile interface will provide flexibility for our system. Updates and the change for new features will be reflected easily. In addition, our system design is based on packages and those packages are divided into controllers and working modules. This modular design enables us to extend our system easily.

1.2 Interface Documentation Guidelines

In the documentations, UML design principles are used in the description of class interfaces, diagrams, scenarios and use cases, subsystem compositions, and hardware-software components depiction. The purpose of UML is to provide a standard notation that can be used by all object-oriented methods and to select and integrate the best elements of precursor notations. UML has been designed for a broad range of applications. In conclusion, it offers constructs for a wide range of systems and activities (distributed systems, analysis, system design and deployment).

Class	Name of the Class
Description	Description of the Class
Attributes	
Name of an attribute	
Methods	
Signature of method	Description of the method

1.3 Engineering Standards

Throughout this report, as well as in our previous ones, we have used the Unified Modeling Language Standard (UML) [3] for the diagrams depicting the details of the system. Also, IEEE (Institute of Electrical and Electronics Engineers) [4] citation style has been used in our references.

1.4 Definitions, Acronyms, and Abbreviations

JS- Javascript

UI- User Interface

UML- Unified Modeling Language

Http- Hypertext Transfer Protocol

DAO- Direct Access Object

API- Application Programming Interface

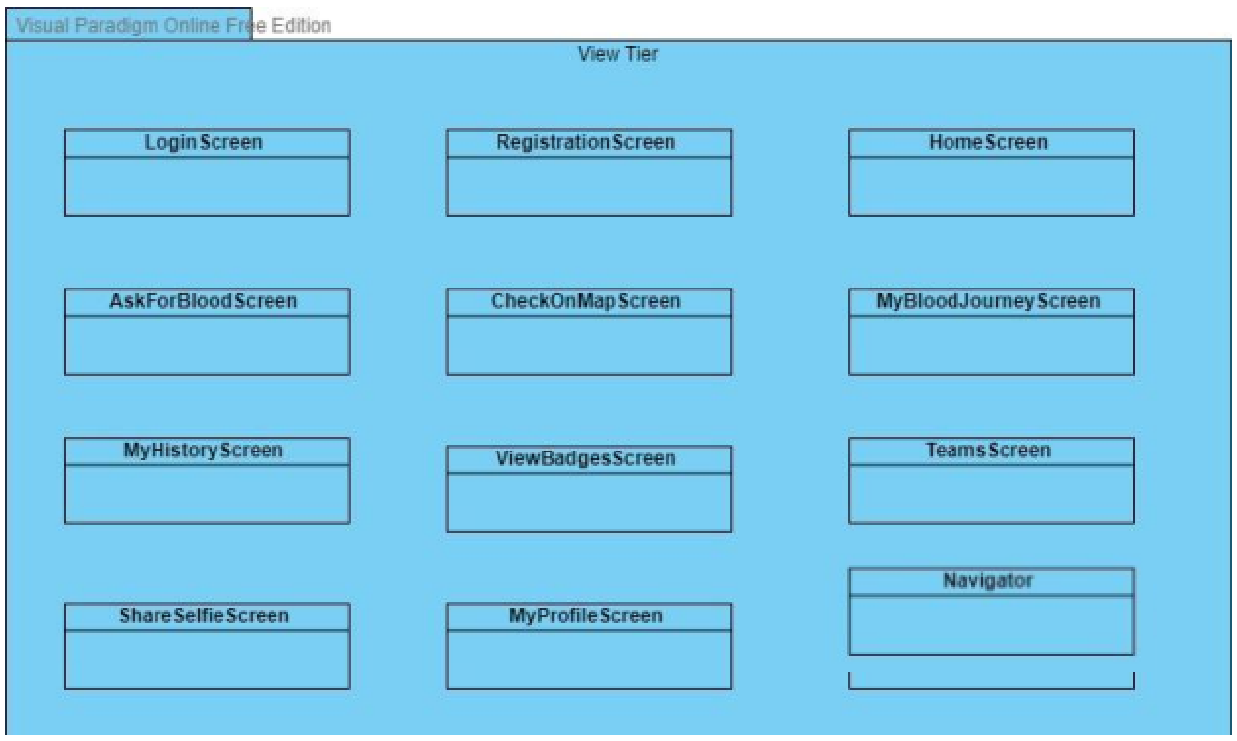
Node.js- open source platform, cross-platform, back-end JavaScript runtime environment

Express.js- back end application framework for Node.js

2. Packages

2.1 Client

2.1.1 View



View Package: The classes in the View package will be responsible for containing React Native UI components. The data shown to the user will be stored in Model package classes. The View package classes will notify the Controller package classes upon interaction with the user.

LoginScreen: Holds the UI elements for the Login screen. Will contain fields for username and password entry, along with buttons for login and registry. Upon successful login the screen will change to HomeScreen, upon registration request the screen will change to RegistrationScreen.

RegistrationScreen: Will hold UI elements for the Registration screen. Will contain a form for the user to fill with their personal information. Will change to HomeScreen upon successful registration.

HomeScreen: The dashboard that will welcome the user upon successful login. Will display user information like current status and statistics, and will contain buttons/menus to facilitate switching to other screens.

AskForBloodScreen: Will contain a form for the app to broadcast a blood request to every user that fills the criteria, that being fitting the requested blood type and being in a close enough radius.

CheckOnMapScreen: This screen is only available upon selecting a blood request shown to the user. It will show the hospital/blood bank that the request originated from and it will show the current location of the user.

MyBloodJourneyScreen: This screen shows the user their past donations in a well presented and interactive way.

MyHistoryScreen: This screen shows the user their past actions in a formatted and listed way.

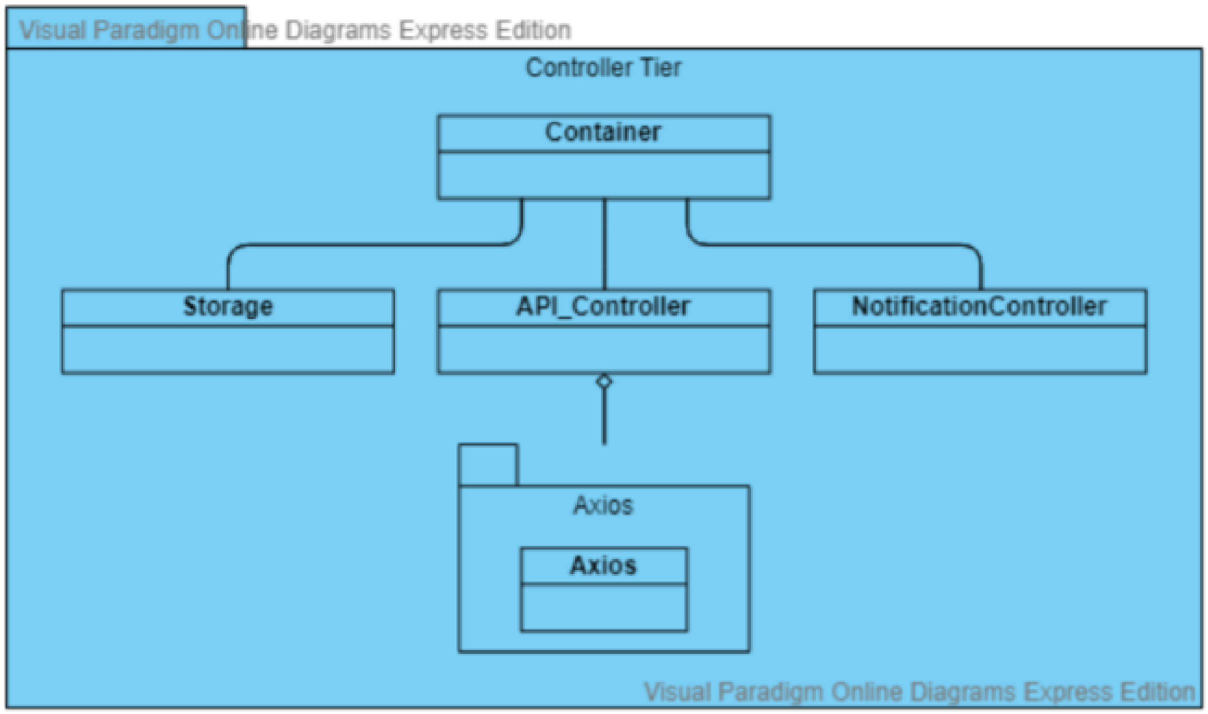
ViewBadgesScreen: Will contain the achievements the user has obtained, along with a leaderboard.

ShareSelfieScreen: Will launch the device’s camera for the user to take a picture. This picture will be utilized to confirm a successful donation.

MyProfileScreen: Will display the user’s personal information, along with options to change the said information and manipulate account data.

Navigator: Will contain React Navigation framework’s components to change screens upon commands from the Controller package.

2.1.2 Controller


Visual Paradigm Online Diagrams Express Edition
Controller Tier
Container
Storage
API_Controller
NotificationController
Axios
Axios
Visual Paradigm Online Diagrams Express Edition

Controller Package: The classes of this package will facilitate communication between the client and the server, along with receiving user commands from the View Package and updating the Model Package upon processing.

Container: This class provides an interface between View and Model Packages for the Controller Package. The View classes will interact with this class, and the Controller Package will update the Model Package via this class.

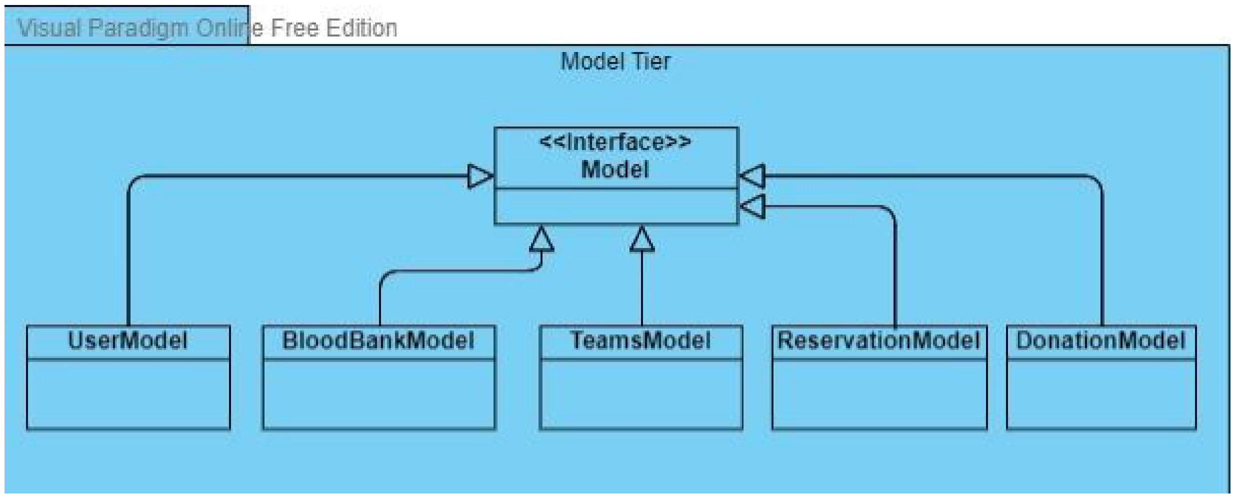
Storage: This class will handle data stored in the device's mass storage. It will be updated upon the Container class' directives.

API_Controller: This class will handle communications with the Axios third party package.

NotificationController: This class will handle the logic of generating notifications via data received from the server.

Axios: Axios is a Node.js framework that provides promise-based HTTP queries and handles responses.

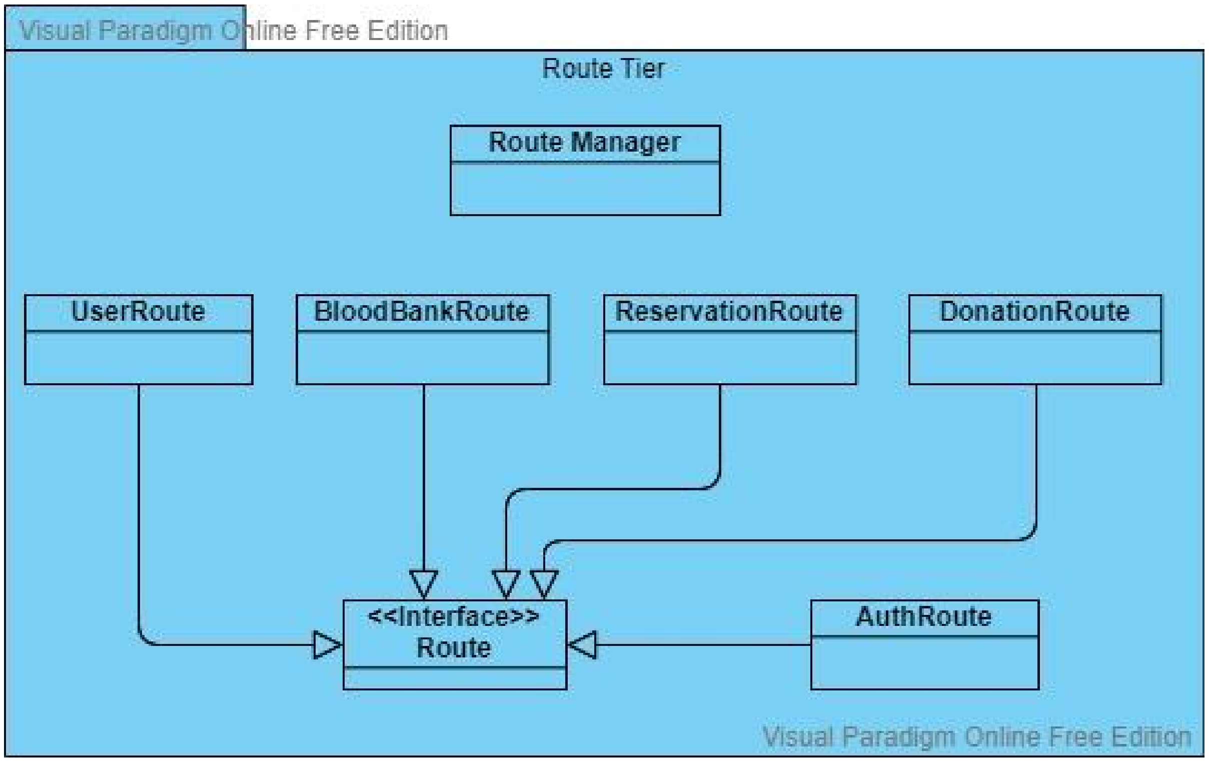
2.1.3 Model



Model Package: Model Package consists of data holding classes for our Model-View-Controller design pattern. The Controller Package alters the contents of the Model Package, while the View Package is notified upon any change to the Model Package. The classes hold necessary variables and data for the View Package, and it is redundant to provide the same explanation for each one.

2.2 Server

2.2.1 Route



Route: This class manages http requests which are post, get, delete, and put.

UserRoute: This class manages http requests to the path “/user/*”

BloodBankRoute: This class manages http requests to the path “/bloodBank/*”

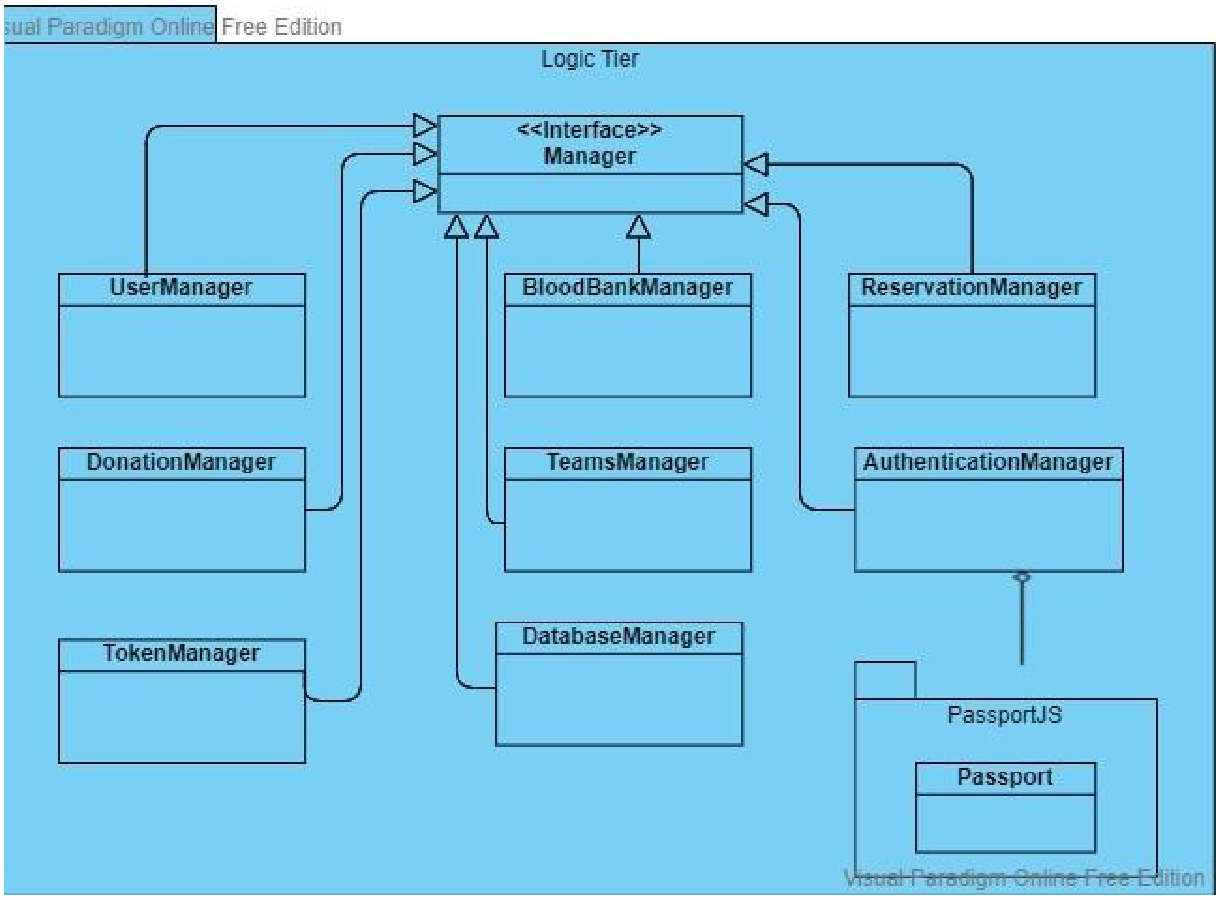
ReservationRoute: This class manages http requests to the path “/reservation/*”`

DonationRoute: This class manages http requests to the path “/donation/*”

AuthRoute: This class manages http requests to the path “/auth/*”

RoutManager: This class provides the server management of the routes.

2.2.2 Logic



Manager: This interface is responsible for the dataflow with the DatabaseManager class.

UserManager: This class provides the business logic behind the user operations.

BloodBankManager: This class provides the business logic behind the blood bank operations.

ReservationManager: This class provides the business logic behind the reservation operations.

DonationManager: This class provides the business logic behind the donation operations.

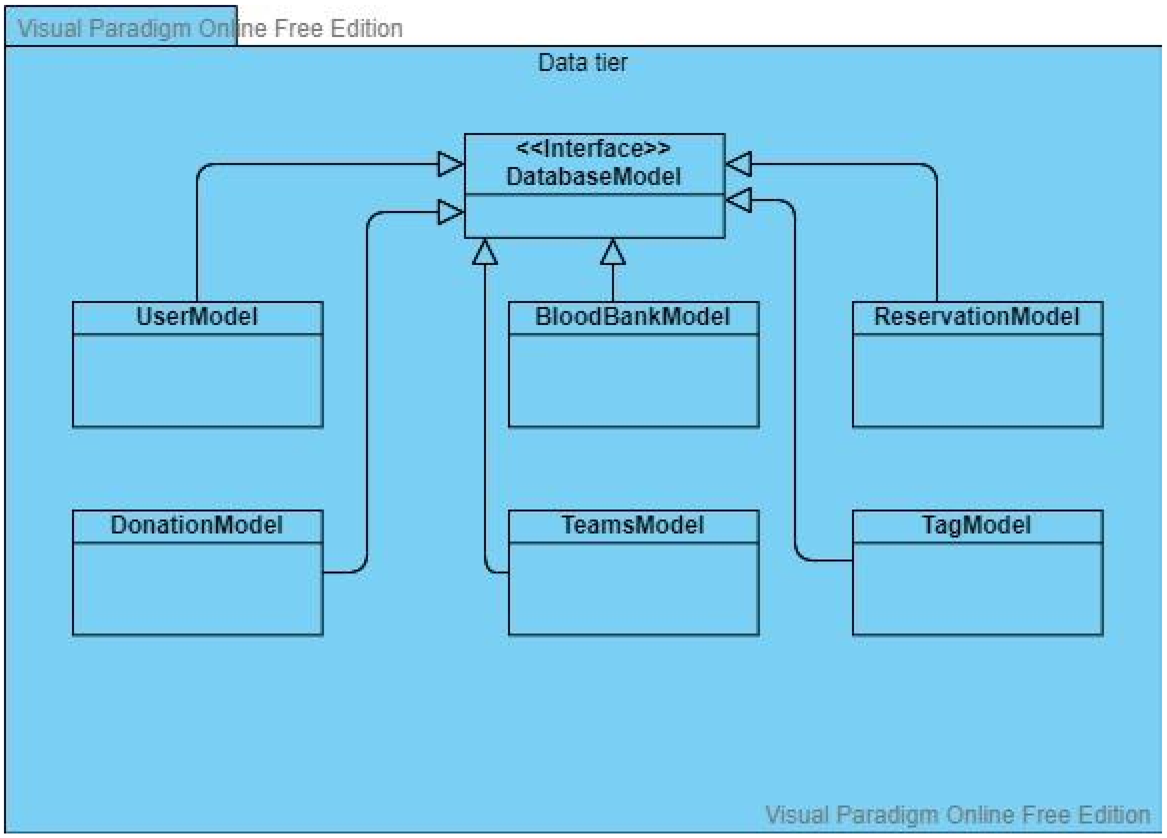
TeamsManager: This class provides the business logic behind the teams' operations.

AuthenticationManager: This class provides the business logic behind authentications operations.

DatabaseManager: This class is in control of the SQL database.

Passport: This class is given by the third-party package PassportJS and is used by Express to construct middlewares and authentication protocols for the web service.

2.2.3 Data



DatabaseModel: This interface is responsible for the get and set methods.

UserModel: This class provides a model for the User system in the database.

BloodBankModel: This class provides a model for the BloodBank system in the database.

ReservationModel: This class provides a model for the reservation system in the database.

DonationModel: This class provides a model for the donation system in the database.

TeamsModel: This class provides a model for the teams system in the database.

TagModel: This class provides a model of the Tag entity in the database.

3. Class Interfaces

3.1 Client

3.1.1 View

class LoginScreen
Holds UI of the login screen.
Attributes
React Native visual components
Methods
Boolean login(*username,password*): Calls login function in Controller Package interface. Navigates to the home screen on success. Boolean signup(): Navigates to registration screen.

class RegistrationScreen
Holds UI of the registration screen.
Attributes
React Native visual components
Methods
Boolean signup(*username,password,...*): Calls signup function in Controller Package interface. Navigates to the home screen on success.

class HomeScreen
Holds UI of the home screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. Boolean acceptRequest(): Calls acceptRequest function in the Controller Package.

class AskForBloodScreen
Holds UI of the blood request screen.
Attributes
React Native visual components
Methods
Boolean bloodRequest(*...*): Calls bloodRequest function in Controller Package interface. Boolean navigate(*Screen*): Navigates to the specified screen. getNearbyBanks(): Gets nearby bank and hospital data from Model Package.

class CheckOnMapScreen
Holds UI of the map screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. getMapData(): Gets current position and nearby requests from the Model Package.

class MyBloodJourneyScreen
Holds UI of the blood journey screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. getHistory(): Gets user donation and request history from Model Package.

class MyHistoryScreen
Holds UI of the history screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. getHistory(): Gets user donation and request history from Model Package.

class ViewBadgesScreen
Holds UI of the badges screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. getBadges() : Gets user badge data from Model Package. getLeaderBoard(): Gets user leaderboard data from Model Package.

class TeamsScreen
Holds UI of the teams screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. getUserTeam(): Gets user team data, if any.

class ShareSelfieScreen
Holds UI of the selfie screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. Boolean launchCamera(): Launches the device’s camera. sendSelfie(): Calls sendSelfie function in the Controller Package.

class MyProfileScreen
Holds UI of the profile screen.
Attributes
React Native visual components
Methods
Boolean navigate(*Screen*): Navigates to the specified screen. getUserData(): Gets stored user data from the Model Package. updateUserData(): Calls the updateUserData function in the Controller Package.

class Navigator
Holds React Navigator components.
Attributes
React Navigator components
Methods
Boolean navigate(*Screen*): Fulfills a variety of functions such as switching the active screen or activating drawer and tab menus. All navigate() functions in other classes of the View Package call this function.

3.1.2 Controller

class Container
Manages communication with other packages, commands other classes in Controller Package.
Attributes
-
Methods
Boolean login(): Sends login information retrieved from the View Package to the server for confirmation. Boolean signup(): Sends registration form to the server for registration. Boolean updateUserData(): Sends user form to the server for alteration. Boolean getHistory(): Retrieves user history from the server. Calls updateStorage. Boolean getUserData(): Retrieves user profile data from the server. Calls updateStorage. Boolean getUserTeam(): Retrieves user team data from the server. Calls updateStorage. Boolean getBadges(): Retrieves user badge data from the server. Calls updateStorage. Boolean getLeaderboard(): Retrieves leaderboard data from the server. Calls updateStorage. Boolean getMapData(): Retrieves location data from the server. Calls updateStorage. Boolean sendRequest(): Sends blood request form to the server. Boolean acceptRequest(): Sends accept request data to the server. Boolean updateStorage(): Via the parameter passed to this function, syncs the server and local data. Boolean createNotification(): Creates a notification object upon directives received from the

server. Boolean sendSelfie(): Calls sendSelfie function in the Controller Package.

class Storage
Holds functions that update the Model Package.
Attributes
-
Methods
Boolean updateUser(): Syncs user and server data. Boolean updateBB(): Syncs blood bank and server data. Boolean updateTeams(): Syncs teams and server data. Boolean updateReservation(): Syncs shown blood requests and server data. Boolean updateDonation(): Syncs user requests and server data.

class API_Controller
Manages communication with the server via the Axios framework.
Attributes
-
Methods
Boolean login(): Sends login information retrieved from the View Package to the server for confirmation. Boolean signup(): Sends registration form to the server for registration. Boolean updateUserData(): Sends user form to the server for alteration. Boolean getHistory(): Retrieves user history from the server. Calls updateStorage. Boolean getUserData(): Retrieves user profile data from the server. Calls updateStorage. Boolean getUserTeam(): Retrieves user team data from the server. Calls updateStorage. Boolean getBadges(): Retrieves user badge data from the server. Calls updateStorage. Boolean getLeaderboard(): Retrieves leaderboard data from the server. Calls updateStorage. Boolean getMapData(): Retrieves location data from the server. Calls updateStorage. Boolean sendRequest(): Sends blood request form to the server. Boolean acceptRequest(): Sends accept request data to the server. Boolean sendSelfie(): Calls sendSelfie function in the Controller Package.

class NotificationController
Manages notification data received from the server.
Attributes
-
Methods
Boolean updateStorage(): Calls Storage class to store the newly created notification. Boolean createNotification(): Creates a notification object upon directives received from the Container class.

class Axios
Handles HTTP queries and responses.
Attributes
Axios Framework attributes.
Methods
Axios Framework methods.

3.1.3 Model

Interface Model
Interface for all Model Classes to implement..
Attributes
-
Methods
Boolean notifyView(): Notifies the View class upon a change made by the Controller Package. Boolean updateStorage(): Abstract function implemented differently by each model.

class UserModel
Holds active User’s data.
Attributes
int userID String username String name String bloodType <Location> currentLocation String[] badges

String[] leaderboard
Methods
Boolean updateStorage(userID,username,name,bloodType,badges): Updates stored user data. Boolean updateLocation(Location): Updates current user location. Boolean updateLeaderboard(leaderboard): Updates leaderboard data.

class ReservationModel
Holds User’s reservation data.
Attributes
int reservationID int targetUserID int targetBbID <Location> targetBbLocation
Methods
Boolean updateStorage(): Updates stored data. Boolean updateLocation(Location): Updates current user location.

class BloodBankModel
Holds nearby blood bank data.
Attributes
int bbID String bbName <Location> bbLocation <Donation>[] donationList
Methods
Boolean updateStorage(): Updates stored data. Boolean updateLocation(Location): Updates current user location. Boolean updateDonationList(): Updates blood bank’s donation list.

class DonationModel
Holds User’s donation data.
Attributes
int[] donationIDList int[] donationTargetIDList, int[] donationBBIDList String[] donationBTypeList

Methods
Boolean updateStorage(): Updates stored data. Boolean updateLocation(Location): Updates current user location. Boolean updateDonationList(): Updates blood bank’s donation list.

3.2 Server

3.2.1 Route

class Route Manager
The class manages http requests for other components
Attributes
React Native router components
Methods
React

class UserRoute

Https request manager for user.
Attributes
private UserManage manageUser
Methods
JSON getCallback(res,req,next) JSON postCallback(res,req, next) JSON putCallback(res,req,next) JSON deleteCallback(res,req,next)

class BloodBankRoute
Https request manager for blood bank data.
Attributes
Methods
JSON getCallback(res,req,next) JSON postCallback(res,req, next) JSON putCallback(res,req,next) public JSON deleteCallback(res,req,next)

class ReservationRoute
Https request manager for reservation data.
Attributes
Methods
JSON getCallback(res,req,next) JSON postCallback(res,req, next) JSON putCallback(res,req,next) JSON deleteCallback(res,req,next)

class DonationRoute
Https request manager for donation data
Attributes
Methods
JSON getCallback(res,req,next) JSON postCallback(res,req, next)

JSON putCallback(res,req,next) JSON deleteCallback(res,req,next)

Interface Route
Https request manager interface used by other routes.
Attributes
-
Methods

get (String path, Callable callback)
 	
post(String path, callable callback)
 	
put(String path, Callable callback)
 	
delete(String path, Callable callback)

3.2.2 Logic

class UserManager
Provides the business logic for operation related to users.

Attributes
-
Methods
UserModel createuser(Object options): creates users. UserModel modifyUser(int id, Object options): modifies users.

class BloodBankManager
Provides the business logic for operations related to blood bank.
Attributes
-
Methods
updateBloodBank (int blood_id, Object options)

class ReservationManager
Provides the business logic for operations related to reservations.
Attributes
-
Methods
ReservationModel createReservation(Object options): creates reservations. ReservationModel modifyReservation(int id, Object option): modifies reservations List<Reservation> getReservationList(int user_id): gets the reservation list

class DonationManager
Provides the business logic for operations related to donations.
Attributes
-
Methods
DonationModel createDonations(Object options): creates donations. DonationModel modifyDonations(int id, Object option): modifies donation related to a specific id. List<Donations> getDonationsList(int user_id): gets the list of donations with the user id.

class TeamsManager
Provides the business logic for operations related to teams.
Attributes
-
Methods
TeamsModel createTeams(Object options): creates teams. TeamsModel modifyTeams(int id, Object option): modifies data related to a specific team List<Teams> getTeamsList(int user_id): gets the list of teams which user is in.

class AuthenticationManager
Provides the business logic for authentication operations
Attributes
Passport instance TokenManager tokenManager
Methods
AuthUserModel auth(String username, string password): Authorized user

class TokenManager
This provides the business logic for token operations.
Attributes
String secretKey
Methods
String generateToken(UserModel user, BloodBankModel restaurant): generates token. String renewToken(String token): renews token Int validate(String token): checks if the token is valid

class DatabaseManager
Provides the business logic for database operations
Attributes
ExpressJS instance
Methods
DatabaseManager getInstance():gets instance of database model. UserModel getUser(int id): gets user.

BloodBankModel: gets blood bank. ReservationModel: gets reservations. DonationModel: gets donations. TeamsModel: gets teams.

3.2.3 Data

Interface DatabaseModel
Declares the methods that are used for other model classes.
Methods
Int get() Void set()

Class UserModel
Contains methods for accessing/mutating data in the user database
Attributes
String Username: Name of the user’s account String <Hashed> Password: Password of the user’s account. String Email: Email of the user’s account. Int Phone: Phone number of user’s account.

Methods
String getUserName(): Accessor method for username. String getPassword(): Accessor method for password. String getUserInfo(): Accessor method for user information. Int getPhone(): Accessor method for phone. String getEmail(): Accessor method for email. Boolean setUserName(): Mutator method for username. Boolean setPassword(): Mutator method for password. Boolean setEmail(): Mutator method for email. Boolean setPhone(): Mutator method for phone. Boolean setUserInfo(): Mutator method for user information

Class BloodBankModel
Contains methods for accessing/mutating data in the bloodBank database
Attributes
int id String name int cap Int comm_number

Class ReservationModel
This class contains the attributes for reservation in the blood donation system.
Attributes
Int reservationDate String reservationInfo Boolean reservationStatus

Class DonationModel
This class contains information for the donation process.
Attributes
String reservationInfo Int id

Class TeamsModel
This class contains information for teams.
Attributes
Int teamId Int teamPoints

4. References

[1] *What is the history of blood transfusion?* Latest Medical News, Clinical Trials, Guidelines - Today on Medscape. (2020, December 6). Retrieved October 10, 2021,from https://www.medscape.com/answers/434176-183004/what-is-the-history-of-blood-transfusion#:~:text=The%20earliest%20known%20blood%20transfusions,Blundell%20in%20London%20in%201818.

[2] U.S. Department of Health and Human Services. (n.d.). *Blood transfusion*. National Heart Lung and Blood Institute. Retrieved October 10, 2021, from https://www.nhlbi.nih.gov/health-topics/blood-transfusion.

[3] Unified modeling language. https://en.wikipedia.org/wiki/UnifiedModelingLanguage, May 2011. Accessed on 4-10-2020

[4] Ieee reference guide. https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. Accessed on 4-10-2020.