جـامعة بيروت العربية
BEIRUT ARAB UNIVERSITY

Faculty of Science

Department of Mathematics and Computer Science

Senior Project (CMPS 444)

SugarSense: Insulin Dosage Optimization

Student Name:	**Student ID:**
Ali Sinno	202204206
Layal Shakhashiro	202204167
Mariam Akkawi	202201541
Mohammad Karim Abo Hosa	202202539

Supervisor:

Dr. Lama Affara — Principal Supervisor

December, 2023

Acknowledgments

We are overcome with a sense of success and great thankfulness as we approach the end of our college experience. We would like to take this opportunity to thank you, Dr. Lama Affara, from the bottom of our hearts for your unfailing support, advice, and faith in our talents throughout the entirety of the senior project.

Your constant guidance and support have been crucial in developing our project and making it a noteworthy undertaking from the very beginning. Your enthusiasm for the topic, your knowledge, and your dedication to fostering young minds have been absolutely encouraging. Your insightful advice has improved our understanding while also giving us the self-assurance to reach greater heights, exercise critical thought, and conquer obstacles.

In addition, we would like to sincerely thank our family and friends for their continuous encouragement, inspiration, and tolerance during this difficult attempt. Our achievement has been made possible by their unfailing love, belief in us, and ongoing inspiration. We dedicate this initiative to them as a token of our unwavering support because we are fortunate to have such a robust support system.We also owe a great deal of gratitude to the exceptional professors and advisors who have helped to shape our academic careers. Their enthusiasm for education, devotion to academic success, and attention to developing students' potential have had a significant influence on our intellectual growth. We are thankful for the opportunities they have given us to broaden our knowledge and cultivate a passion for lifelong learning.

Many thanks and kind regards,

SugarSense’s team.

Abstract

Type 1 diabetes is a serious condition that affects a lot of people nowadays. Patients are required to inject a specific insulin dosage after every meal to regulate their blood glucose level. In fact,most patients struggle with that and often miscalculate the injections leading to fatal health problems. Thus,we came up with a solution to all their problems.

SugarSense is a smart mobile application that can determine the appropriate dose of insulin to administer to diabetic patients. The app provides an accurate dosage estimate by using reinforcement learning as an adaptive artificial intelligence technique integrated into our database of meals and their corresponding carbohydrate levels, lowering the danger of low or high blood sugar. Additionally, this adaptive learning improves insulin approximations' precision with each subsequent user input. The application will also provide doctor communication with patients, where we will provide diabetic patient data in the form of reports and grant the doctor access to it whenever they need it. We have plans to scale up this app to reach non-diabetics who are also interested in the glucose levels such as athletes.

We built SugarSense Mobile App and admin panel as a very important tool for anyone struggling with this condition in addition to associated health care professionals. Our App eases the calculation process, as well as links every patient to their doctor, and most importantly insures the patients' health throughout their lives.

Table of Contents

Chapter 1. Introduction

Do you know how many people suffer from diabetes? Approximately 1 in 10 adults are diagnosed and this number is predicted to rise in the upcoming years according to the International Diabetes Federation. Aside from the physical constraints, it is also emotionally challenging due to it changing their whole lifestyle including their meal patterns, physical activity, and medication. Especially for the patients who are required to take daily insulin injections. That's where our project comes to the rescue.

A. Main Project Description

SugarSense is a mobile application mainly targeting diabetic patients to assist them in calculating the right amount of insulin dosage to be taken for every meal . The users only input their glucose level and other relevant patient information, in addition to the amount of carbohydrates in the meal, and our application returns the optimal dosage of insulin. Then this app saves all the user's historical records: blood glucose levels, insulin usage, meal patterns and other relevant factors. This will allow admins using the app to visualize the data in the form of a medical report that can be sent to the patient's doctor. The users will have access to a public database that optimizes according to each patients' inputs. Additionally, we can scale up this application to fulfill the needs of non-diabetics such as bodybuilders or people who are just interested in keeping track of their carbohydrate intake.

B. Problem Statement

Despite the best efforts from diabetic centers and doctors to help and teach diabetic patients how to deal with their new condition. Newly diagnosed patients still struggle to adjust to the new lifestyle. Calculating the right amount of insulin dosage can be complex, especially for type-1 diabetics who are mostly diagnosed from a very young age. Also, this extends to uneducated people who are unable to come up with the correct dosage needed. Wrong carbohydrates approximations may lead to incorrect insulin dosage. All of these factors can induce critical health issues like Hypoglycemia that can result in: Weakness, confusion and even seizures, or Hyperglycemia that may cause permanent damage to the eyes, nerves and kidneys. Thus, the main goal of this application is to minimize those risks and insure the patient's health.

C. Project Goal

The major goal of our project is to provide all diabetic patients with a mobile application that will make it easier for them to adjust to their new lifestyle and calculate their insulin dosage. Using machine learning, which will analyze the

processed data and information gathered from previous insulin calculations of other patients, will scan and classify the wrong assumptions from the right ones and recognize which meal should be adjusted to optimize the dosage. That way, we can also reduce the possibility of errors and therefore prevent fatal health problems that may occur. Graphical data can provide the doctor with a more thorough and precise report of their patient's condition over the course of a month, thus improving the accuracy of the treatment process. But most importantly, our software will allow us to build a community for all diabetic users and educate them throughout their journey.

D. System and Domain Review

Our application SugarSense is designed to assist diabetic patients in managing their insulin dosages effectively.

To evaluate our system, we need to consider several key aspects:

-Integrating an intuitive and user-friendly interface, allowing users to input relevant information such as blood glucose levels, carbohydrate intake, current insulin dosage , and calculating an accurate intake bolus dose. It should also provide clear instructions on how to use the application effectively.

-The system should allow users to input their relevant data easily. This may include either allowing manual entry of glucose reading. It should securely store and protect sensitive data.

-The application should utilize self-correcting algorithms to analyze the user's data and provide optimized insulin dosage recommendations. These algorithms should consider factors such as blood glucose levels, carbohydrate intake, and individual insulin sensitivity. The accuracy and reliability of these algorithms are critical for the system's effectiveness.

-The application should have a notification feature to remind users to take their insulin doses at the appropriate times.

Understanding the domain of insulin optimization is also crucial, considering:

-Integrating different types of diabetes where each requires distinct approaches. For example, individuals with type 1 diabetes rely on insulin for survival, whereas those with type 2 diabetes may necessitate the addition of insulin to their current treatment regimens.

-Several insulin routines are available, such as multiple daily injections (MDI) and insulin pump therapy. The application should be flexible to support various routines and offer optimization suggestions accordingly.

-The application should consider individual variability and adjust recommendations accordingly since each person's response to insulin can vary.

-The application should incorporate safety checks to avoid recommending high or low insulin doses.

-The application should allow users to set personalized goals and preferences. It should consider factors like lifestyle, and meal timing when providing insulin recommendations.

The basic idea of our application is not new. However, existing applications lack certain features that we hope our system can provide. Among these existing systems, the most commonly used are Beat Diabetes [8], mySugr [9], and diabetes:M [10]. Summarized below is the availability of a few features present in our application. As shown, none of these three applications have all the features found in SugarSense. Our users will be able to get accurate bolus calculations, track their data, connect their data with their doctor and have access to informative articles as a guide book for patients who got diabetes recently.

Criteria	Beat Diabetes	mySugr	diabetes:M	SugarSense
Informative Information	✓	✗	✓	✓
Bolus Calculation	✗	✓ Only available for certain countries	✓ May require carbs information from user	✓
Track data	✗	✓	✓	✓
Doctor Connection	✗	✗	✗	✓
Connection to blood sugar meters	✗	✓	✗	✗
Bolus calculation self correction	✗	✗	✗	✓

Figure 1: Comparison table between other diabetes applications and SugarSense

Chapter 2. Project Plan

A. SDLC Model

To ensure maximum efficiency in work flow and quality of our product, we opted to follow an agile software development lifecycle model that fits our application SugarSense. Working with an SDLC model allowed us to improve the communication among the team members and distribute tasks and responsibilities. It also helped us overcome most of the challenges we have faced in risk management and implementation rework that may have resulted from certain mistakes. So we decided to go with the Scrum Framework that corresponds to the Agile methodology, emphasizing on collaborative decision-making and full transparency between members.

Scrum is considered fast, simple, flexible and adaptive to changes. This framework focuses on delivering value and quality to the customer throughout the development process by dividing it over multiple short sprints. We had weekly sprints where we met to specify the proceeding tasks. Our development team whose roles include analysis, design, implementation and testing, consists of four members: Mohammad Karim, Layal, Ali and Mariam. Our scrum master –Dr. Lama Affara, coordinated our meetings to supervise our work and ensure we're on the right track.

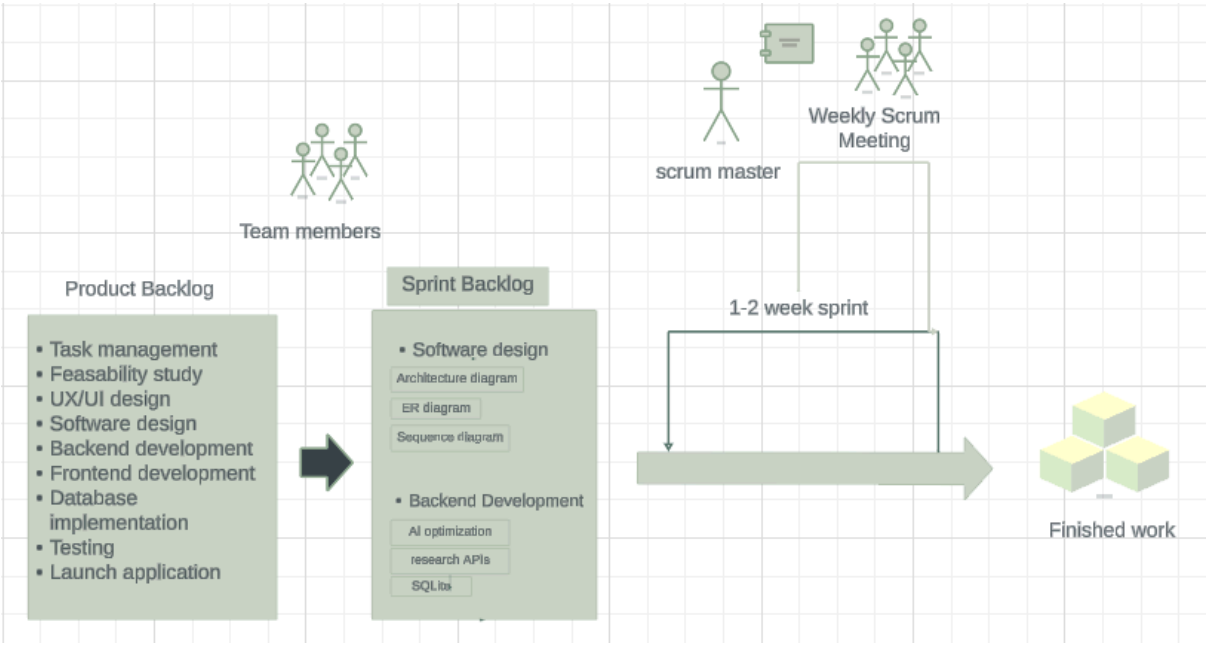


Figure 2: Sprint Backlog figure

B. Project Organization

The most critical part of every project's implementation phase is to specify every member's role. So we had a meeting to discuss which task fits every member's skills and capabilities. Layal was responsible for the design; that includes the UX/UI. She made sure

the designs of the application are user friendly. Next we have Mariam who organized the database structure and models to easily store and access data. This helped facilitate the implementation of the backend that was taken by Mohammad Karim. Finally, Ali was responsible for the architecture of the application.

C. Schedule/Timeline

In order to distribute the tasks among the team members and prevent any possible delay of the project, we created a weekly schedule all through the fall semester to ensure quality and fast productivity are met. We assigned the tasks as follows:

Time(in weeks)	Tasks	Team Members
Week 3	Generate Project Idea	Everyone
Week 3	Discuss with Advisor	.....
Week 4	Insulin Optamization Research	.....
Week 5	Project Proposal	.....
Week 5	Project Description	ali
Week 5	Problem Statement	ali/mohammad
Week 5	Project Goal	ali
Week 6	System And Domain Review	mohammad/layal
Week 6	SDLC Model	mariam
Week 6	Project Organization	mariam
Week 7	Ethical Standards And Guidelines	ali
Week 7	Schedule	mariam
Week 7	Feasability Study	Layal/mohammad
Week 7	Project Plan Report	Everyone
Week 8	Present to Committee	.....
Week 9	Software Requirements	layal/mohammad
Week 9	Use Case Diagram	mariam
Week 10	Data Flow Diagram	ali
Week 10	Database Diagram	layal/mohammad
Week 11	Relational Diagram	layal/mohammad
Week 11	Sequence Diagram	mariam
Week 12	Software Architecture	Ali
Week 12	User Interface Prototype	layal
Week 13	Methodology	.....
week 14	Final Report Draft	Everyone
Week 15	Final Report	Everyone
Week 15	Present to Committee	Everyone

Figure 3: Tasks Assignment table

Upon starting the implementation phase, we initiated the SugarSense project and employed it on the “Jira Software” which is an online management tool mostly used in agile development projects; in our case the Scrum model. We divided our work into multiple sprints with each task assigned to a member. We set up a visualization of our progress with the help of this software, it sent us alerts every time our tasks were overdue and highlighted the low/high points in our workflow. Overall, using “Jira” helped us keep better track of our performance and ensured an efficient and quick implementation of our application.(Appendix D)

D. Ethical standards and guidelines

Business ethics goes beyond just a moral code. So it was our priority for SugarSense to uphold the highest standards of integrity, responsible behavior and ensure this code is upheld by colleagues and co-workers. We will guarantee complete transparency and honesty with our users. We will always act fairly in favor of every member, support the welfare of our community and promote appropriate policies and practices. Therefore SugarSense will:

- ✔ ensure the privacy and confidentiality of its users by protecting their data.
- ✔ Guarantee a safe and private transfer of medical report between patients and their doctors.
- ✔ ask the user for authorization if they agree to share their data into the public database.
- ✔ Guarantee authenticity of information to support a healthy safe environment of care for diabetic patients.

E. Feasibility Study

a. Risk Management

Risk management is concerned mainly with the minimization of potential risks that may occur during the development process. It usually considers factors like budgets, schedules, personnel and resources issues; and generates a plan to avoid and solve those problems. The main 3 types of risks that we may face are:

- ❖ **Project risks:**

1. The app recommending a wrong dosage is a significant risk thus we must design the app to minimize the risk of errors. We will achieve that by using as much data as possible and machine learning.

2. Another risk is the small spread of the app between diabetic people. We will ask doctors to recommend the apps to their patients (they could get discounts or free trials).

3. The app should be certified for diabetic people so that our app would be legal to use in as many countries as possible. Thus, we would talk to health authorities and apply for the app to be certified.

- **Time risks:**
  It's the risk concerning the delayed delivery of our application. That our project will take longer than expected to be launched which can impact negatively on the budgets, resources and overall performance. To guarantee a quick delivery, we overestimated the time of production, so that the deadline of our project is one week earlier. Therefore, ensure smooth workflow and equal division of tasks among team members.

- **Technical risks:**

  This risk focuses on the loss of some data which can be harmful during the development of the application. So to completely avoid this dangerous problem, we will implement a regular data backup. We kept several copies of our work separately and shared it among the team members in case of any system crash or deleted files.

b. Operational Feasibility

Operational feasibility assesses whether our system can be used after the project is completed and put into action and is dependent on the human resources that are available for it. So we made an online survey for diabetic patients to see if they find our idea useful to them along with a website that explains the application and the prototypes to check if it's user-friendly. We searched some surveys made in Lebanon from the World Health organization, Lebanese Diabetes Society, International Diabetes Federation, Ministry Of Public Health, DiaLeb and other worldwide studies revolving around type 1 diabetics. And turns out the majority of diabetics struggle with the calculation of daily insulin dosages. (Appendix A)

So we believe that SugarSense can be a major breakthrough for diabetic patients in Lebanon. Since most newly diagnosed patients have trouble

adjusting to their new lifestyle, our application can ease their transition and help ensure the safety and health of patients.

We also made an online survey dedicated for doctors (Appendix B) to see whether they'd be willing to suggest our application to their patients. But unfortunately, we couldn't reach a sufficient number of doctors to be able to conduct a study regarding this matter. But we're still trying to further share it to get some results soon.

c. Technical Feasibility

The study of technical feasibility focuses on identifying the hardware and software needed to successfully implement our project and meet user needs. SugarSense is easily deemed technically possible as it only requires software and programs that are now available in the market. This provides the necessary languages and frameworks to build a website and a mobile application, making our proposal theoretically possible.

- Hardware: our own personal computers, server for database.
- Software:
  - Editors, Libraries, and Languages (Flutter, Dart, Drift, python, sql, VS code, github…)
  - Design Software (astah, lucidChart)
  - Prototype Software (Figma)
- developers: 4 intermediate level developers.

d. Economic Feasibility

Our app should have returned money benefits thus the app will have users above the age of 22 choose between payment options:

-1. 1$ per month subscription

-2. 80$ for life-time access

Year	Number of Users	Cost $	Cumulative Costs $	Benefits	Cumulative benefit
0	0	$16,400.00	$16,400.00	$0.00	$0.00
1	4000	$8,120.00	$24,520.00	$89,000.00	$89,000.00
2	8000	$8,200.00	$32,720.00	$101,000.00	$190,000.00
3	13000	$8,300.00	$41,020.00	$134,500.00	$324,500.00
4	18000	$8,400.00	$49,420.00	$149,500.00	$474,000.00
5	22000	$8,400.00	$57,820.00	$143,000.00	$617,000.00

Figure 4: Five-Year Cost-Benefit Analysis

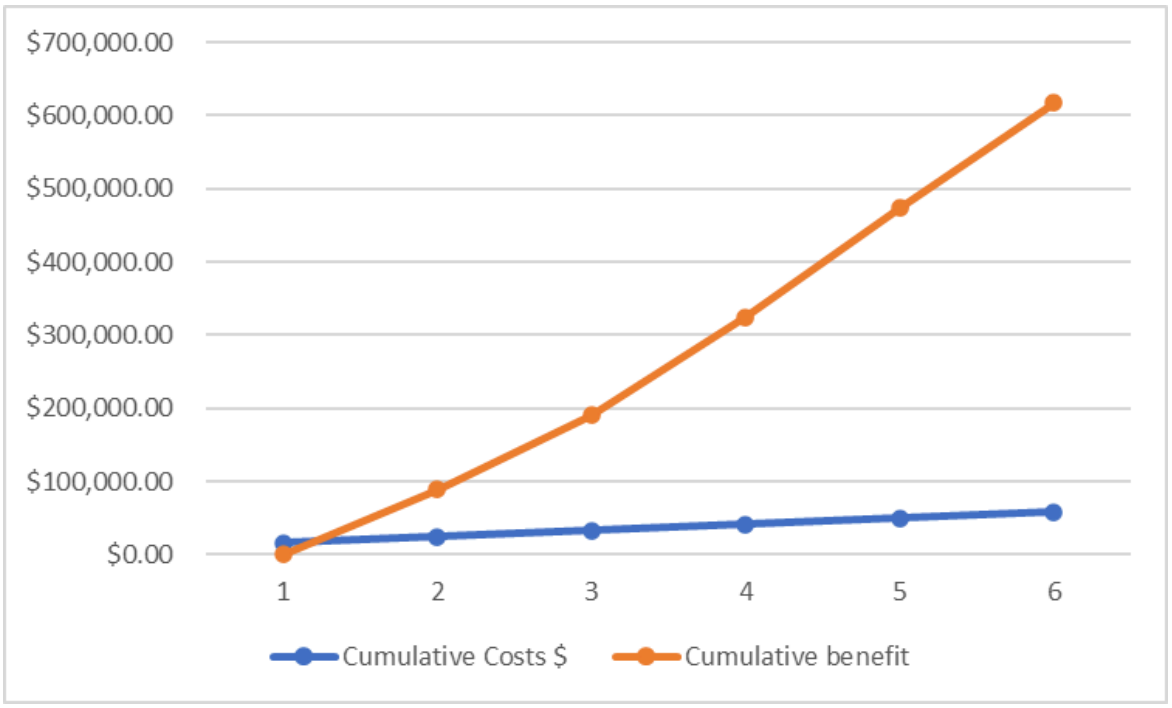


Figure 5: Cumulative Costs vs. Cumulative Benefits Over Five Years

Note:We assume that half the users will be underage. And the people above 22, half of them will use the 1$ subscription and the other half will pay the 80$.

Year	Year 0		Year 1		Year 2	
Division	half 1	half 2	half 1	half 2	half 1	half 2
						Costs
Development	$4,000.00	$4,000.00	$0.00	$0.00	$0.00	$0.00
Laptops & Mobile Cost	$4,500.00	$1,500.00	$0.00	$0.00	$0.00	$0.00
Server rental	$0.00	$0.00	$60.00	$60.00	$100.00	$100.00
Office rental	$1,200.00	$1,200.00	$1,200.00	$1,200.00	$1,200.00	$1,200.00
Maintenance	$0.00	$0.00	$2,800.00	$2,800.00	$2,800.00	$2,800.00
						Totals
Number of User	0	0	2000	4000	6000	8000
Total Cost	$9,700.00	$6,700.00	$4,060.00	$4,060.00	$4,100.00	$4,100.00
Revenue	$0.00	$0.00	$43,000.00	$46,000.00	$49,000.00	$52,000.00
Cash Flow	($9,700.00)	($6,700.00)	$38,940.00	$41,940.00	$44,900.00	$47,900.00
Cumulative Cash Flow	($9,700.00)	($16,400.00)	$22,540.00	$64,480.00	$109,380.00	$157,280.00

Year 3		Year 4		Year 5	
half 1	half 2	half 1	half 2	half 1	half 2
$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
$150.00	$150.00	$200.00	$200.00	$200.00	$200.00
$1,200.00	$1,200.00	$1,200.00	$1,200.00	$1,200.00	$1,200.00
$2,800.00	$2,800.00	$2,800.00	$2,800.00	$2,800.00	$2,800.00
10000	13000	15000	18000	20000	22000
$4,150.00	$4,150.00	$4,200.00	$4,200.00	$4,200.00	$4,200.00
$55,000.00	$79,500.00	$62,500.00	$87,000.00	$70,000.00	$73,000.00
$50,850.00	$75,350.00	$58,300.00	$82,800.00	$65,800.00	$68,800.00
$208,130.00	$283,480.00	$341,780.00	$424,580.00	$490,380.00	$559,180.00

Figure 6: Biannual Cost Distribution and Revenue Analysis for the First Five Years

Break Even Analysis:

The break even point where we'll start making profit in the 1st year with about 4000 active users (check chart). Hence, with this estimated cost and with the potential revenue of our project, we can say that our project is economically feasible.

e. Delivery

The app will have a guidebook that will help the users understand how the app works and how they can use it.

Chapter 3. Software Requirement Specification

A. Product Functions

SugarSense sets itself apart from other apps available in the market by incorporating distinct features that define its unique identity. Among these features are:

- Bolus Calculation:The application will calculate the insulin dose based on the user's glucose levels, carbohydrate intake, and insulin sensitivity.
- Meal Tracking: The users can input their meal choice by either adding a new meal and specifying added meal characteristics or choosing an existing meal from the incorporated database.
- Glucose level tracking: Users can enter their glucose readings manually.
- Graphical representation: The application can generate a visual graph in which the users will be able to trace their monthly glucose level activity.
- Doctor connection: The application will allow each doctor to have access to their patients' records.
- Educational Resources. The app can provide educational resources such as articles or links to help users learn more about insulin management, diabetes care, and healthy lifestyle choices.
- Information privacy: The application can allow users to choose how much information they are willing to expose to their doctors, to ensure the users' privacy.
- Reminders and Notifications: The application can send reminders and notifications to users for tasks such as checking blood glucose levels, taking insulin doses, and providing meal information. These reminders can help users stay on track with their insulin management routine.
- Sign in/up: The application will ask the user to either sign in if they're already an old user or sign up by creating an account.

SugarSense can be improved and updated on a larger scale. We can add features in future updates that will make SugarSense a better application. Among these "stretch" features are:

- The application will fulfill the needs of non-diabetics such as bodybuilders or people who are just interested in keeping track of their carbohydrate intake.

B. User characteristics

Our application SugarSense is meant to help diabetic patients by giving them accurate Bolus calculations to regulate their blood glucose levels. The team behind

SugarSense is trying their best to minimize users' involvement to make it much easier to use. Therefore, our users don't need to have prior skills to use the application . SugarSense's user-friendly interface will easily guide the user through the application along with our step-by-step guide website for more information. Thus, only having access to any mobile phone as well as basic knowledge of using your phone is enough.

C. Non-functional Requirements

Non-functional requirements are requirements that put constraints on the services or functions offered by the system or on timing, process, standards, etc. There are three main non-functional requirements: product, organizational, and external requirements.

1) Product Requirements:
    a. Portability:
        - Users can use the same account over several devices.
    b. Performance:
        - Launching the application should not take more than 5 sec.
        - The application can hold a large volume of user data without performance issues.
    c. Usability:
        - User friendly, simple, and interactive design.
    d. Compatibility:
        - The application is compatible with a range of different mobile devices.

2) Organizational Requirements:
    a. Standards:
        - Our application will follow all rules and regulations set by international committees for the safety and privacy of users.
    b. Delivery:
        - Our application will be launched within a year after starting development where it will first include a UI prototype which will be reviewed by diabetic patients as such we will use the feedback to improve the design and product efficiency. Followed by an immediate release after improvements are implemented.

3) External Requirements:
    a. Privacy:

- The user will have their data stored locally and it'll be backed up on the server.

b. License/Copyright Issues:
   - The application shall not display any educational resource that goes against copyright laws.

D. Domain Requirements

Domain requirements are related to the domain of our application. It is important to specify these requirements and make sure they are satisfied because if these requirements are not satisfied, our system might not work. Among these requirements are:

1. The application should be compatible with various mobile devices(IOS/Android).
2. The application should have a user-friendly interface that facilitates easy data entry, visualization, and navigation.
3. The application should obtain user consent for data collection and ensure transparent information handling practices.
4. The application should be verified by the world government as well and it should comply with relevant data regulations of the health authorities to make sure the application is safe to use.
5. The application should allow users to manually input their relevant data.
6. The application should have the ability to analyze historical data and regulate them to improve the accuracy of predictions and optimize insulin dosing recommendations.
7. The application should offer a comprehensive database of food items with nutritional information including ingredients used.
8. The application should provide features for tracking blood glucose levels.
9. The application should handle a consistent increase of users and a consistent increase in the meals database.
10. The application should allow doctors to supervise app usage to ensure it is safe to use.

E. Functional Requirements

The specification of the functional requirements is the most important task before implementing the application. It has 2 categories: user requirements and system requirements. These requirements describe in detail the features and criterias of our application. To guarantee a flawless product for diabetics, we will include every requirement mentioned below.

1. **User Requirements:**
    - Users will be given an option to connect to their doctor through Id.
    - Users can input their glucose level manually.
    - Users can input their meals by searching through the database.
    - Users can add a new meal if their meal is not found.
    - Users can customize their own meals.
    - Users can keep track of their history (insulin intake, glucose levels, meals taken…).
    - Users can search for nutritional values of specific meals.
    - Users can read articles and guidelines about diabetes.
    - The patient will be able to favorite an article for easy access.
    - Only the doctors can keep track of their patients' history.
    - Doctors will acquire a specific code after accessing their accounts which will be shared with their patients.
    - Doctors can check their schedules and set up appointments for their patients.

2. **System Requirements:**
    - The system will offer different ways for users to input their data.
    - The system will save patient's data and show detailed graphs over the past day, month, or 3 months.
    - The system will optimize itself to always provide an accurate prediction of insulin intake.

Use Case Diagrams:

The use case diagram is a behavioral UML diagram type. It visualizes the main features implemented in the application, and describes how the user will perform actions and interact with a particular system. Since in our application we have 3 differents actors each taking part in specific roles we designed the following uses cases:

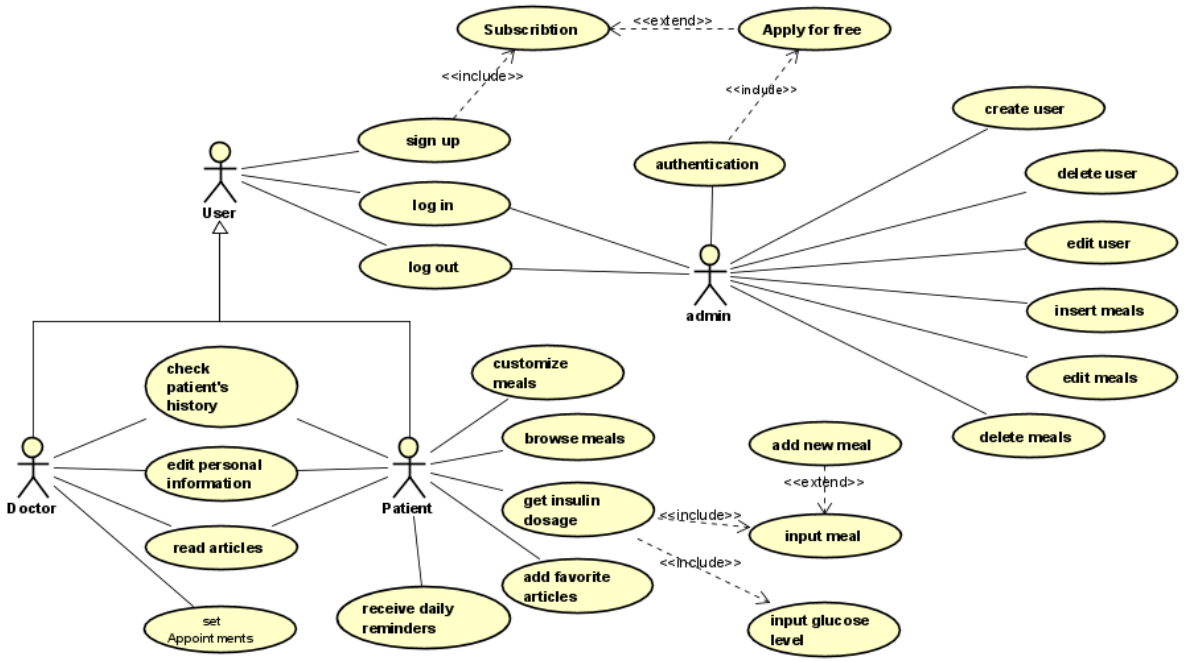


Figure 7: Use Case Diagram for User, Doctor, and Admin Roles in SugarSense

a. User/Admin use case:

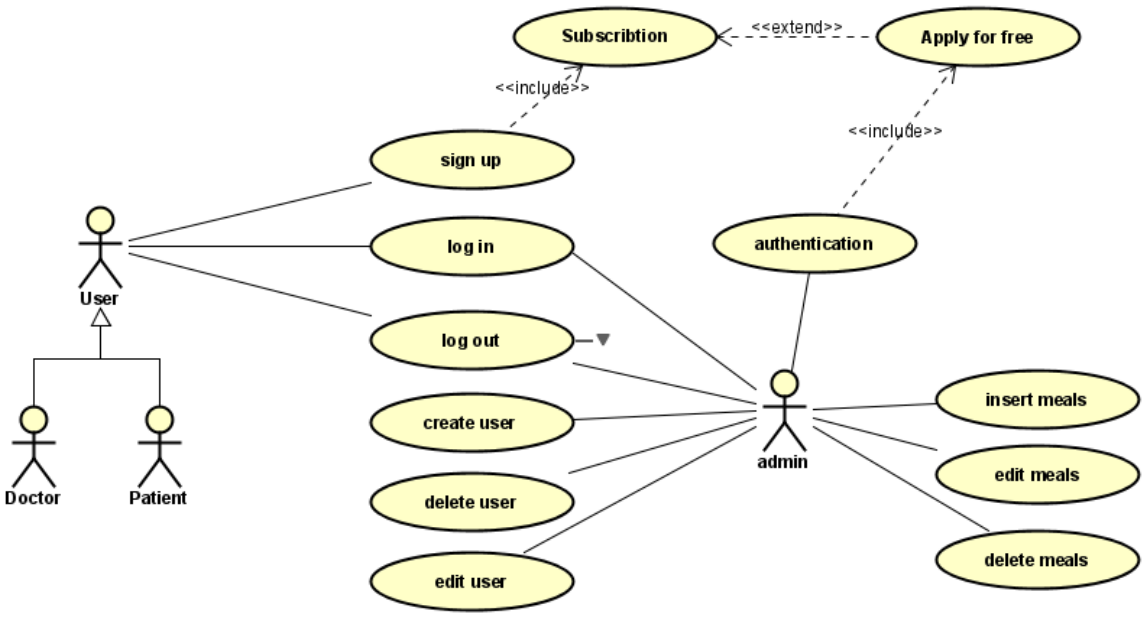


Figure 8: Use Case Diagram Highlighting Authentication Processes for Users, Doctors, and Patients

The user signing up in SugarSense can either be a patient or a doctor. The doctors can contact the admins and after verifying the doctors' credentials, the admin will give access to the doctors' website to sign in. When the doctors sign in, the website will generate a code that will be shared with their patients to collect their data. Since users under 22 years old can use the app for free, they are required to fill in a form

that will be sent to the admin who will verify their age; if accepted they will receive an acceptance email with their username and password, otherwise they have to subscribe. Then the user will be allowed to login and use our application then logout.

The admin as well can login into the application then log out. He's also allowed to create, edit or delete users in some cases if some rules and regulations are violated (user is not diabetic, user is more than 22 of age and is not subscribed…). The admin is able to insert, edit and delete meals from the database.

b. Patient use case:

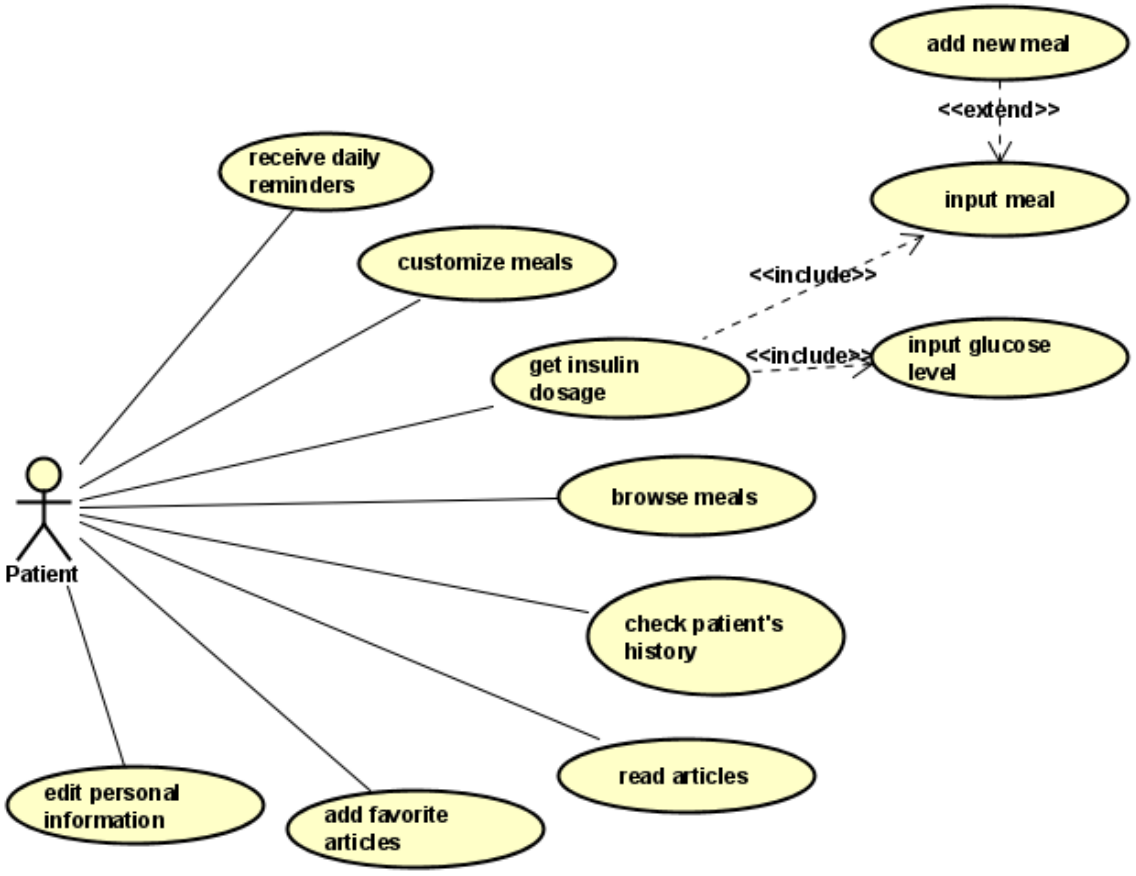


Figure 9: Use Case Diagram for User Interactions in SugarSense

After signing up, the patient will have to insert some medical information related to their diagnosis such as glucose sensitivity, carb ratio… Now the user can try to calculate their insulin dosage by inputting their current blood glucose level and meals taken.

To input the meal, the user can choose from the list of meals that will be given. If they couldn't find their meal, then they could add a new one. To do this, the user only has to input the ingredients of the meal and the application will estimate the

amount of carbohydrates to later further optimize it. The user can, additionally, customize their own meal by editing certain meals or creating new ones.

For every calculation the patient's history is saved, the user can pull up a detailed report consisting of the levels of insulin, blood glucose and meals taken, for the past week, month or year. This graphical representation can also be accessed by their doctor and the patient can specify which factors they'd like to share with their doctor.

Furthermore, the user can search for information regarding nutritional value of some meals. They can also check out some articles provided by the application and add them to their favorites. The articles mainly involve topics about diabetics' lifestyle and general information and guidelines about diabetes to help newly diagnosed patients.

c. Doctor use case:

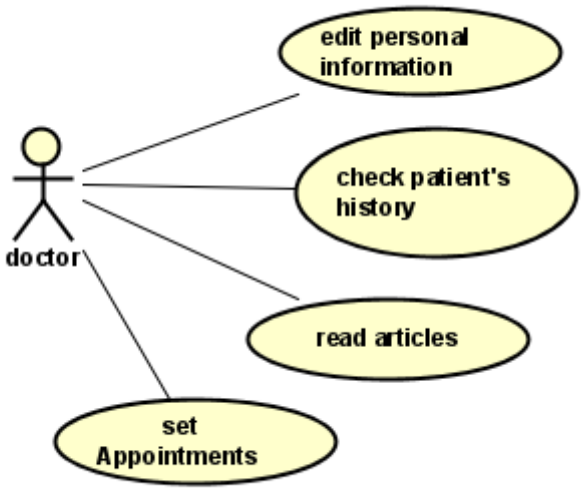


Figure 10: Use Case Diagram for Doctor's Role

The doctors mainly use the application to check on all their patients' data and examine their progress and also set next appointments for their patients and check their daily, monthly schedule. A doctor is not allowed access to users who are not registered as patients. Since our application will include links to articles about diabetes guidelines and lifestyle, the doctor can read those articles. The doctor is also allowed to edit their profile and change their personal information.

Chapter 4. Project Design

A. User Interface Prototype

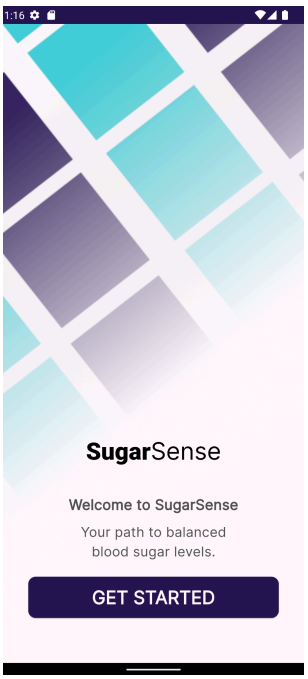
Designing a prototype is a crucial step in our application development process as it empowers us, as developers, to evaluate the functionality of different aspects and identify areas that require further refinement. The user interface prototype serves as a valuable tool for testing the workflow of our application, SugarSense, across multiple scenarios. The primary objective is to ensure that the most common user scenarios are optimized for seamless usage with minimal effort.

As evident in our prototype, SugarSense boasts a highly user-friendly interface, enabling users to navigate the application effortlessly. The prototype showcases a range of use cases that cover the comprehensive functionality of SugarSense. By exploring the prototype, users can gain a clear understanding of the features and capabilities offered by SugarSense, making it a valuable asset for both new and existing users.

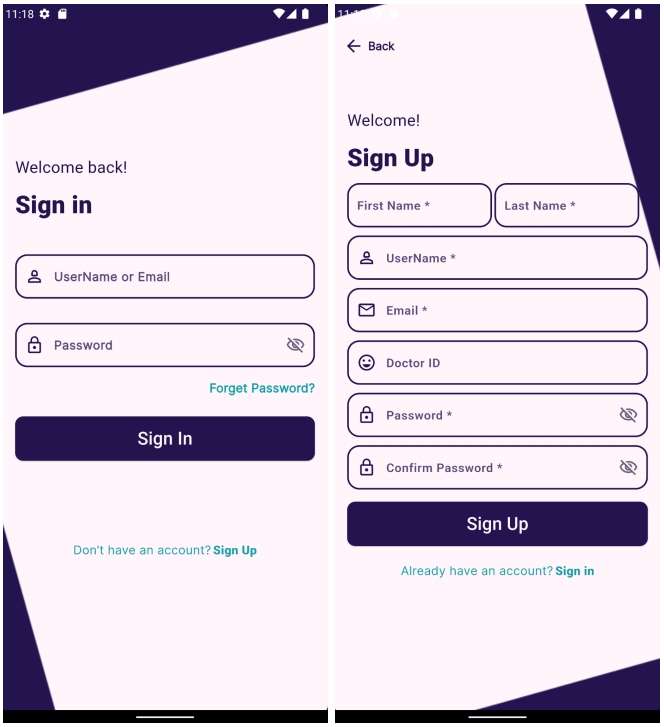
The prototype serves as a visual representation of our application's intended design and functionality, facilitating effective communication and collaboration throughout the development process.

Overall, the prototype for SugarSense plays a pivotal role in validating our design choices, identifying potential improvements, and ensuring that the application meets the needs and expectations of our target users.

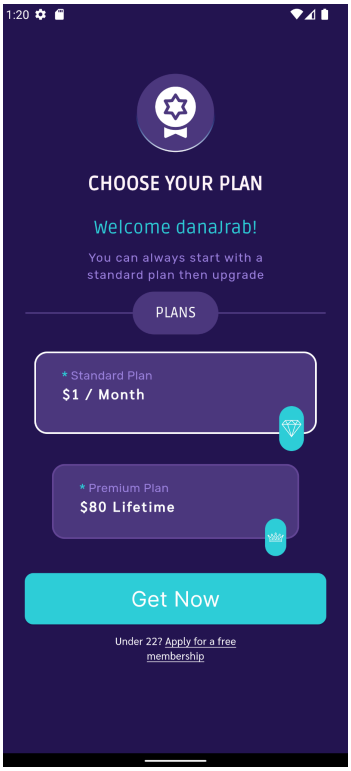
1) Launch page



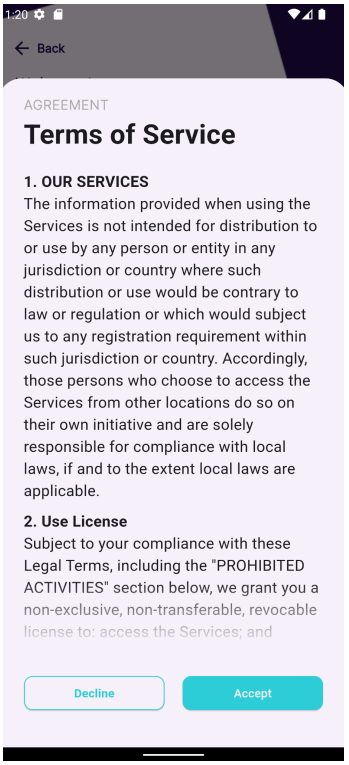
2) Sign In/Sign up page



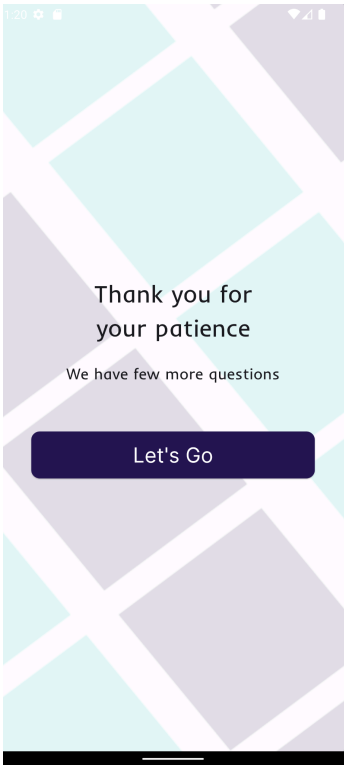
3) Subscription page

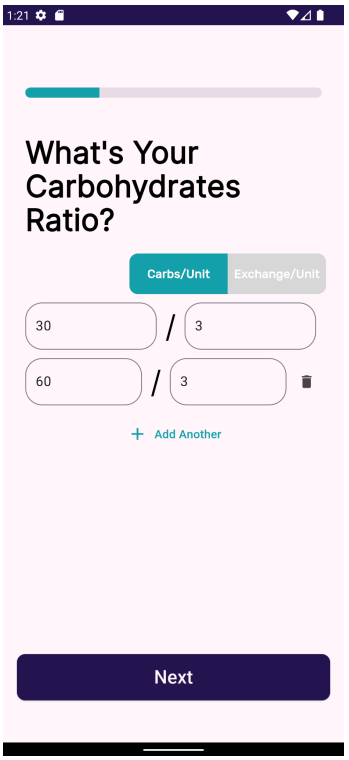


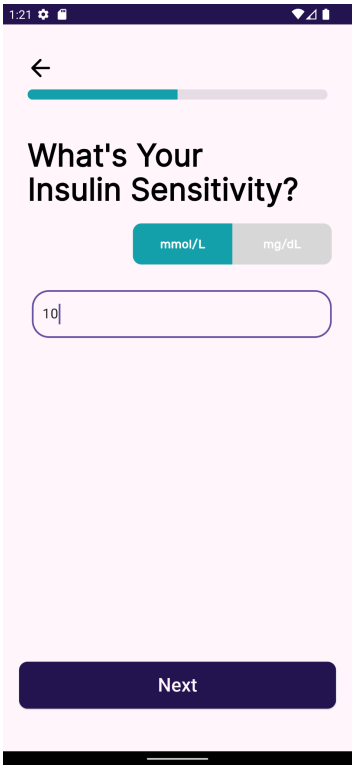
4) Terms & conditions page

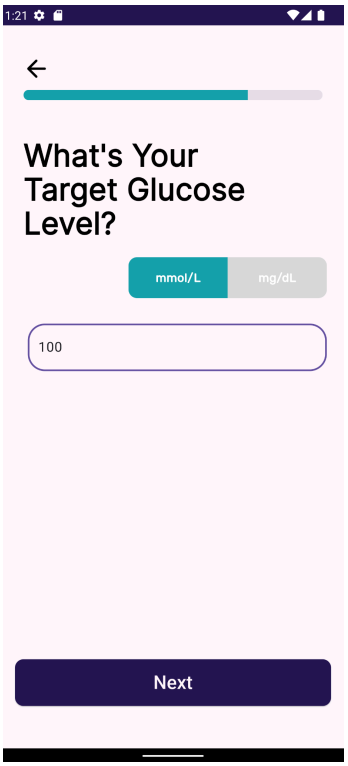


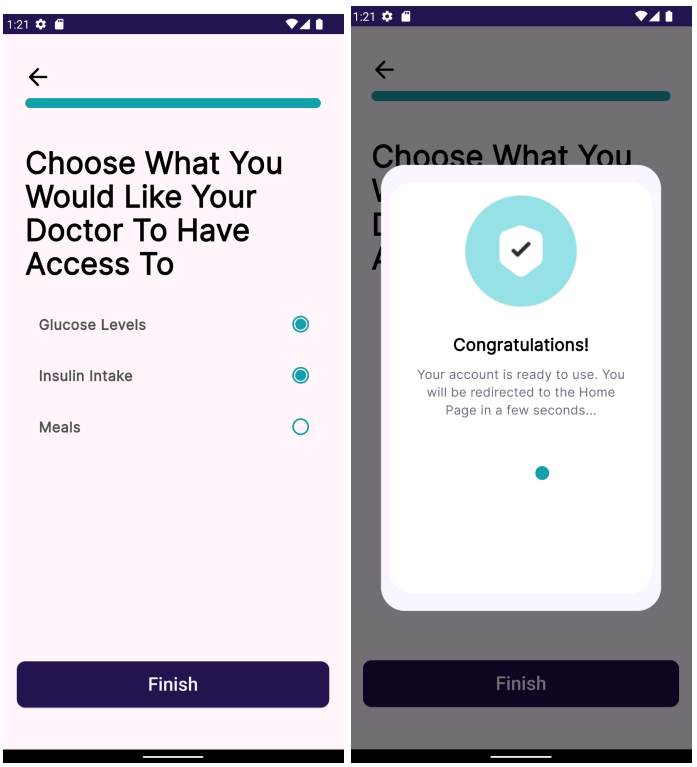
5) Set Up pages



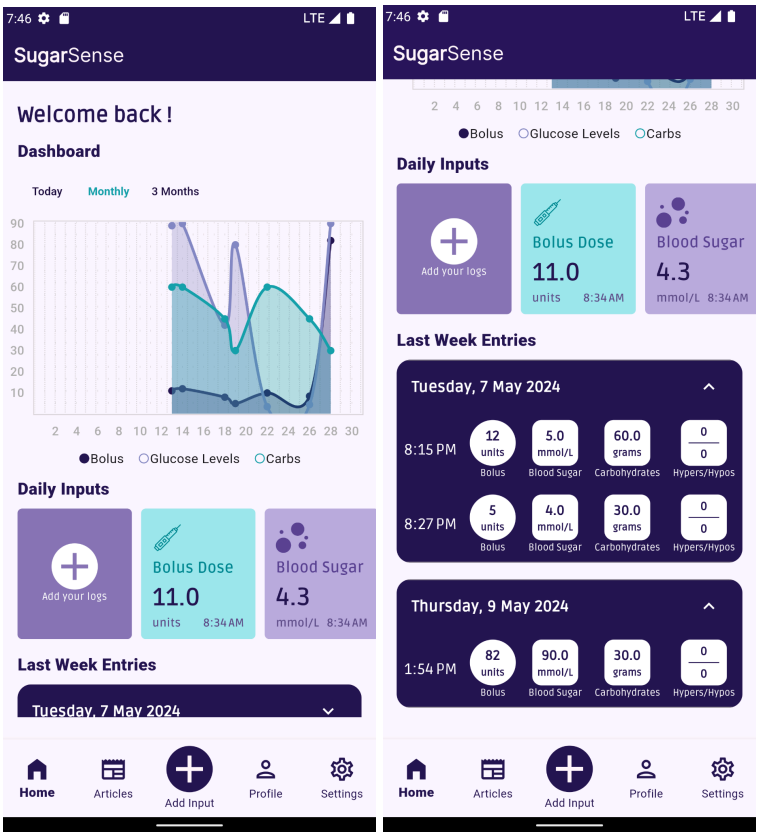




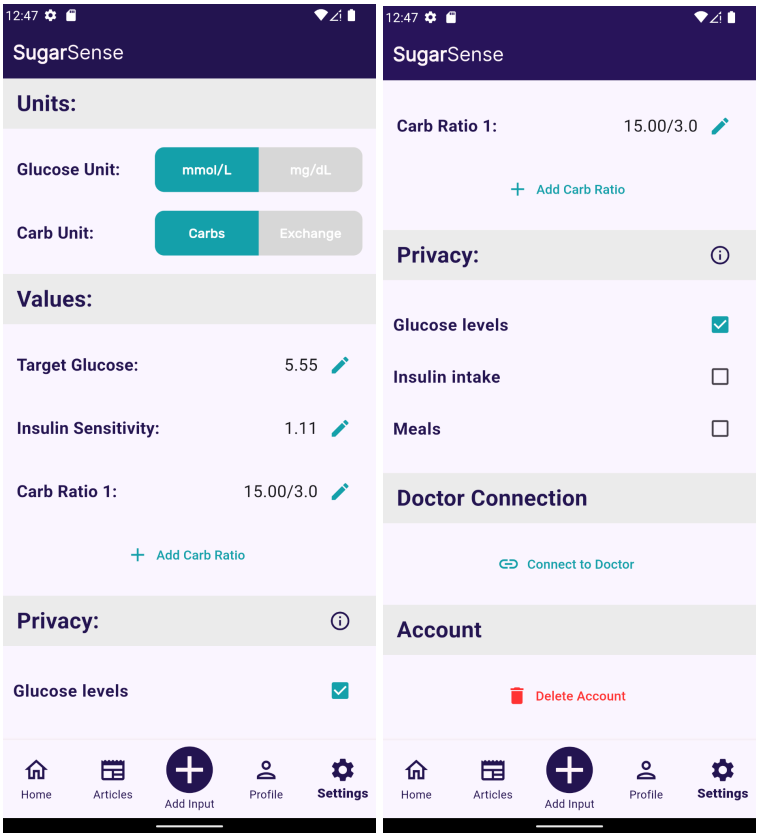




6) Dashboard page

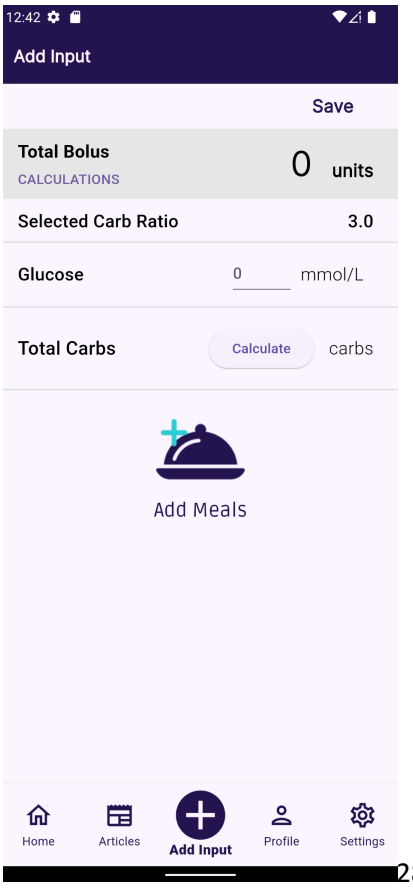


7) Setting page

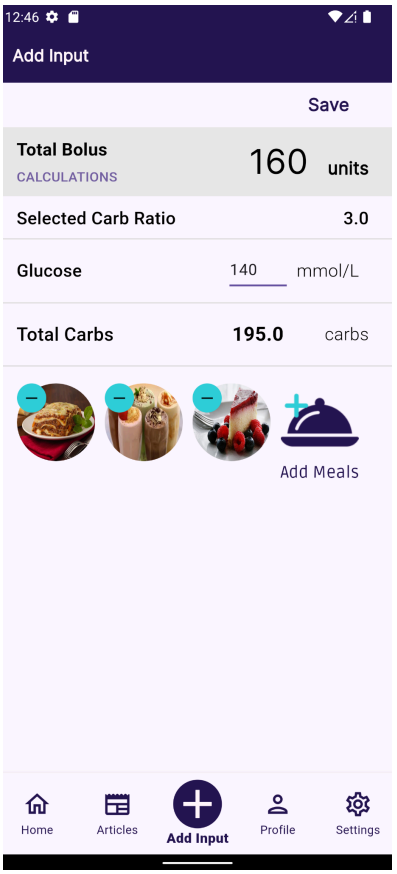


8) Add Input page/ meals page /meal Info page

Before adding meals:

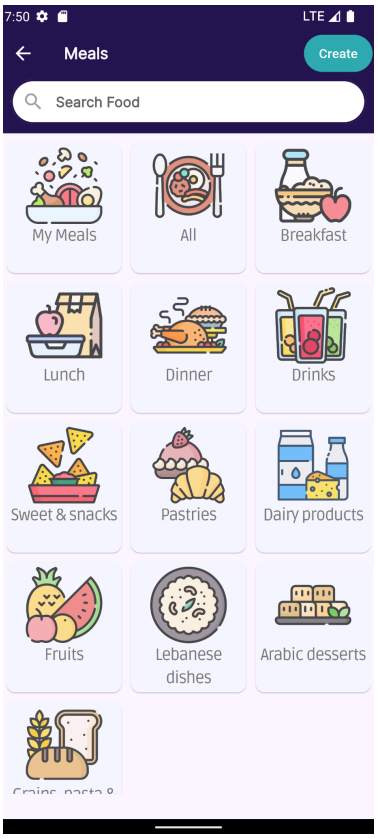


After adding meals:

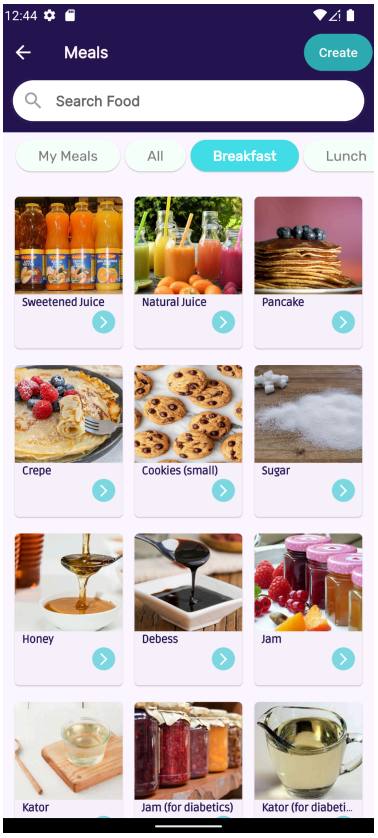


9) Meals Selection Pages

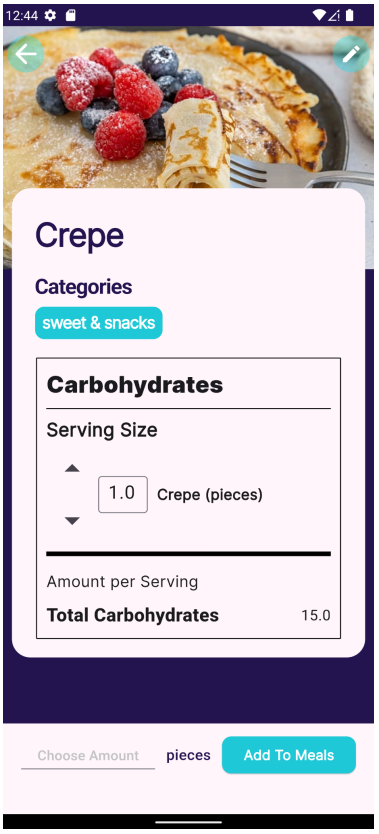
Categories Page:



Meals Page:

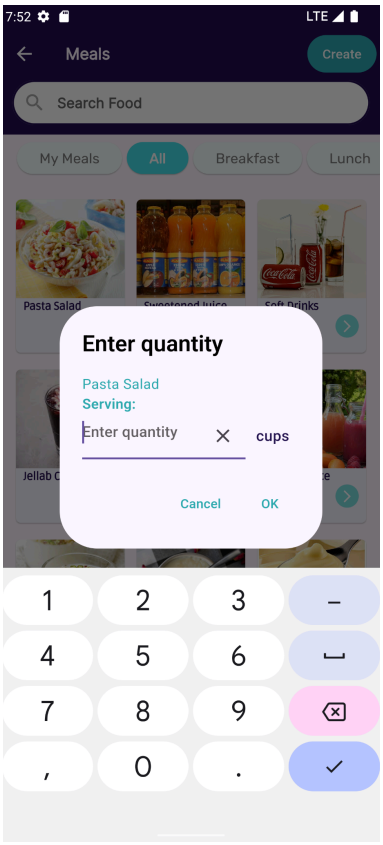


Meal Info:

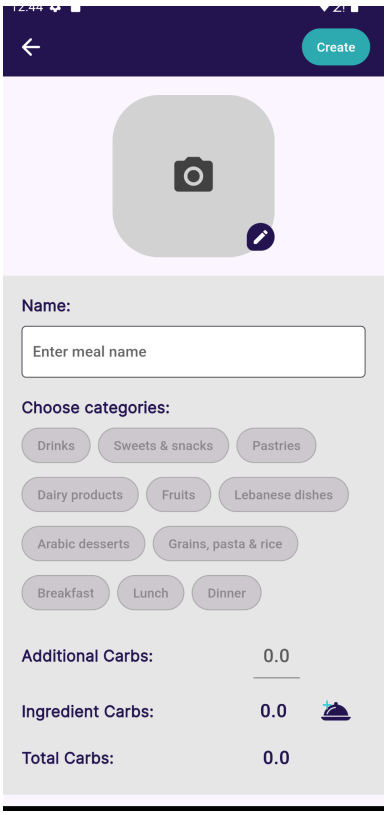


Edit Meal:

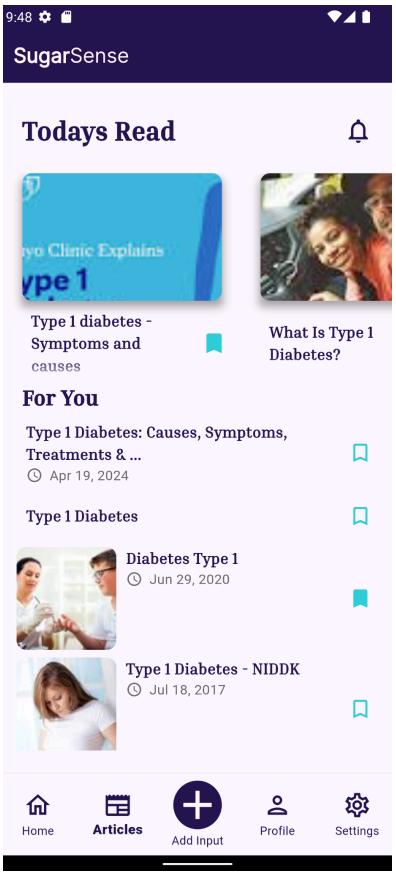
10) Quantity Selection



11) Create Meal page

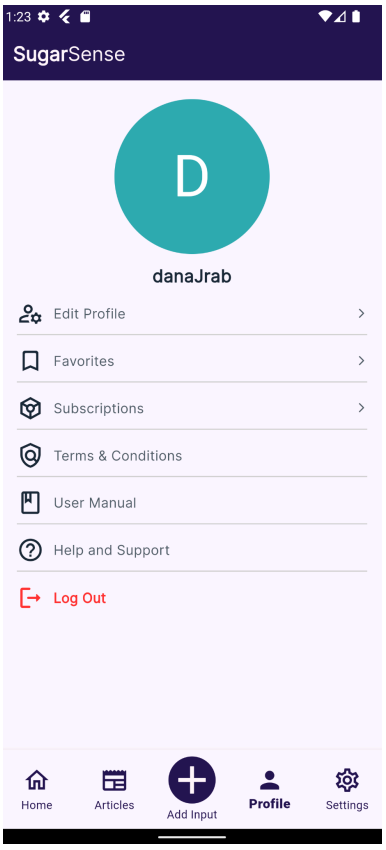


12) Articles Page

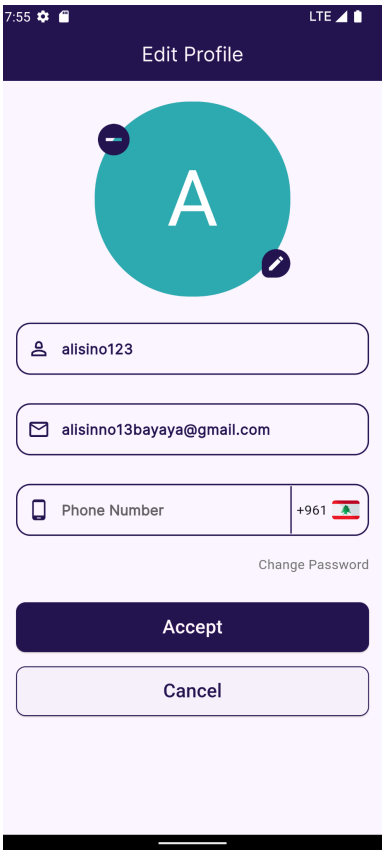


13) Account Settings Page

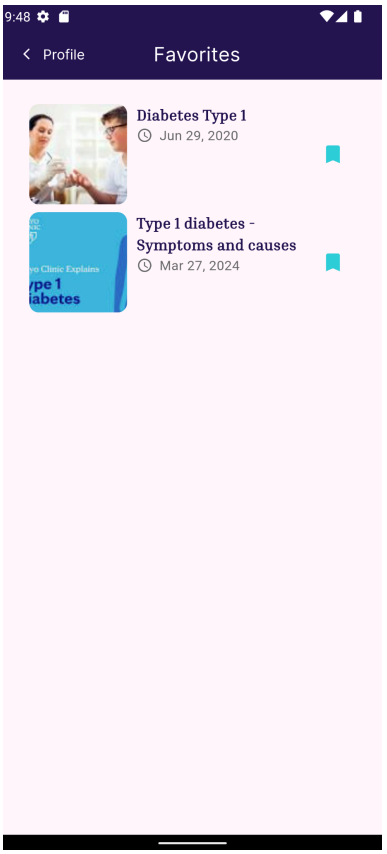
Profile Page:



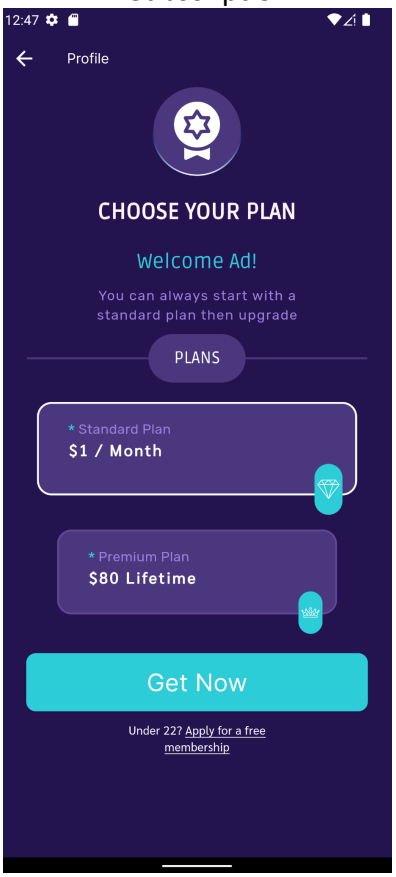
Edit Profile page:



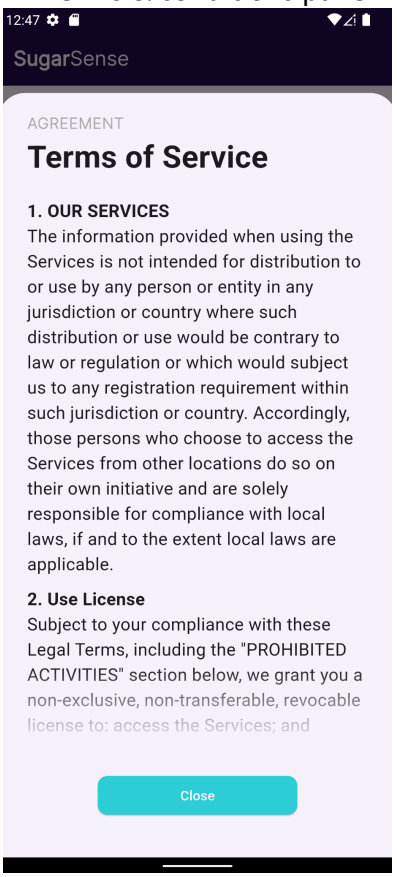
Favorites page:



Subscription:



Terms & conditions panel:



B. Data Flow Diagram

A data flow diagram (DFD) is a detailed representation of data flow through the system. It visualizes the transfer of data from user input to the database. Our application SugarSense relies on data from different categories. Firstly, the user needs to create a personal account through a unique code given to them by their doctor. After the user logs in, their personal information will be saved in the database. Whenever the user calculates their insulin dosage, the history of blood glucose level and meals taken will be also stored in the database then a graphical history of insulin intake will be shown on the homepage. The system will save and display the user's customized meals as well as their favorite article. Therefore, the DFD can represent the transfer of data with increasing amounts of detail with every level.

a. Level 0

Also referred to as "Context Diagram". It's considered the highest-level system overview out of the data flow diagrams. It mainly represents the interaction between the actor (admin, patient, doctor) and the system. It only shows the input of the users and the output the app generates.

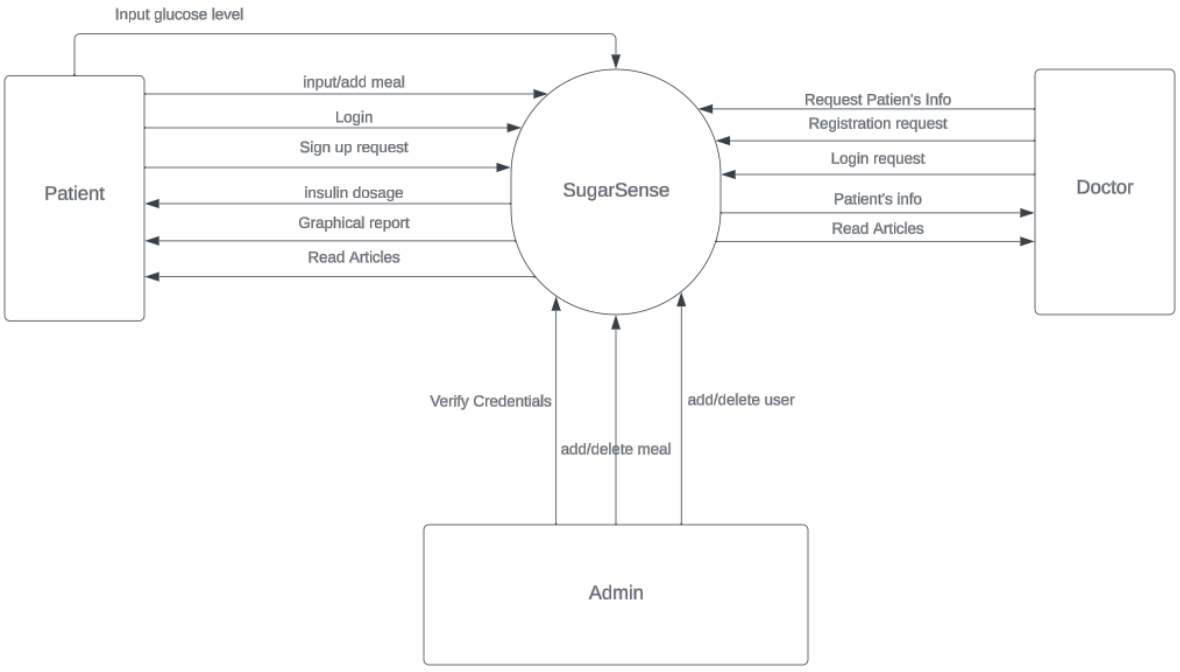


Figure 11: Level 0 Data Flow Diagram

b. Level 1

This level includes additional detail and subprocesses while maintaining a rather wide description of the system. This diagram will show the simplest process which is creating an account and signing in.

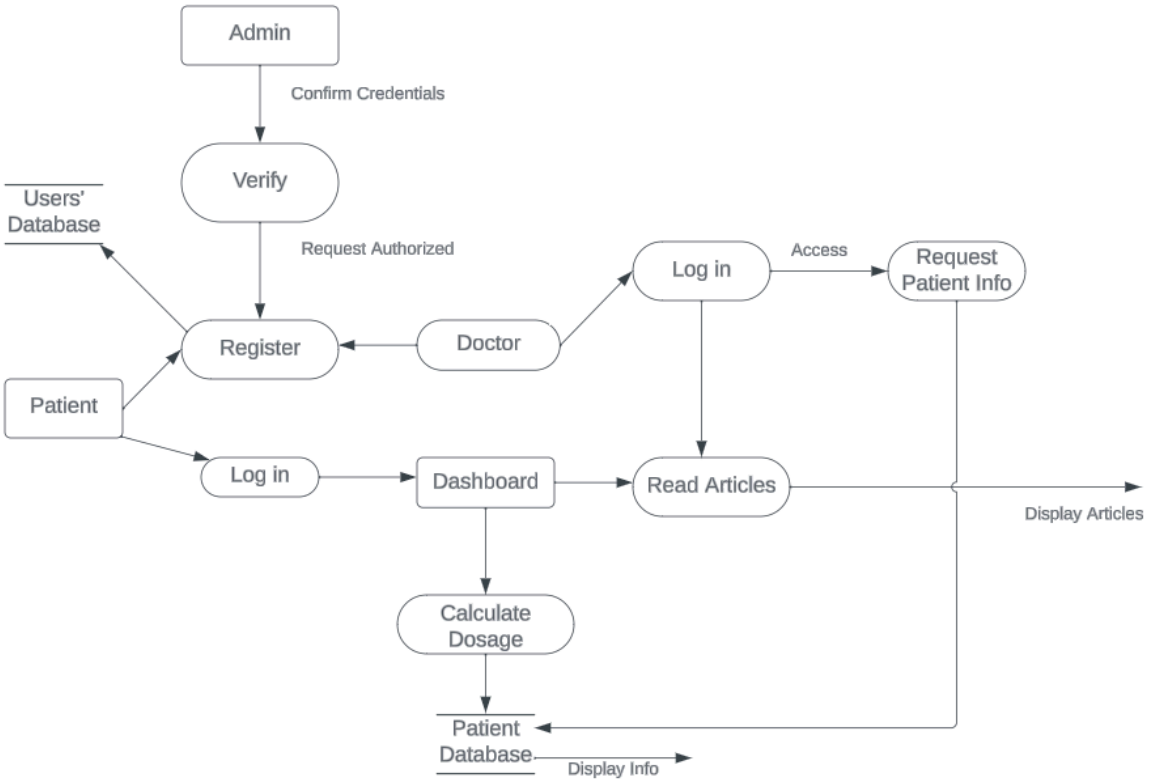


Figure 12: Level 1 Data Flow Diagram

c. Level 2

At this level, we will be taking a deep look at the process of “calculation of the insulin dosage” as well as the “handling of non existent products” process.

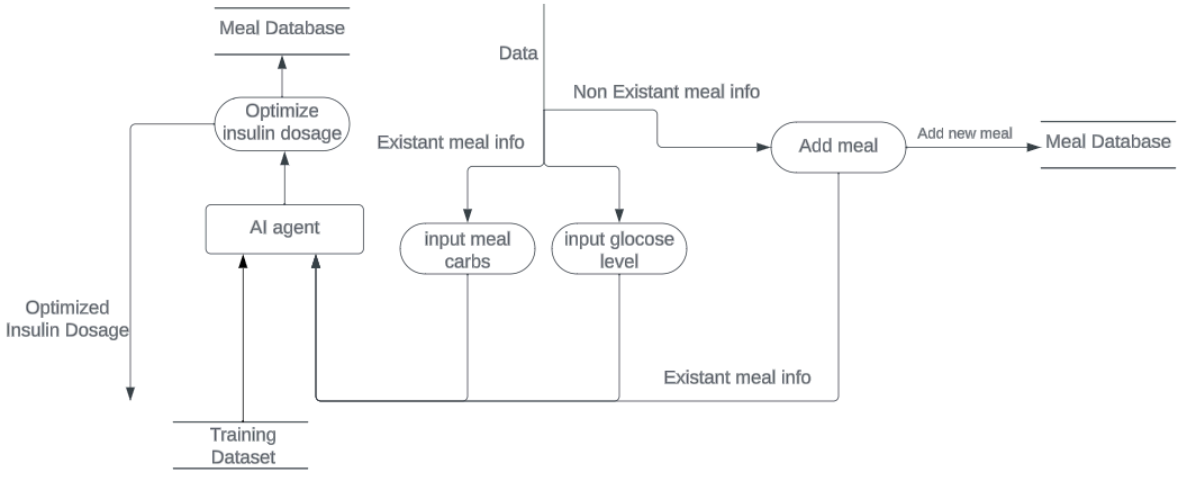


Figure 13: Level 2 Data Flow Diagram

C. Database Diagram

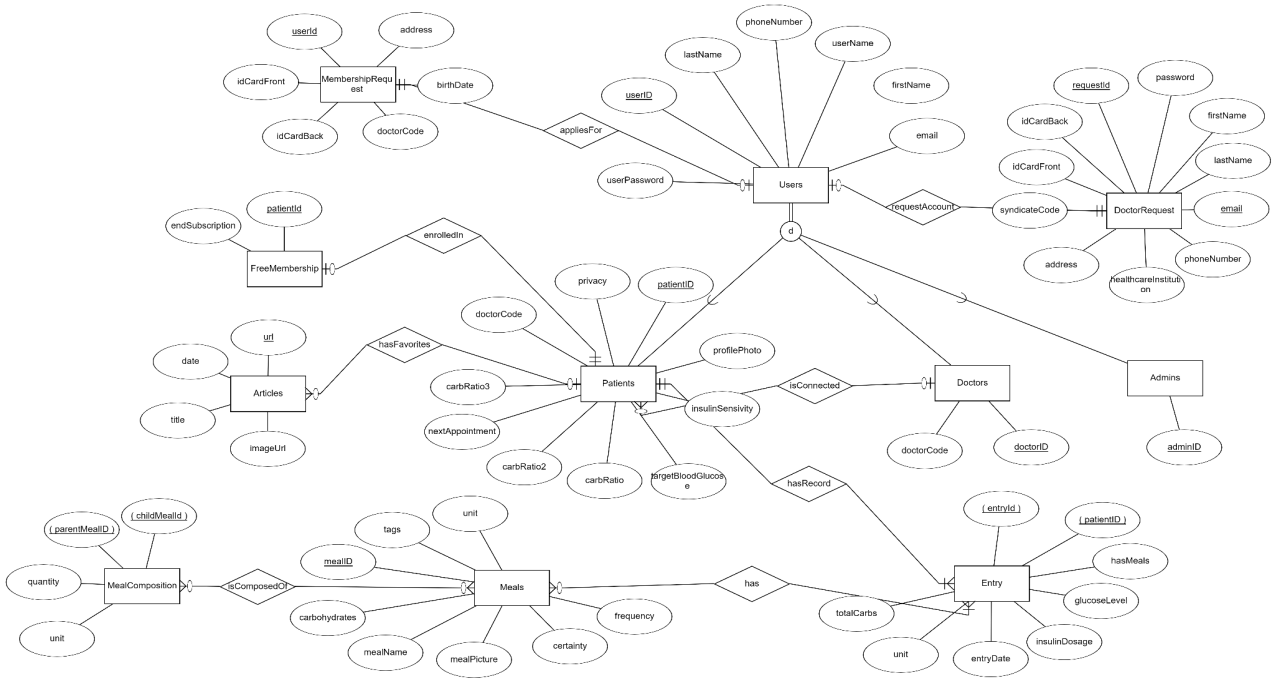


Figure 14: SQL Database Schema

SQL databases use Structured Query Language (SQL) to access and manipulate data. They store the data in tables that represent entities, and each table has rows representing records with attributes. Keys are used as unique identifiers for relationships. SQL employs a predefined and structured schema, making it organized according to a specific structure and relationships. This enables efficient retrieval and manipulation of the data while accurately handling complex relationships, which is important for precise calculations in SugarSense.

The Users table represents everyone that is using SugarSense in other words the diabetic patients and their Doctors. It has a unique identifier userID and 2 attributes for login credentials, which are userName/email and password. Additionally, each user has firstName and lastName. Two other tables, Doctors and Patients extend the Users table using doctorID and patientID respectively as foreign keys to the userID. Patients also have a profilePhoto and contact information which are phoneNumber and email, and the diabetic information carbRatio and insulinSensitivity and targetBloodGlucose. In addition, to a privacy attribute, so that the patient can specify which factors they'd like to share with their doctors. Patients will have a many-to-one relationship with Doctors to connect each patient with their doctor, this will be achieved using doctorCode attribute in the patient table as a foreign key.

An Admins table in a database typically holds the information specific to administrative users who have higher-level access privileges to manage the application. The table typically includes fields that store admin identification details, login credentials such as username and password. Admins are able to add or delete any user they want if needed whether it's a doctor or a patient. They also have the privilege to alter meals data.

The Meals table will have mealID as its unique identifier and will store the mealName and mealPicture for the user to select their meals visually. Each meal is measured with a unit and has an amount of carbohydrates, both are important for the insulin calculation, thus will be stored in the Meals table. To make the search engine more effective, the Meals will have tags, which is a string of words that could be related to each meal. A certainty attribute will be used for the AI model to use. The frequency attribute is set by default to 0 and it changes every time the patient saves a meal, which will help display his most frequent meals later on. The Meals will have a self-referencing many-to-many relationship with other Meals to indicate the ingredients of each meal using the MealComposition table which has parentMealID and childMealID both as primary keys and foreign keys for the primary keys of the meal. In each meal composition the quantity and amount of carbohydrates are also being saved.

The Entry table refers to every time the user uses the app to calculate their insulin dosage. It will have the primary key entryID and a foreign key patientID to issue a many-to-one relationship with the Patients table to link each entry to its patient. The glucose level in the blood and the amount of insulin taken will be stored as a float attribute in the Entry table as glucoseLevel and insulinDosage respectively, while the meals eaten will be stored in the hasMeal attribute as a many-to-many relationship with the Meals table through the attribute hasMeals, a string that includes every meal taken with its id, quantity and carbohydrates. In addition to the total amount of carbohydrates calculated in this entry. To give the user the ability to see their progress on a timeline, each entry will have a date attribute referring to when the calculation happened, this attribute will be of type datetime to have 1 minute accuracy.

The Article table will be used to achieve a many-to-many relationship from patients to their favorite articles using their respective unique identifiers as foreign keys. The Articles table is identified by its primary key url, and includes other attributes: title, imageUrl and date of each article.

The MembershipRequest refers to users of age less than twenty two, who’d like to apply for a free membership. The information stored are address, birthDate and

photos of his identity card from front and back. The userId will help create a one-to-one relationship with the Users table. If the patient's application was accepted, his patientID will be saved in the FreeMembership along with the endSubscription for when his free membership expires.

The DoctorRequest refers to doctors who'd like to sign up into our website. The information saved is critical for us to confirm their true identity and profession. After reviewing their application, we manually add the doctor and send him a confirmation email. The information stored here requestId, firstName, lastName, password, phoneNumber, healthcareInstitution, address, personal syndicate code and photos of his identity card from front and back. The primary key email will help create a one-to-one relationship with the Users table.

In conclusion, utilizing a well-structured SQL database provides numerous benefits for our computation-intensive app, SugarSense. It ensures reliability and integrity of data, facilitates efficient and accurate calculations, and enables precise handling of complex relationships. Therefore, a structured SQL database serves as the ideal backbone for SugarSense.

D. Relational Diagrams

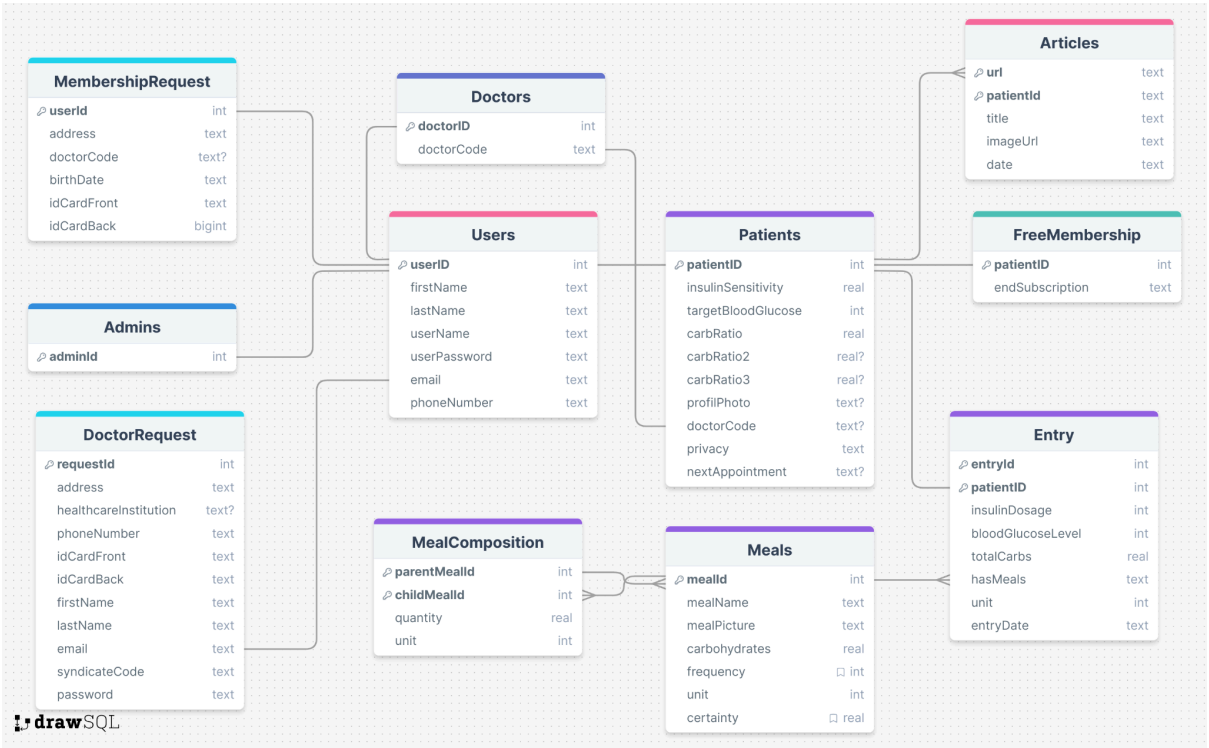


Figure 15: Relational Diagram for SugarSense

E. Sequence Diagrams

The sequence diagram shows the whole interaction between the user and the system,
and since we have 2 types of users: patient and doctor. Therefore, we created 2 sequence diagrams to show each their respective use cases.

- The first sequence diagram shows how the patients will be able to input their glucose level, as well as how to either search for a meal or add a new one to the database.

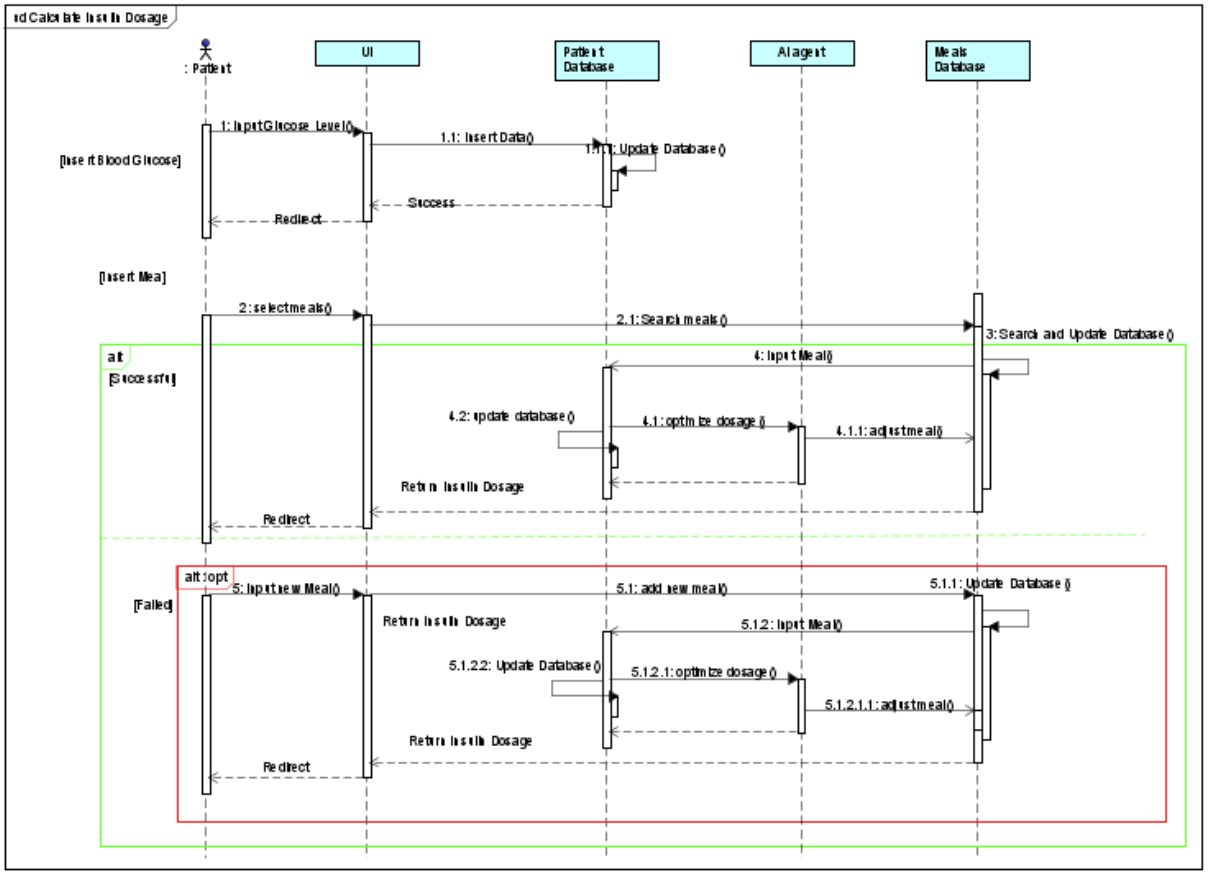


Figure 16: Sequence Diagram for Calculating Insulin Dosage

- The second sequence diagram shows how the doctor searches for the patient's information. This diagram also shows how the doctors will be notified as soon as one of their patients has successfully signed up.

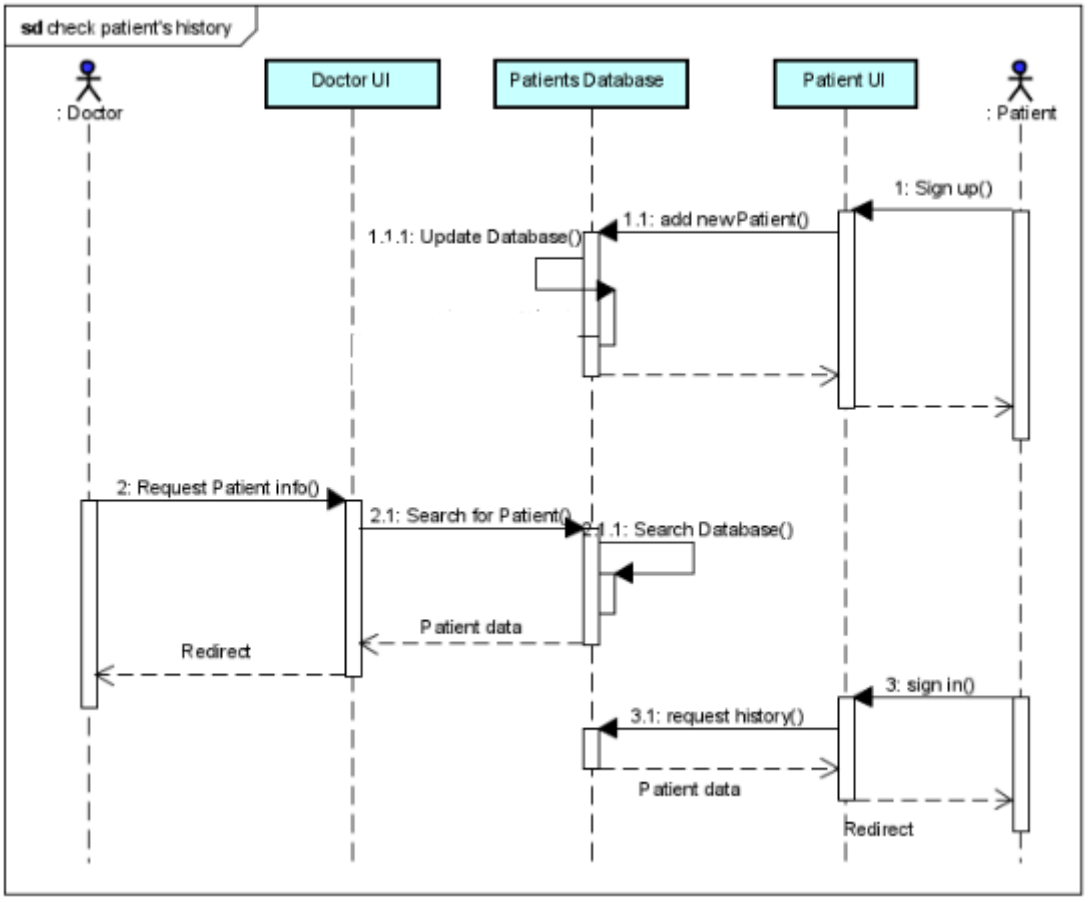


Figure 17: Sequence Diagram for Checking Patients History

Chapter 5. Methodology

A. Software Architecture

The Layered Architecture Diagrams are known to be the most frequently used framework in the software development industry since it's very simple and clearly shows how an application functions as well as its main functionalities. Another name of it is N-Tier Architecture, and it shows a pattern consisting of multiple separate horizontal layers that depend on each other and function together to make up the app.

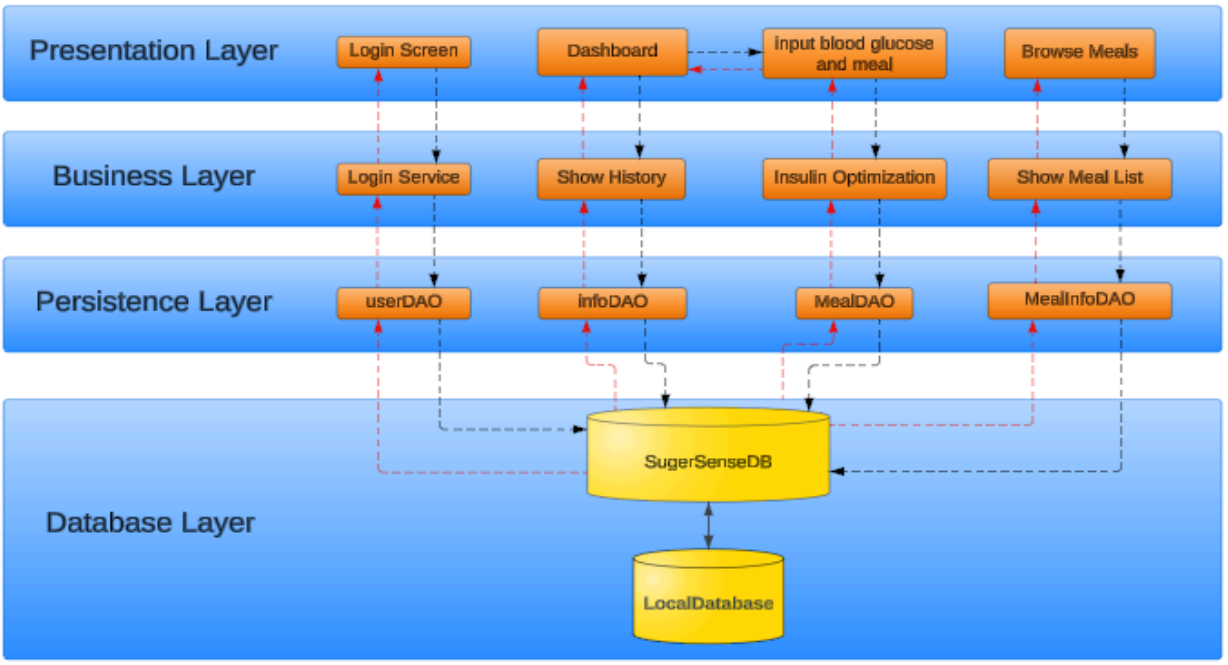


Figure 18: Layered Architecture Diagram for SugarSense

This diagram consists of an arbitrary number of layers selected by the developer or software designer, but most of them typically follow this structure:

1- Presentation Layer: In this layer, we put all the components responsible for user interactions with the application.

2- Business Layer: It consists of the components related to achieving functional requirements.

3- Persistence Layer: It contains all the algorithms and programming components used.

4- Database Layer: Here is where data is handled and put into databases.

Just like any other framework out there, it features some advantages as well as disadvantages. They are as follows:

- Advantages:
  - Simple and easy to implement framework.
  - Easy testing since we can test every layer individually.
  - Loosely coupled since each layer is separate from the other.
  - Typically, cheaper than most other frameworks.
- Disadvantages:
  - Low scalability since framework limits growth.
  - Can be difficult to maintain, changing one layer will affect all the other layers.
  - Parallel Processing is not possible in this framework.

We have chosen this framework since it is simple to learn and implement, since it's typically advisable to use the layered architecture due to it supporting small applications as well as projects that need to be built quickly. It is also beneficial whenever the developers don't have much prior knowledge on software architectures or are indecisive on which one to use.

B. Implementation

- ### Frontend

Flutter and Dart will be the primary technologies used to implement the frontend of the application. Flutter's rich widget library, coupled with Dart's expressive and flexible programming language, provided an efficient and streamlined approach to building the frontend of the application.

One of the significant advantages of using Flutter and Dart is their ability to build a single codebase and use it for several platforms such as the web, desktop and mobile. This helped in reducing development time and effort, and ensured that the app's user experience and functionality remain consistent across different platforms.

The Flutter framework's hot-reload feature, coupled with Dart's Ahead-of-Time (AOT) and Just In Time (JIT) compilation, made the development process even more efficient. These features will allow us to make changes to the app's frontend code and instantly see the updates without needing to rebuild the entire application.

Code Snippet examples:

1. **Graph concept:**
    - Bottom Tiles widget:

A widget that specifies the UI of the x-axis and the time values that will be displayed depending on the chosen time range.

```
Widget bottomTitleWidgets(double value, TitleMeta meta) {
  const style = TextStyle(
    fontWeight: FontWeight.w500,
    fontSize: 13,
    color: Color.fromARGB(255, 178, 178, 178),
  ); // TextStyle
  Widget text;
  if (today) {
    switch (value.toInt()) {···
  } else if (monthly) {
    switch (value.toInt()) {···
  } else {
    if (value >= 1 && value <= 92) {···
    } else {···
    }
  }
  return SideTitleWidget(
    axisSide: meta.axisSide,
    child: text,
  );
}
```

Figure 19: Graph implementation for the bottom tiles

- Left Tiles widget:

A widget that specifies the UI of the y-axis values whether it be glucose levels, bolus levels or carbohydrates levels along with their usual ranges of each.

```
Widget leftTitleWidgets(double value, TitleMeta meta) {
  const style = TextStyle(
    fontWeight: FontWeight.w500,
    fontSize: 13,
    color: Color.fromARGB(255, 178, 178, 178),
  ); // TextStyle
  String text;
  if (b) {
    switch (value.toInt()) {···
  } else if (g) {
    switch (value.toInt()) {···
  } else {
    switch (value.toInt()) {···
  }

  return Text(text, style: style, textAlign: TextAlign.left);
}
```

Figure 20: Graph implementation for the left tiles

- Graph data points:

A widget that specifies the data points UI view along with the graph grids shapes.

```
LineChartData mainData() {
  return LineChartData(
    clipData: const FlClipData(
      bottom: true, // Clip the bottom side
      top: true, // Clip the top side
      left: true, // Clip the left side
      right: true, // Clip the right side
    ), // FlClipData
    gridData: FlGridData(
      show: true,
      drawVerticalLine: true,
      horizontalInterval: 1,
      verticalInterval: b
          ? 1
          : g
              ? 10
              : 5,
      getDrawingHorizontalLine: (value) {
        //hori grids color
        return const FlLine(
          color: Color.fromARGB(103, 162, 162, 162),
          strokeWidth: 1,
          dashArray: [1, 10],
        ); // FlLine
      },
      getDrawingVerticalLine: (value) {
        //ver grids color
        return const FlLine(
          color: Color.fromARGB(103, 162, 162, 162),
          strokeWidth: 0.5,
          dashArray: [2, 11],
        ); // FlLine
      },
    ), // FlGridData
    titlesData: FlTitlesData(
      show: true,
      rightTitles: const AxisTitles(
        sideTitles: SideTitles(showTitles: false),
      ), // AxisTitles
      topTitles: const AxisTitles(
        sideTitles: SideTitles(showTitles: false),
      ), // AxisTitles
      bottomTitles: AxisTitles(
        sideTitles: SideTitles(
          showTitles: true,
          reservedSize: 25,
          interval: yearly ? 1 : 2,
          getTitlesWidget: bottomTitleWidgets,
        ), // SideTitles
      ), // AxisTitles
```

Figure 21: Graph implementation for the data points (A)

Figure 22: Graph implementation for the data points (B)

Figure 23: Graph implementation for the data points (C)

2. Meals List:

- Meal class:

The class will specify each meal in the database as a Meal with its own attributes.

```dart
class Meal {
  final String name;
  final String imageUrl;
  final int id;
  final double carbohydrates;
  final double quantity;
  final int unit;
  final List<eIngredient> ingredients;

  Meal({
    required this.name,
    required this.imageUrl,
    required this.id,
    required this.carbohydrates,
    required this.quantity,
    required this.unit,
    required this.ingredients,
  });
  @override
  String toString() {
    return 'Meal{name: $name, imageUrl: $imageUrl, id: $id, carbodydrates: $carbohydrates, quantity: $quantity, unit: $unit}';
  }

  Map<String, dynamic> toMap() {
    return {
      'name': name,
      'imageUrl': imageUrl,
      'id': id,
      'carbohydrates': carbohydrates,
      'quantity': quantity,
      'unit': unit,
    };
  }
}
```

Figure 24: Implementation of the Meal class

- Meal Box:

In order to display each meal as a card a MealBox class was made that extends a stateful widget which helps in redrawing on the screen with different data or in a different state, in our case different meals.

```dart
class MealBox extends StatefulWidget {
  final Meal meal;
  final int ind;
  const MealBox({super.key, required this.meal, required this.ind});
  @override
  _MealBoxState createState() => _MealBoxState();
}
```

Figure 25: Implementation of the Meal Box widget

- _MealBoxState:

Specifies the UI view of each meal box and how it will look and contain.

```
class _MealBoxState extends State<MealBox> {
  void refresh() {
    setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () async {
        final TextEditingController quantityController =
            TextEditingController();
        await showDialog( ···
      },
      child: Stack(
        clipBehavior: Clip.none,
        children: [
          Card(
            shape: RoundedRectangleBorder(
              borderRadius:
                  BorderRadius.circular(5), // Change this value as needed
            ), // RoundedRectangleBorder
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                SizedBox( // SizedBox ···
                Column( // Column ···
                //meal name
              ],
            ), // Column
          ), // Card
          FutureBuilder<bool>(
            future: db.searchMealForCat(widget.meal.id, 'mymeal'),
            builder: (BuildContext context, AsyncSnapshot<bool> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const CircularProgressIndicator(); // Show a loading spinner while waiting for the data
              } else if (snapshot.hasError) {
                return Text(
                    'Error: ${snapshot.error}'); // Show an error message if something goes wrong // Text
              } else {
                // Use snapshot.data in a condition
                if (snapshot.data == true) {
                  return Positioned( // Positioned ···
                } else {
                  return Container(); // Return an empty container if the Future returned false or null
                }
              }
            },
          ), // FutureBuilder
        ],
      ), // Stack
    ); // GestureDetector
  }
}
```

Figure 26: Implementation of the Meal Box state

3. UserInfo questions:

```
final controller = PageController();
```

In order to navigate between questions when the user is creating their account, we used the PageController that allows us to navigate between pages in the same field. In that aspect whenever the user clicks next, it checks if input was valid or next then navigates them to the next question.

```
onPressed: () {
  if (currentPage == 0) {
    if (formKeys[currentPage].currentState!.validate()) {
      setState(() {
        updatefirstAnswer();
        if ((answers[0] as List)[0] != 0.0) {
          updatefirstAnswer();
          currentPage = controller.page!.round() + 1;
          if (currentPage < questions.length) {
            controller.nextPage(
              duration: const Duration(milliseconds: 500),
              curve: Curves.ease,
            );
          }
        }
      });
    }
  } else if (formKeys[currentPage].currentState!.validate()) {
    setState(() {
      if (answers[currentPage] != 0.0 ||
          answers[currentPage] != 0) {
        currentPage = controller.page!.round() + 1;
        if (currentPage < questions.length) {
          controller.nextPage(
            duration: const Duration(milliseconds: 500),
            curve: Curves.ease,
          );
        }
      }
    });
  }
  // ignore: avoid_print
  print(answers);
},
```

Figure 27: Implementation of the answers update of patients information

We also specified the UI view for each question page.

Figure 28: Implementation of the UI view of every question

- ## Backend

We chose to implement using the SQLite database engine in order to ensure that the storage and retrieval of data operate at peak performance. Using the SQFlite package, the database upholds an efficient authenticating service that enables users to safely store and easily retrieve their personal data, as well as access a variety of well-organized structured data.

SQLite is an embedded, server-less relational database management system. It's an open source software that doesn't require any separate server process or system to operate and can run on all platforms: Linux, macOS, windows, Android, IOS… and since it's a file-based database, it doesn't rely on network connectivity and is lightweight with a small memory footprint; thus performs well even on low-resource devices when an entire database can be contained within a single file, making it easy to distribute and deploy within systems. This simplicity and independent nature make SQLite suitable for mobile applications. SQLite is also flexible and reliable, it updates user's data

continuously and allows multiple processes to operate simultaneously without interfering with each other.

Although we're having some issues with SQLite. Mainly, the limited the capacity which is at most 2GB and may require frequent data management tasks such as archiving or purging old records. Also we may face a problem with concurrency since SQLite supports an unlimited number of simultaneous readers, but it will only allow one writer at time; Thus we'll need to manage transactions and reader-writer lock appropriately in these situations.

As we've mentioned in the database diagram section, we designed an EER diagram with multiple entities and relations connecting them with each other. Mainly, when patients register, a record of their information will be saved instantly in the database and can login into their accounts. From then on, they are connected to the meal database, articles feeds, or their own local database for example and can perform all operations (pick favorite article, search meals, customize meals…). And since the database is set up locally, the users can access all the information about their daily intakes and meal information without a network connection.

As for the remote Server database, we set it up through the "Microsoft Azure SQL Database" which is a managed cloud database. Using the Azure database helped us overcome a lot of our challenges. It allows users to easily sign up and login into their accounts, sync the public meal database into their devices, but most importantly it serves as a backup to synchronize all the patient's information and daily intakes, so that the doctor can have access to.

- ## Packages

We used at least 20 packages during the SugarSense development process to optimize our implementation and improve user experience overall. Since Pub is the standard and most popular package manager for Dart and Flutter development, we included it in our application. Pub helps with managing dependencies smoothly and effectively, and therefore makes it simple to add additional libraries and packages to our codebase.

The following packages were carefully chosen based on their suitability for SugarSense's particular requirements as well as their capabilities. By using these packages, we were able to achieve our non-functional objectives for quality and usability while also producing a polished final product that will save time and effort during development. These packages included are:

Dart packages:

- **sqlite/sqflite.dart:** This package enables an efficient and simple backend implementation. And it facilitated the storage and retrieval of user's data.
- **logging:** package that will allow us to log all the user's activity and will allow us to see if the application is behaving as intended.
- **http:** allows us to send http requests to any api endpoint using dart source code.
- **Shared Preferences:** allow us to store single variables and we can then retrieve those values even after restarting the application.
- **Pie_chart:** a package that allows the use of already made pie charts in flutter.
- **calendar_view:** a package that contains the ability to display a calendar and gives the rights to add and delete from it.
- **fl_chart:** allows us to use ready existing chart templates.

Packages for python:

- **Serp Api for Google Search:** This package allows us to perform multiple google searches about diabetic articles and guidelines.
- Smtplib: Allows us to send emails using our official gmail.
- fastApi: gives us the ability to develop and run an Api endpoint with custom requests.
- hashlib: is a package that allows us to easily hash the password inputted by the user.
- pyodbc: will allow us to connect to the database to be able to perform CRUD operations.

- **APIs**

To allow a connection between the database and the application. We require a middleware API that formats all the data in the correct way, as well as perform all the validations needed for data processing. Therefore, we have created such an API, using the fastApi package, which can perform all kinds of operations on the database such as: insert, delete, update and select.

```
########################################
######|Database Connection String|#########
########################################
############################################################
server = 'sugarsense.database.windows.net'
database = 'sugarsensedb'
username = 'sugaradmin'
password = 'SUG@Rs!!7891'

if platform.system() == "Windows":
    driver = '{ODBC Driver 17 for SQL Server}'
else:
    driver= '{ODBC Driver 18 for SQL Server}'

connection_string = f'DRIVER={driver};SERVER={server};DATABASE={database};UID={username};PWD={password};'
cnxn = pyodbc.connect(connection_string)
cursor = cnxn.cursor()
############################################################
```

Figure 29: FastAPI connection to the server

To be able to connect to the database in order to do any CRUD operation we need, we are going to have a connection string that consists of multiple parameters. Mainly, the driver of the current device, the name of the server we want to connect to, the name of the database in that server, and the password that was set up by the database admin. After we format them in this way, we can call the connect() function with the connections string as the parameter.

```python
@app.post("/token", response_model=Token)
async def login_for_access_token(form_data: UserForm = Body(...)):
    user = User(username=form_data.username, password=form_data.password)
    try:
        # Query the database for the user
        cursor.execute("SELECT userPassword,userID FROM Users WHERE CAST(userName AS NVARCHAR(MAX)) = ?", (user.username,))
        rowUsername = cursor.fetchone()
        cursor.execute("SELECT userPassword,userID FROM Users WHERE CAST(email AS NVARCHAR(MAX)) = ?",(user.username,))
        rowEmail = cursor.fetchone()

        # If the user doesn't exist or the password is incorrect, return a 401 Unauthorized response
        hashed_password = hashlib.md5(user.password.encode()).hexdigest()

        if (rowUsername is None or hashed_password != rowUsername[0]) and (rowEmail is None or hashed_password != rowEmail[0]):
            if(rowUsername is None and rowEmail is None):
                raise HTTPException(status_code=401, detail="Invalid email or username")
            else:
                raise HTTPException(status_code=401, detail="Incorrect password")

        # If the email and password are correct, return a 200 OK response
        uid = getUserById(user.username)
        cursor.execute("SELECT * FROM Patients WHERE patientID = ?", uid)
        pid = cursor.fetchone()
        subs=await getSubscription(uid)
        print(subs)
        if(subs == 0):
            raise HTTPException(status_code=400, detail="This user is not subscribed")
        elif(pid is None):
            raise HTTPException(status_code=402, detail="This user is not a patient")
        else:
            access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
            access_token = create_access_token(
                data={"sub": user.username}, expires_delta=access_token_expires
            )
            return {"message": "Authenticated successfully", "ID": uid,"access_token": access_token, "token_type": "bearer"}
    except HTTPException as e:
        raise e
    except Exception as e:
        raise HTTPException(status_code=403, detail=str(e))
```

Figure 30: Authentication function for the login

We opted to use OAuth as our method of authentication of our users. When a patient logs in to their account for the first time, we will generate and give them an authentication token which identifies this specific user. After every login we check the validity of his token, and if it is then we let the user into their account.

```python
SECRET_KEY = "bfedd62227958245913f74acc9ed79a86b3e1df1863e428a5e4728bfe0986315"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 42800      You, 1 second ago • Uncommitted changes

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

def create_access_token(data: dict, expires_delta: timedelta):
    to_encode = data.copy()
    expire = datetime.utcnow() + expires_delta
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

Figure 31: Create Token implementation

In order to create the access token, we made a function that takes a dict and an expiry date, and uses both of those parameters as well as encodes them, to obtain a valid access token and return it.

```python
def get_current_user(token: str = Depends(oauth2_scheme)):
    print(f"Token: {token}")
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        print(f"Payload: {payload}")
        username: str = payload.get("sub")
        print(username)
        if username is None:
            raise credentials_exception
        return username
    except JWTError:
        raise credentials_exception
```

Figure 32: Token implementation

To check who the current user is from his access token, we simply decode the token and we check if the username field exists. If it does then we return it, and if it is not then we are going to raise an exception.

To further our features a bit more, we used some additional APIs:

Serp API: Our application will have a feature that displays articles related to diabetes. And to achieve that, we used Serp api that searches for whatever parameters we put on google and returns the results in json format.

```python
apiKey = 'eb03e6986533c14f2abb8e891cd2438a7f519c1e05935951a61141e23f9fd3b0'
def get_results(search):
    global results
    results = search.get_dict()

@app.get("/News/{query}")
async def get_news(query: str):
    params = {
        "q": query,
        "hl": "en",
        "gl": "us",
        "google_domain": "google.com",
        "api_key": apiKey
    }
    search = GoogleSearch(params)

    for i in range(10):  # Retry up to 3 times
        thread = threading.Thread(target=get_results, args=(search,))
        thread.start()
        thread.join(10)

        if thread.is_alive():
            time.sleep(5)  # Wait for 5 seconds before retrying
        else:

            return results["organic_results"]
    return {"error": "Failed to get results after 3 attempts"}
```

Figure 33: Serp API for article fetching

For the integration of the news API, we created a route that takes a parameter which we will use as a google search. And Serp API will then return the result of the query we input in json format, so that we can display it easily.

- ## Deployment and delivery

For deployment and delivery, we will submit documentation that contains all the pages of SugarSense, along with a detailed description of each page and its feature.

The documentation will look something like this:

1) Launch Page

From the moment users open the app, they are greeted by a welcoming startup screen. The purpose of the launch page is to create a positive first impression, set the tone for the application's user interface, and establish a sense of anticipation for the user.

2) Sign in/Sign up Pages

Upon passing the captivating launch page, users are warmly welcomed with an inviting sign in/sign up interface that offers several options for

account creation. This page has been thoughtfully designed to be user-friendly and intuitive.

- ➢ Signup: Users can sign up for a new account by entering their full name, email, doctor ID and creating a new password.
- ➢ Sign in: For users who already have an account, the login process is straightforward, and they can easily reset their password if needed.

3) Subscription Page

The subscription page of our application SugarSense serves as a pivotal component, enabling users to access application features and services by subscribing to various subscription plans. This page provides a clear and user-friendly interface that guides users through the subscription process. Our subscription offerings include three distinct plans that cater to different user preferences:

- ➢ Standard Plan: Priced at $1 per month, the Standard Plan provides users with access to the application's features and services on a monthly basis.
- ➢ Premium Plan: For those seeking long-term access, the Premium Plan offers a lifetime subscription at a one-time payment of $80. This plan grants users unrestricted and perpetual use of all features.
- ➢ Special Plan (Under 22): We also offer a special plan exclusively designed for users who are under 22 years old. Eligible users have the opportunity to apply for a free subscription, subject to meeting specific conditions.

Users have the flexibility to begin their subscription journey by opting for the Standard Plan and later upgrade to the Premium Plan, unlocking the benefits of lifetime access to all features.

4) Terms & conditions page

The terms & conditions page of our application serves as a transparent and comprehensive outline of rules, guidelines, and legal terms that govern the use of our application. This page provides users with important information regarding their rights, responsibilities, and obligations while using our services.

This page covers various aspects, including:

- ➢ Our services: It provides a brief description of the services offered by our application, highlighting their purpose and functionality.

- ➢ Acceptance of Terms: It states that by accessing or using the application, users agree to abide by the provided terms and conditions.
- ➢ User License: Outlines the responsibilities and expectations of users, such as complying with applicable laws, maintaining the confidentiality of their account information, and using the application responsibly.
- ➢ Privacy Policy: It explains how data is collected, stored, and used, emphasizing our commitment to safeguarding user privacy and complying with relevant data protection regulations.
- ➢ Usage restrictions: It depicts any restrictions or limitations on the use of the application, including prohibited activities, unauthorized access attempts, or misuse of the platform.
- ➢ Termination and Suspension: It outlines the circumstances under which user accounts may be terminated or suspended, such as violation of the terms or fraudulent activities.
- ➢ Governing Law and Jurisdiction: It specifies the applicable laws and jurisdiction that govern the terms and conditions and any potential disputes that may arise.

Users are required to review and accept the terms and conditions before using our application.

5) Set Up Pages:

The setup pages of our application are carefully designed to offer users a user-friendly and intuitive interface, guiding them through the initial configuration process. These pages are dedicated to gathering important information from users, such as their carbohydrate ratio, insulin sensitivity, target glucose level, and preferences regarding the data accessible to their healthcare provider. This information is crucial for creating a comprehensive user account that aligns with their individual requirements.

As users progress through the setup pages, they are prompted to provide specific details that are essential for personalizing their data within the application. These details include:

a. Carbohydrates Ratio: Users are asked to input their individual carbohydrates ratio, which helps the application make accurate insulin dosage recommendations based on their carbohydrate intake.

b. Insulin Sensitivity: Users should enter their insulin sensitivity information, allowing the application to calculate and suggest appropriate insulin doses based on their individual response to insulin.
c. Target Glucose Level: Users are given the opportunity to set their target glucose level, enabling the application to provide personalized recommendations and alerts to help them maintain optimal blood glucose control.
d. Data Sharing Preferences: Users are provided with options to choose what data they would like to share with their healthcare provider, ensuring a collaborative approach to diabetes management. This may include sharing glucose readings, insulin doses, and other relevant information to facilitate effective communication and decision-making.

By gathering these specific details, the setup pages enable users to establish a highly customized and accurate profile within the application. This ensures that the provided recommendations and insights are tailored to their unique diabetes management needs, empowering them to make informed decisions and achieve better control over their health.

6) Dashboard Page:

SugarSense dashboard page plays a crucial role in our application by providing users with a centralized and visually informative interface that allows them to monitor, analyze, and manage key aspects of their experience within the application.

The applications' dashboard consists of:

a. Data visualization: The dashboard presents relevant data and information, such as bolus levels, glucose levels, and carbohydrates, in a visually appealing and easy-to-understand manner. It utilizes graphs with colorful visual elements to condense complex data sets into meaningful and actionable insights that can be scaled up to request. This visual representation helps users quickly interpret and analyze the data, enabling them to make informed decisions and take appropriate actions based on the information presented.
b. Input and Data Tracking: As users enter data through the input forms, the dashboard updates in real-time, reflecting the latest information entered. This ensures that users have an up to date view of their data.

c. Historical Data Tracking: The dashboard maintains a historical record of the entered data, enabling users to monitor and review their health journey with detailed tracking of their data, ensuring a comprehensive understanding of their progress over time.

7) Settings Page:

The settings page in SugarSense serves as a central hub where users customize and fine-tune various aspects of their insulin therapy and overall user experience. It provides a comprehensive range of configurable options and preferences to ensure that the application aligns with each user's unique needs.

Here's what the settings page provides:

a. Carbohydrate Ratio and Insulin Sensitivity: The settings page enables users to customize their carbohydrate ratio and insulin sensitivity settings. Users can enter their specific values based on recommendations from their healthcare provider.
b. Glucose Target: Users can set their target glucose levels on the settings page. This allows the application to provide guidance and notifications based on individualized goals.
c. Data Sharing and Privacy: Users can choose what data they want to share with their healthcare provider, such as glucose readings, insulin doses, and meals eaten. Privacy preferences can be adjusted to ensure the confidentiality and security of personal health data.

8) Input page:

The input page is the heart of the SugarSense. It's the key component that allows users to conveniently and accurately input their insulin related data and relevant information. This page consists of three parts:

a. **Glucose Input:** The input page provides a dedicated section for users to enter their glucose readings. Users can input their blood glucose levels, which is manually measured using a glucometer.
b. **Carbohydrates Intake:** A dedicated section on the input page enables users to record their carbohydrate intake. By clicking on the add meal function, users will be redirected to the meals page where they can simply add and choose their meal by either creating their own or using our database then the application will add all carbohydrates inputs together.

- Meal Creation: SugarSense provides a user-friendly and flexible meal creation feature. With this feature, users can effortlessly select ingredients, specify their quantities, and assign personalized names to their created meals.
- Database meals: SugarSense database contains an extensive collection of predefined meals that users can choose from. This database offers users an extensive collection of meal options to choose from, catering to a diverse range of dietary preferences and requirements.
    - Wide Variety of Meal Options: The SugarSense Meals Database provides users with a wide variety of meal choices, ranging from breakfast, lunch, dinner, to snack options. Users can explore a diverse range of cuisines, flavors, and dietary styles to suit their preferences.
    - Easy Search Functionality: To simplify meal selection process, SugarSense incorporates a search feature within the meals database. Users can search for specific meals by name or ingredients enabling them to quickly find the meals they want.
    - Regularly Updated:the database is regularly updated to ensure a fresh and expanding selection of meals for users. New meals are added, taking into account user feedback, emerging dietary trends, and nutritional guidelines. This ensures that users always have access to a diverse and up-to-date range of meal options.
    - Portion Size Specification: When users select a meal from the database, they are prompted to specify the portion size they plan to consume. This feature allows for precise insulin calculations and helps users manage their glucose levels effectively.
    - Customization Options: While SugarSense database provides predefined meals, users have the flexibility to customize and adapt these meals to fit their individual preferences and dietary needs. Users can make ingredient substitutions, adjust portion sizes, or modify recipes according to their specific requirements, allowing for a personalized meal experience. Upon saving them, it will be saved as a new meal.

c. **Bolus Calculation**: The bolus calculation feature in SugarSense automates the process of determining the appropriate insulin dosage based on the user's glucose reading and carbohydrate intake. When

the user inputs their glucose reading and total carbohydrates consumed, the app performs a real-time calculation to suggest the bolus insulin dose. The app uses an advanced algorithm to calculate the suggested bolus insulin dose based on the user's glucose reading and carbohydrate intake. The calculation takes into account factors such as the user's insulin sensitivity, target glucose range, and carbohydrate-to-insulin ratio. To promote transparency and allow users to understand the calculation process, the app offers a feature to check the formula details. Users can access information on how the bolus calculation is performed, including the specific variables and factors considered in the algorithm.

9) Articles page:

The Articles Page in our insulin optimization application serves as a valuable resource for newly diagnosed diabetic patients and all users seeking information, insights, and educational content related to diabetes management, insulin use, and overall well-being. The articles featured on the page are written by healthcare professionals, experts, and reputable sources in the field and are carefully selected and curated to ensure accuracy, reliability, and relevance. Users can easily save and access their preferred articles, recipes, or other articles using the add to favorites feature present in the articles page. This page is specifically designed to provide guidance and support to individuals who are suffering from the challenges of diabetes for the first time.

10) Account Page:

The Account page provides users access to their account information, including their name, phone number, and profile picture. Additionally, it offers various functionalities, such as Edit Profile, Favorites, Subscriptions, Terms & Conditions, User Manual, Help & Support, and Log Out option.

- Edit profile: By selecting Edit Profile, users are redirected to another page where they can modify their account photo, username, email, phone number and password.
- Favorites: The favorites display a list of saved and bookmarked articles the user previously added.
- Subscriptions: The Subscriptions page in our application provides users with a clear overview of their current subscription plan and offers them the opportunity to make changes accordingly.

- Terms & Conditions: Upon clicking Terms & Conditions a slider page will show allowing users to read and review the terms and conditions SugarSense provides.
- User Manual: When users click on the User Manual option, they will be automatically redirected to the User Manual page on SugarSenses' website. The user manual page on the website will contain detailed instructions, tutorials, and information about using SugarSense effectively.
- Help & Support: When users click on the Help & Support option, they will be redirected to the Contact Us page on SugarSenses' website, where they will be able to send or enquire for any question they have on the contact us section.
- Logout: Lastly, the Logout option allows users to securely log out of their account.

CS Knowledge and Machine Learning

1. AI optimization

To increase the accuracy of SugarSense bolus calculation we used machine learning, more specifically, supervised learning.

Definitions:

1. Values specific to each patient and are given by the Doctor:
    1. r: Carb ratio
    2. s: Insulin sensitivity
    3. g*: Target glucose level (targetBloodSugar)
2. A meal is a specific food in our database with values associated such as
    1. m: Name (ex. Apple)
    2. c: Carbohydrates (ex. 15)
    3. w: Certainty, which refers to how certain are we that said carbohydrates is accurate for this meal and it's a range between 0 and 1 (ex. 0.5)
3. An entry is a collection of data referring to a specific point of time when the user entered some input and received an insulin calculation dosage. Data stored:

1. g: GlucoseLevel, refers to the glucose level of the user at the time of the entry
2. Date, referring to the time the entry was made
3. HasMeals, which is stores the meals consumed by the user at the time of entry, including the quantity of each
4. InsulinDosage, which is the insulin taken by the user at the time of the entry (the insulinDosage is calculated by the app)

Traditionally, the insulin dosage is calculated by the following formula:

let C be the total carbs in all meals of an entry

dosage = (g - g*)/s + C*r

But the carbs of most meals are always just an estimate or a guess by the patient, which is why we are using reinforced learning to get a better estimate of the carbs in the meals.

```
int calculateDosage(double totalCarbs, double bloodSugar, double carbRatio) {
  double ans = 0;
  if (glucoseUnit_ == 0) bloodSugar = bloodSugar * 18.0156;
  ans += (bloodSugar - targetBloodSugar_) / insulinSensitivity_;
  ans += ((totalCarbs / 15) * carbRatio);
  return ans.round();
}

double calculateTotalCarbs(List<Map> meals) {
  double ans = 0;
  for (Map meal in meals) {
    ans += meal["carbohydrates"] * meal["quantity"];
  }
  return ans;
}
```

Figure 34: calculate insulin function

Algorithm:

Input:

1. Current glucose level (bloodSugar)
2. Previous entry

We address whether the current glucose level is within an acceptable range.

First, we compute:

bloodSugarDiff = bloodSugar - targetBloodSugar

Then if the absolute value of bloodSugarDiff is less than insulinSensitivity then the glucose level is within acceptable range, thus the calculation we did for the previous entry was accurate therefore we increase the certainty of the meals in the previous entry.

First, we get mealCount, the number of distinct meals in the entry

Then we calculate alpha = 1/sqrt(mealCount)

Then for each meal we set its newCertainty = certainty + alpha * (1 – certainty) / 2

We are multiplying by alpha because the more meals we have, the higher the chance of their errors canceling each other out. For example, if an entry had meal1 = 15 carbs and meal2 = 15 carbs it will total to 30. Even if 30 was the true total carbs, that doesn't prove for certain that meal1 has 15 carbs and meal2 has 15 carbs.

Alternatively, if the absolute value of bloodSugarDiff is more than insulinSensitivity then we deduce that our previous calculation was wrong, thus we adjust the carbs for the meals inside the previous entry and decrease their certainty.

First, we compute unaccountedCarbs = (bloodSugarDiff / insulinSensitivity) / carbRatio

But we only take unaccountedCarbs/2 into account because it's safer to increase the carbs gradually rather than risking hypoglycemia (low blood sugar).

Then to distribute the unaccounted carbs over the meals, each meal should have a ratio. We start by computing blame = carbs * (1-certainty) for each meal. Then we compute totalBlame as Σ(blame*quantity) for each meal. The ratio is then blame/totalBlame. This ensures that meals with lower certainty take more of the blame and meals with higher certainty won't be affected as much. We are also factoring the original carbs to distribute the unaccounted carbs proportionally to each meal's original carbs content, maintaining a fair distribution rather than equal allocation.

Then, newCarbs for each meal would be carbs (original carbs) + unaccountedCarbs * ratio.

Lastly, the newCertainty would be certainty - alpha * certainty/5

The code:

```
void updatePrevMeals(double bloodSugar, double carbRatio) async {
  DBHelper dbHelper = DBHelper.instance;
  List<Map> latestEntry = await dbHelper.getLatestEntryId(1);
  int prevEntryId = latestEntry[0]['entryId'];
  List<Map> hasMeals = await dbHelper.getMealsFromEntryID(prevEntryId);
  List<Map> meals = [];
  double bloodSugarDiff = bloodSugar - targetBloodSugar_;
  logger.info("AI working");
  if ((bloodSugarDiff).abs() <= insulinSensitivity_) {
    logger.info("Patient is healthy");
    //patient is healthy certainty factor goes up for all meals
    int mealCount = 0; //number of distinct meals
    for (Map mid in hasMeals) {
      List<Map> mealResponse = await dbHelper.getMealById(mid["id"]);
      Map meal = mealResponse[0];
      meals.add(meal);
      mealCount++;
    }
    double alpha = sqrt(mealCount) / mealCount;
    for (Map meal in meals) {
      double newCertainty = meal["certainty"] +
          alpha *
              (1 - meal["certainty"]) /
              2; //ensuring certainty does not go above 1
      await dbHelper.updateMealById(
          meal["mealId"], meal["carbohydrates"], newCertainty);
    }
```

Figure 35: certainty calculation function (A)

```
} else {
  logger.info("Patient is not healthy");
  //patient is not healthy carbs for each meal go up (or down depending if
  //they have low or high bloodsugar). And certainty goes down
  int mealCount = 0; //number of distinct meals
  double totalBlame = 0;
  for (Map mid in hasMeals) {
    mealCount++;
    List<Map> mealResponse = await dbHelper.getMealById(mid["id"]);
    Map<String, dynamic> meal = Map<String, dynamic>.from(mealResponse[0]);
    meal["quantity"] = mid["quantity"];
    double blame = meal["carbohydrates"] * (1 - meal["certainty"]);
    meal["blame"] = blame;
    totalBlame += blame * meal["quantity"];
    meals.add(meal);
  }
  double alpha = sqrt(mealCount) / mealCount;
  double unaccountedCarbs =
      ((bloodSugarDiff / insulinSensitivity_) / carbRatio) * 15;
  unaccountedCarbs /= 2; //accounting for uncontrollable factors
  logger.info("Unaccounted Carbs: $unaccountedCarbs");
  for (Map meal in meals) {
    double ratio = meal["blame"] / totalBlame;
    double newCarbs = meal["carbohydrates"] + unaccountedCarbs * ratio;
    double newCertainty = meal["certainty"] -
        alpha *
            (meal["certainty"]) /
            5; //ensuring certainty does not go below 0
    await dbHelper.updateMealById(meal["mealId"], newCarbs, newCertainty);
    logger
        .info("updated: ${meal["mealName"]} to $newCarbs and $newCertainty");
  }
}
```

Figure 36: certainty calculation function (B)

In the terminology of reinforcement learning:

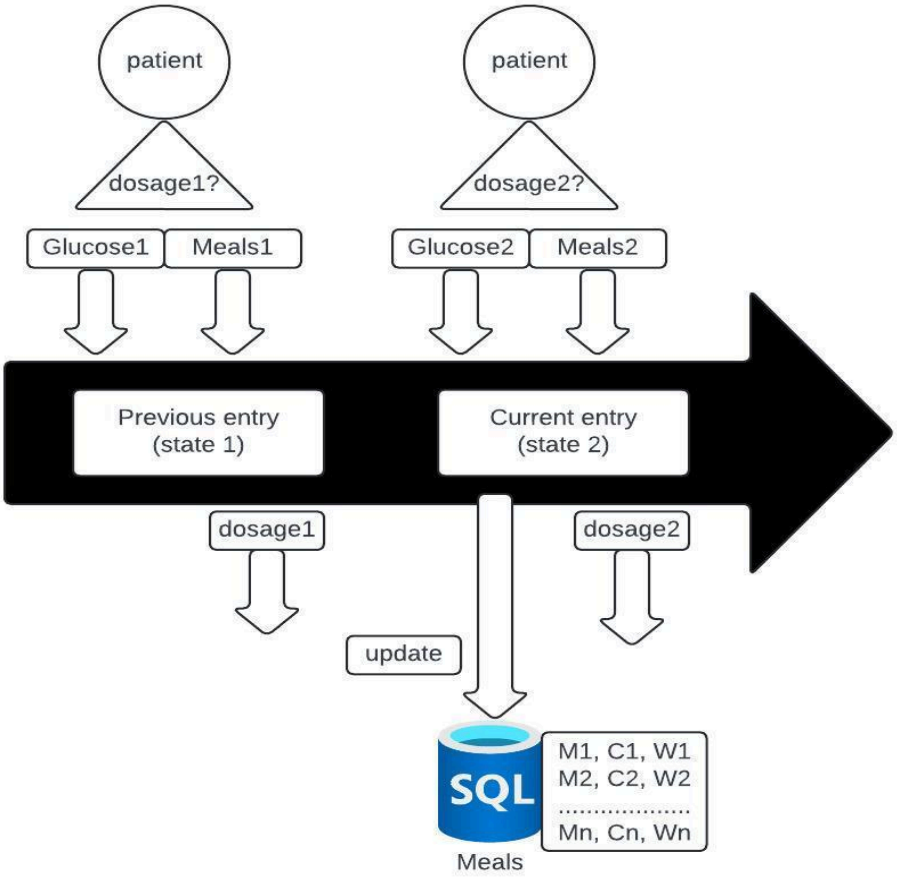


Figure 37: reinforcement learning timeline

The state is the current glucose level of the patient and the current carbs and certainty of the meals in the database.

The input is the meals and current glucose level

The action is calculating the dosage

The policy is:

Δg = target glucose - glucose
s = insulin sensitivity

	Wi↘	Wi↗
Wi	|Δg| > s	|Δg| <= s

	Ci↗	Ci↘	Ci
Ci	|Δg| > s & Δg > 0	|Δg| > s & Δg < 0	|Δg| <= s

Figure 38: reinforcement learning policy

The value function is the summation of w + 0.5(1-w) for all meals

The reward function is:

Healthy = 1

Not healthy = -0.5

2. Database Systems and SQL Systems

After we took the Database Systems (CMPS342) course, we had grasped a deeper understanding of how important and crucial the database functions and role are in any application, systems or websites. This knowledge helped us acquire the skills needed to design and structure our database using SQLite by defining the right

entities and their respective attributes along with the relations that connect them together. And Since we're implementing an SQL based database, we already have a perfect understanding of how SQL queries are generated which has proved to be a significant advantage to us, while developing SugarSense, to facilitate the storage and retrieval of data for users.

As we mentioned, our database systems are a combination of the remote Azure server and the local Sqlite database initialized in every device upon signing up. Both work together in harmony to store and retrieve user's data.

- **Azure SQL database:**

  This database serves as a remote Server that stores all the user's information (username, email, phone number…) and patient's details (carb ratio, insulin sensitivity..) that will be sent when first registering into the application. As we can see in the following code, after checking that the username and email inserted don't already exist in the database, we hash the password and insert a new user. All the requests made for the server are done through the API we set up.

```python
@app.post("/register")
async def registerfunction(user: NewUser):
    try:

        if not checkUsername(user.username):
            raise HTTPException(status_code=401, detail="Username already exists")

        if not checkEmail(user.email):
            raise HTTPException(status_code=401, detail="Email already exists")

        hashed_password = hashlib.md5(user.password.encode()).hexdigest()

        cursor.execute("INSERT INTO Users (firstName, lastName, userName, email, userPassword) VALUES (?, ?, ?, ?, ?)",
                       (user.firstName, user.lastName, user.username, user.email, hashed_password))
        cnxn.commit()
        return {"message": "Registered successfully"}
    except HTTPException as e:
        raise e
    except Exception as e:
        return {"error": str(e)}
```

Figure 39: User registration function

Same steps followed when registering a patient, we take the userID and connect with the creation of a new patient.

```python
@app.post("/regPatient")
async def registerfunction(user: NewPatient):
    print("entered /regPatient")
    try:
        id = getUserById(user.username)
        print(id)
        print(user)
        cursor.execute("INSERT INTO Patients (patientID, doctorCode, insulinSensivity, targetBloodGlucose ,
        +"carbRatio, carbRatio2, carbRatio3, privacy) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
                       (id, user.doctorID, user.insulinSensivity, user.targetBloodGlucose, user.carbRatio1,
                       user.carbRatio2, user.carbRatio3, user.privacy))
        cnxn.commit()
        print("Registered patient successfully")
        return {"message": "Registered patient successfully"}
    except HTTPException as e:
        raise e
    except Exception as e:
        return {"error": str(e)}
##########################################################
```

Figure 40: Patient registration function

The server also stores all the meals' table which were shared by a diabetic center here. It holds around 200 meals certified by the center at most accuracy. Each meal has its name, carbohydrates and unit. When the local database is initialized, all the meals will be synced and saved into the local device database.

```dart
Future<void> syncMeals() async {
  logger.info("Syncing meals...");
  DBHelper dbHelper = DBHelper.instance;
  var response = await http.get(Uri.parse('http://$localhost:8000/meals'));
  if (response.statusCode == 200) {
    // If the server returns a 200 OK response, parse the JSON.
    var meals = jsonDecode(response.body);
    final Database? db = await dbHelper.db;
    for (var meal in meals) {
      await db?.insert(
        'Meals',
        meal,
        conflictAlgorithm: ConflictAlgorithm.replace,
      );
    }
    logger.info("Meals have been synced successfully.");
    // Now you can use meals to insert data into the database
  } else {
    // If the server returns an error response, throw an exception.
    throw Exception('Failed to load meals');
  }
}
```

Figure 41: Meals syncing function

Now that the patient can choose meals and calculate their insulin dosage, we need to sync his entries into the server so that the doctor can have access to them.

```
@app.post("/addNewEntry")
async def addNewEntry(entry: NewEntry):
    print("entered /addNewEntry")
    try:
        print(entry)
        row = cursor.execute("INSERT INTO Entry (patientID, entryID, entryDate, glucoseLevel , insulinDosage,
        +"totalCarbs, unit, hasMeals) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
                             (entry.patientID, entry.entryID, entry.entryDate, entry.glucoseLevel, entry.insulinDosage,
                             entry.totalCarbs, entry.unit, entry.hasMeals))
        print(row)
        cnxn.commit()

        print("inserted entry successfully")
        return {"message": "inserted entry successfully"}
    except HTTPException as e:
        raise e
    except Exception as e:
        print(e)
        return {"error": str(e)}
```

Figure 42: Entries syncing function

If the user wishes to change their password, the operation requires the old and new password to verify the identity of the user before updating it. It compares the hashed values of both and if the condition is verified, the new password is hashed and changed depending on the user's ID.

```
@app.get("/changePassword/{old}/{new}/{id}")
async def changeUsername(old,new,id):
    if(checkPassword(old,id)):
        hashed_passwordOld = hashlib.md5(old.encode()).hexdigest()
        hashed_passwordNew = hashlib.md5(new.encode()).hexdigest()
        cursor.execute("UPDATE Users SET userPassword = ? where userID  = ?",(hashed_passwordNew,id))
        cnxn.commit()
        if(hashed_passwordOld == hashed_passwordNew):
            raise HTTPException(status_code=400, detail="new password can't be the same as the old one")
        if cursor.rowcount > 0:
            return True
        else:
            raise HTTPException(status_code=500, detail="couldn't change password")
    else:
        raise HTTPException(status_code=401, detail="incorrect password")
```

Figure 43: Update password function

In conclusion, the requests for the server are preserved for authentication, registration, syncing meals or entries and editing and updating any personal information for the user.

- **Sqlite local database:**

  Now that the registration and authentication were successful the local database is initialized in the device.

```
initialDb() async {
  String DbPath = await getDatabasesPath();
  path = join(DbPath, 'SugarSense.db');
  Database database = await openDatabase(path,
      onCreate: _onCreate, version: 38, onUpgrade: _onUpgrade);
  logger.info("Local Database has been initialized.");
  return database;
}
```

Figure 44: Initialization of the Sqlite database

```
_onCreate(Database db, int version) async {
  await db.execute('''
CREATE TABLE "Entry"(
  entryId INTEGER PRIMARY KEY AUTOINCREMENT,
  glucoseLevel REAL NOT NULL,
  insulinDosage INTEGER NULL,
  entryDate TEXT NOT NULL,
  unit INTEGER NULL,
  totalCarbs REAL NOT NULL,
  hasMeals TEXT NOT NULL,
  sync INTEGER NOT NULL DEFAULT 0
);
''');
  await db.execute('''
  CREATE TABLE "Meals"(
    mealId INTEGER NOT NULL PRIMARY KEY,
    mealName TEXT NOT NULL,
    mealPicture TEXT NULL,
    unit INTEGER NOT NULL,
    carbohydrates REAL NOT NULL,
    tags TEXT NULL,
    frequency INTEGER NOT NULL,
    certainty REAL NOT NULL
  );
''');
  await db.execute('''
```

Figure 45: Creation of the Sqlite database

From this moment forward, any operation is done through the local database. Any adding or retrieval of data is done locally. Some examples of queries using SQLite on Flutter, you can find the rest in (Appendix C):

If the patient wants to add a new meal into the database, it uses the following query:

```
createMeal(String mealName, String picture, List<Map> childMeals,
    List<String> categories, double carbohydrates) async {
  double totalCarbs = carbohydrates;

  if (picture == "") {
    picture = "meals/All.png";
  }

  String tags = "";
  categories.forEach((element) {
    tags += "$element, ";
  });
  tags += "myMeals";
  int newMealId = await createNewMeal(mealName, totalCarbs, 7, picture, tags);
  if (newMealId != -1) {
    if (childMeals.isNotEmpty) {
      childMeals.forEach((element) {
        createMealComposition(newMealId, element['id'], element['unit'] ?? 7,
            element['quantity']);
      });
    }
    logger.info("Meal has been edited successfully with id $newMealId.");
    return newMealId;
  } else {
    logger.info("Error meal wasn't edited.");
    return -1;
  }
}
```

Figure 46: Create meal function (A)

```
Future<int> createNewMeal(
  String name,
  double carbs,
  int unit,
  String picture,
  String tags,
) async {
  double certainty = 0.0;
  int frequency = 0;
  Database? mydb = await db;
  int mealId = await getMealIdByName(name);

  int nextId = await getNextMealId();

  if (mealId < 0) {
    int response = await mydb!.rawInsert(
        '''INSERT INTO Meals(mealId,mealName,carbohydrates,unit,mealPicture,tags,certainty,frequency)
        VALUES($nextId,"$name",$carbs,$unit,"$picture","$tags",$certainty,$frequency);''');
    logger.info("New meal $name has been created successfully.");

    return nextId;
  } else {
    logger.info("Couldn't create meal. Meal already exists.");
    return -1;
  }
}
```

Figure 47: Create meal function (B)

When the user needs to calculate their insulin dosage and insert the entry into the local database:

```
generateNewEntry(double glucose, int insulin, String date, int unit,
    double totalCarbs, String hasMeals) async {
  Database? mydb = await db;
  int entryId = await mydb!.rawInsert('''
INSERT INTO Entry (glucoseLevel, insulinDosage, entryDate, unit,totalCarbs,hasMeals)
VALUES($glucose,$insulin,'$date',$unit,$totalCarbs,'$hasMeals');
''');
  if (entryId > 0) {
    var latestEntry = await getLatestEntryId(1);
    return latestEntry[0]['entryId'];
  } else {
    return -1;
  }
}
```

Figure 48: Adding new entry to database

To retrieve all entries from the database and display them for the user (daily, weekly and monthly):

```
getEntriesDaily() async {
  Database? mydb = await db;
  var res = await mydb!.rawQuery('''
  SELECT * FROM Entry WHERE substr(entryDate, 1, 10) = date('now')
''');
  return res;
}

getEntriesWeekly() async {
  Database? mydb = await db;
  var res = await mydb!.rawQuery('''
  SELECT * FROM Entry WHERE substr(entryDate, 1, 10) BETWEEN date('now', '-7 day') AND date('now')
''');
  return res;
}

getEntriesMonthly() async {
  Database? mydb = await db;
  var res = await mydb!.rawQuery('''
  SELECT * FROM Entry WHERE substr(entryDate, 1, 10) BETWEEN date('now', '-1 month') AND date('now')
''');
  return res;
}
```

Figure 49: Retrieving the entries from the database

To be able to master implementation on SQLite we watched some videos on expert's YouTube channels Snippet Coder, the Pub.dev site, and code snippets Ericgitic from GitHub that covered all the main principles and basics for the implementation; which included models, schemas, key-word and functions. Learning these ideas enabled us to recognize the benefits and features of SQLite databases. Since SQLite is a server-less software and file-based database, that makes it more flexible and reliable in mobile applications.

Additionally, to offer offline access and faster response, we opted to use the shared preferences package. This package helps store user's small data such as username, email, crab ratio, insulin sensitivity… that are used in profile and settings. The advantages of using shared preferences concept is that it is secure no one can have access to them, flexible they can be accessed anywhere in the application, they have persistence storage, the values are saved even if the application is closed or the device rebooted. They only get reset if the user deletes his account or the application from his device.

lib > Database > variables.dart > savePreferences

```dart
import 'dart:convert';

import 'package:shared_preferences/shared_preferences.dart';
import 'package:http/http.dart' as http;
import 'package:sugar_sense/main.dart';

Future<void> saveStringList(List<String> stringList) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setStringList('deleteEntryList', stringList);
}

Future<void> savePreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();

  await prefs.setInt('targetBloodSugar', targetBloodSugar_);
  await prefs.setInt('insulinSensitivity', insulinSensitivity_);
  await prefs.setDouble('carbRatio', carbRatio_);
  await prefs.setDouble('carbRatio2', carbRatio_2);
  await prefs.setDouble('carbRatio3', carbRatio_3);
  await prefs.setString('username', username_);
  await prefs.setString('firstName', firstName_);
  await prefs.setString('lastName', lastName_);
  await prefs.setString('email', email_);
  await prefs.setBool('signedIn', signedIn_);
  await prefs.setString('doctorCode', doctorCode_);
  await prefs.setString('phoneNumber', phoneNumber_);
  await prefs.setString('profilePicture', profilePicture_);
```

Figure 50: Implementation of the shared preferences

The shared preference defined above are considered like any other variable. They can be accessed anywhere in the pages and implemented into the application. The 2 main operations we needed are the set…() which allows us to save the updated values of the shared preferences and the get…() which help us load the shared preferences upon launching the application. Here are a few examples of the functions we used:

```
Future<void> saveProfile() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setString('profilePicture', profilePicture_);
  await prefs.setString('username', username_);
  await prefs.setString('email', email_);
  await prefs.setString('phoneNumber', phoneNumber_);
}

Future<void> saveP() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setString('profilePicture', profilePicture_);
}

Future<void> loadPreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  syncedInsulin_ = prefs.getBool('syncedInsulin') ?? true;
  syncedRatios_ = prefs.getBool('syncedRatios') ?? true;
  syncedTarget_ = prefs.getBool('syncedTarget') ?? true;
  syncedPrivacy_ = prefs.getBool('syncedPrivacy') ?? true;
  targetBloodSugar_ = prefs.getInt('targetBloodSugar') ?? 100;
```

Figure 51: Implementation of the shared preferences operations

3. Computer Security

Our patients' privacy and security are the most important to us; Thus, we guarantee that their passwords are encrypted safely using the best hashing algorithms used by most of the top companies. Md5 (message digest method 5) will be the hashing algorithm that we used to encrypt the passwords, it is an iterative algorithm that applies a pseudorandom function. It is resistant to dictionary attacks as well as rainbow table attacks, especially after specifying required lengths of the passwords.

A string of any length can be encoded into a 128-bit fingerprint to generate an MD5 hash. The 128-bit hash value produced by the MD5 method will always be the same when encoding the same string. When storing passwords, credit card numbers, or other sensitive data in databases like MySQL, MD5 hashes are frequently employed with shorter strings. With the help of this program, you may quickly and simply encode an MD5 hash from a short string of up to 256 characters.

Additionally, we have tested and made sure that our program will not be affected by any type of sql injections. That is because of our

implementation of multiple steps to mitigate the threat, which includes parameterizing the inputs so that we take the input as a string and not concatenate it directly to the query, making all SQL injection attempts futile.

C. Website Implementation (SugarSense Doctor website)

We designed a simple and easy to navigate website to help doctors monitor their patients' data that uses SugarSense as their monitoring application. The website serves as a way of connecting the patients data with their doctor to always be up to date with the information. While also facilitating hassle-free appointment scheduling.

1) Sign in/Sign up Pages

Upon opening the website the doctor should either:

Sign in if they are already an authorized doctor user

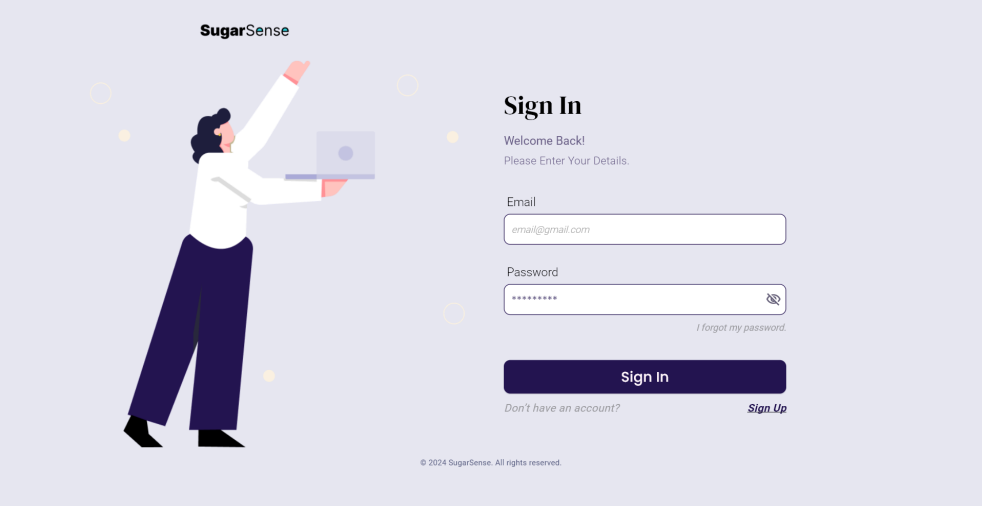


Figure 52: Doctor's sign in page

Or sign up by creating a new account that will ask them for a few information that will be sent to the administration for verification. Along with the Information first/last name, email and password, doctors should also fill out the necessary information about their phone number, healthcare institution name, institution address and syndicate code together with their front and back page of their identity card.

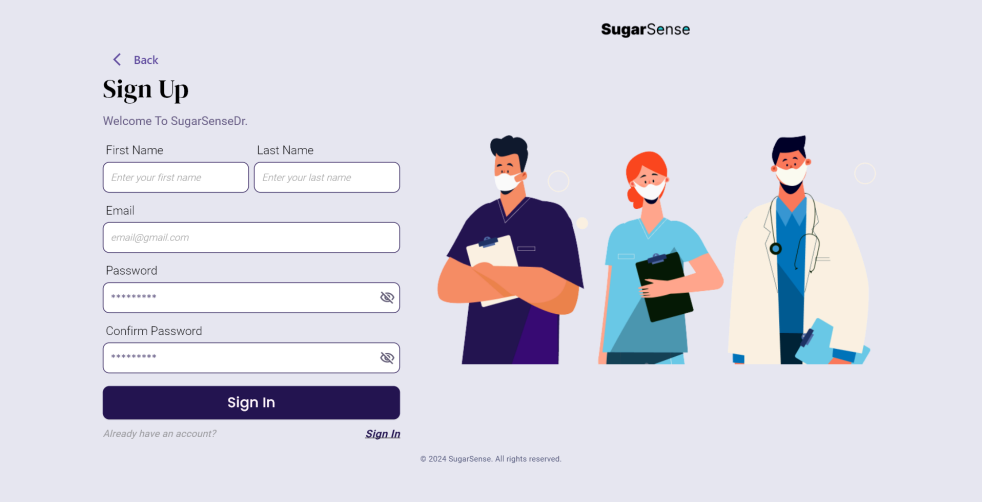


Figure 53: Doctor's sign up page

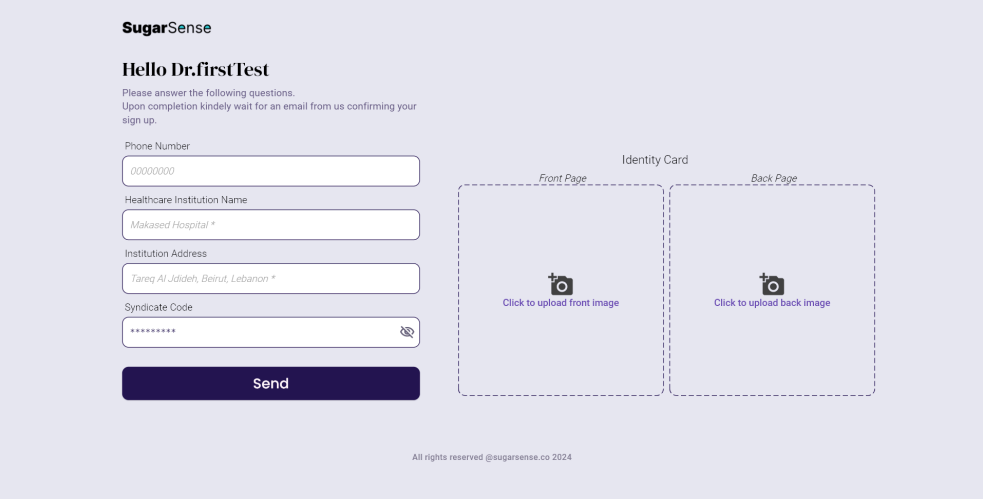


Figure 54: Doctor's identification page

When the application form is completed their information will be sent to the administration for verification, and if accepted the doctor will be able to have access to the website when logging in with their requested information.

2) Patients Page

Patients page plays a crucial role in showcasing the list of patients the doctor is assigned to along with their important information such as the patient's email, carb ratios, insulin sensitivity, target glucose level and the average glucose level of the last 3 months which also indicates the hyper/hypo indication depending on the color of the average container(green: normal, blue: hypo, red:hyper). Doctors are also able to search for a specific patient in the list using the search bar that exists at the top of the page.

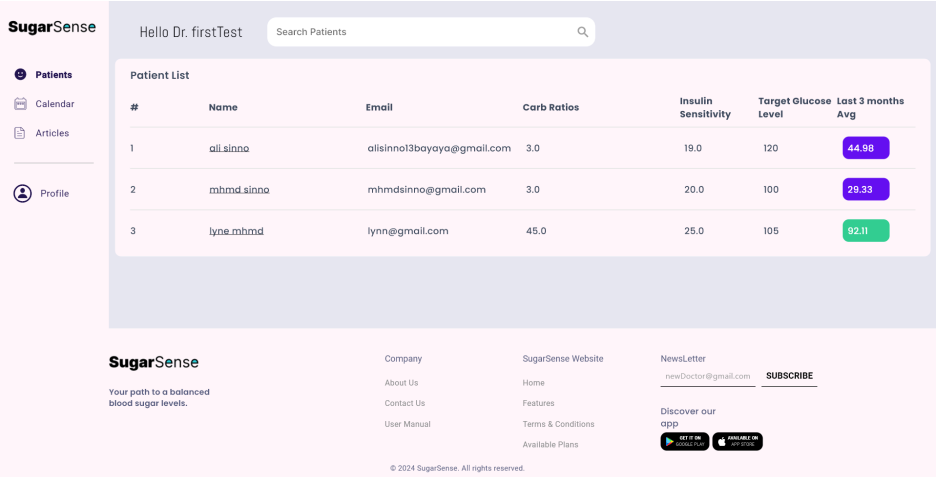


Figure 55: Patient's list page of doctor

3) Patient Details Page

This page is the heart of the website, where all of the detailed information of the chosen patient is displayed. It consists of 4 components, the first holding the carb ratios, insulin sensitivity and target glucose levels, along with the ability to change them, it also showcases the patients next appointment date at the top of the page. The second component consists of pie graphs for each time range(3 months, 1 month and 1 week), the pie graph shows the percentages of hypo/hyper activities in each time range. The third component displays the line graph of glucose levels for each time range , along with the glucose and bolus average levels for each. The last component shows the history of the patient's input throughout the last 3 months, 1 month and the last 1 week, taking into consideration the privacy components the patient wants to conceal from his doctor.

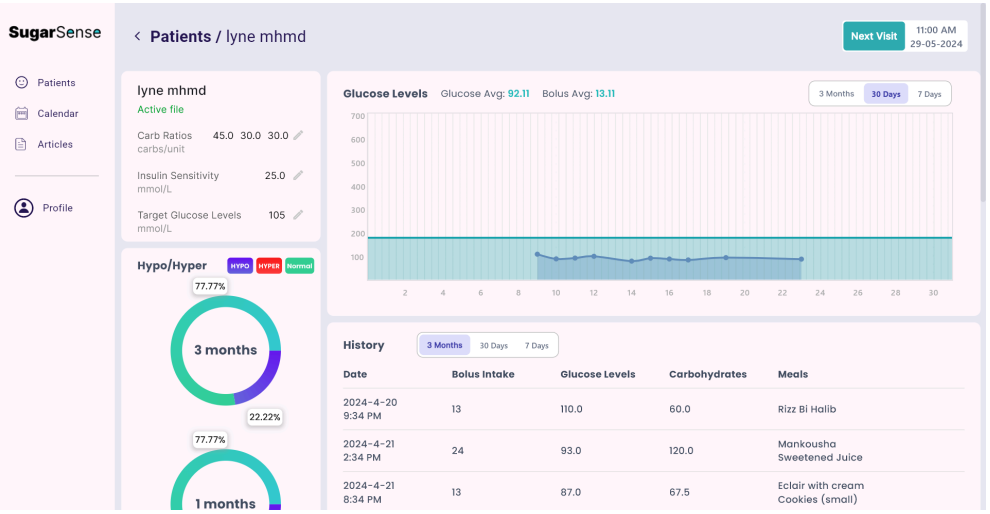


Figure 56: Patient's details page of doctor

4) Calendar Page

All the doctors appointments with his patients are displayed in the calendar page. Doctors can view the calendar, add a new appointment and delete the week appointment if wanted.

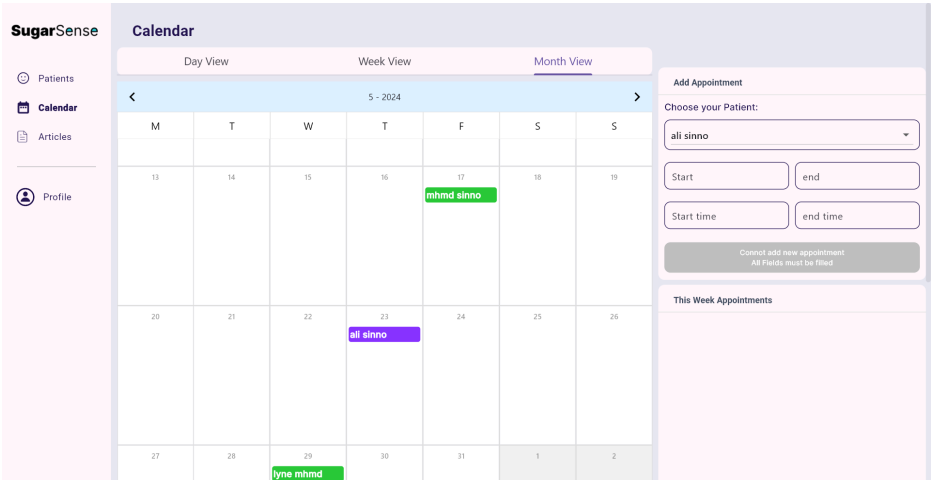


Figure 57: doctor's appointment schedule

5) Articles Page

Like their patients, doctors are also able to browse through and read the latest articles related to diabetes.

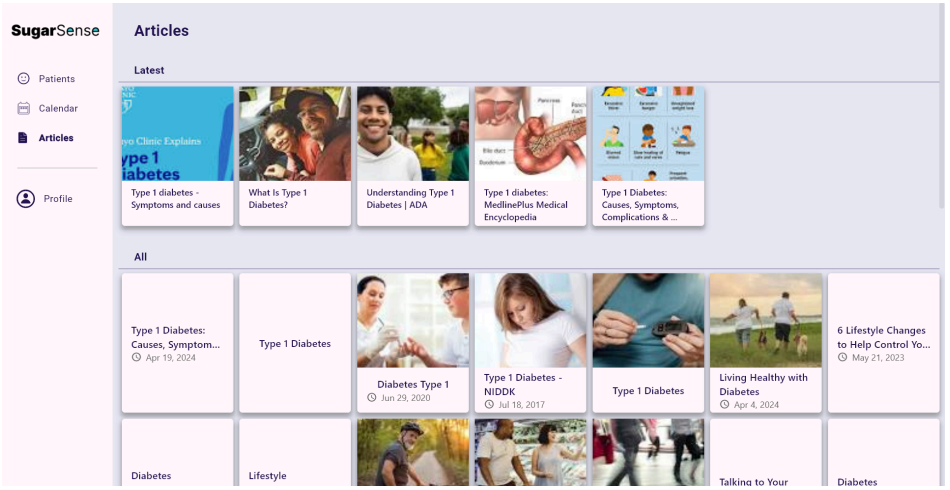


Figure 58: doctor's articles page

6) Profile Page

Through the profile page doctors are allowed to change their email address, phone number and password along with choosing the unit they want the patient details to be displayed in. And most importantly the random generated doctor code is displayed at the top of the page where the doctor can provide their patients with to connect with them.

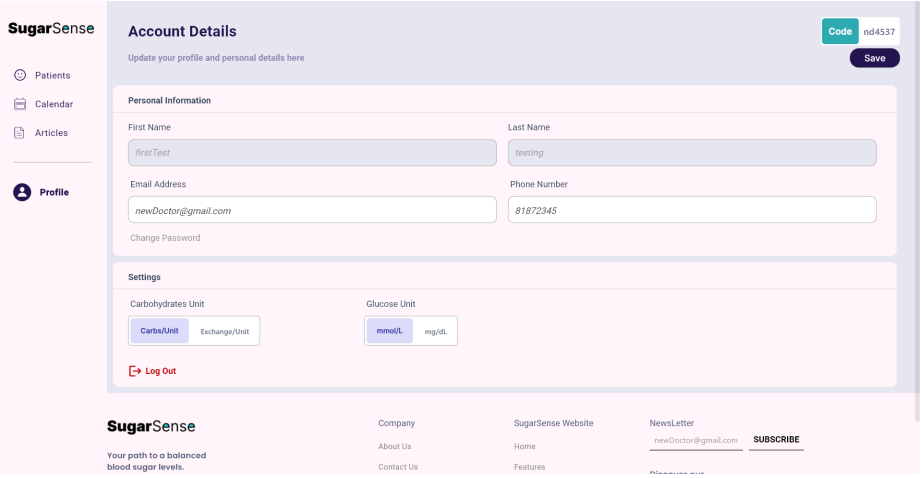


Figure 59: doctor's profile page

D. Website Implementation (SugarSense website)

We designed an engaging and visually appealing website to effectively showcase our senior project, the SugarSense application. The website serves as an informative and advertisement platform, aimed at providing users with a comprehensive overview of the features and benefits of SugarSense, while also allowing them to easily navigate through different sections and interact with our team.

The website features a sleek and intuitive user interface, with a navbar located on the top right corner, providing easy access to different pages including the Home, About Us, Features, User Manual, and Contact Us sections. The navbar is designed to be responsive and user friendly, allowing users to seamlessly explore the website and access relevant information. At the top of the website, along with a captivating illustration showcasing the launch screen of the SugarSense app, giving users a sneak peek into the user interface of the application. Accompanying the illustration are two prominent download buttons that direct users to the app store and play store, providing a convenient and straightforward way for users to download the SugarSense app and experience its features first hand along with SugarSenses slogan.

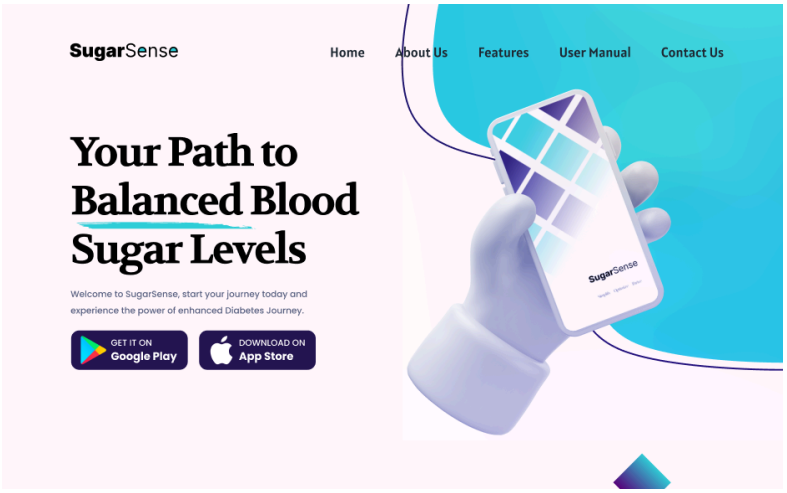


Figure 60: SugarSense website landing page

As users scroll down the home page, they are greeted with an eye-catching Feature section that highlights the four most important features of SugarSense in an interesting and visually appealing manner. This section is carefully designed to capture users' attention and provide them with a clear understanding of the unique functionalities that SugarSense offers. To further enhance users' engagement, a "Read More" button is prominently placed with each feature, allowing them to delve deeper into the features of SugarSense on a dedicated Features page.

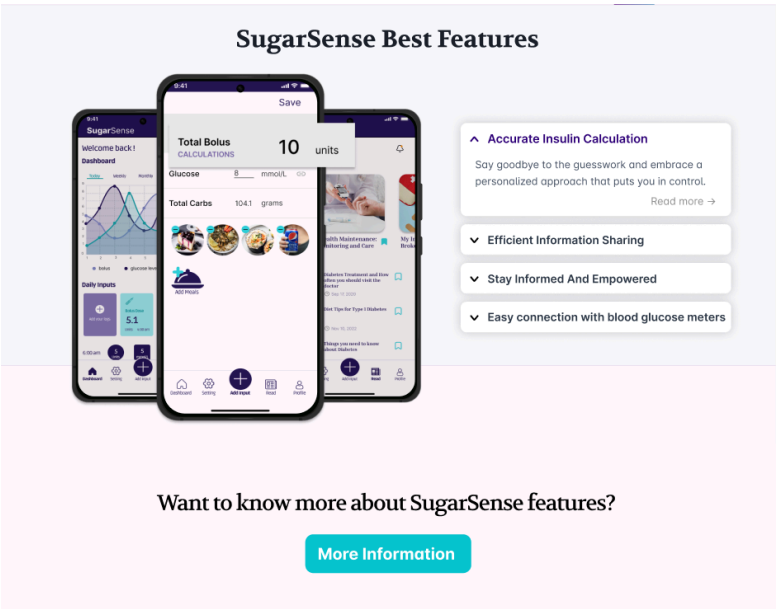
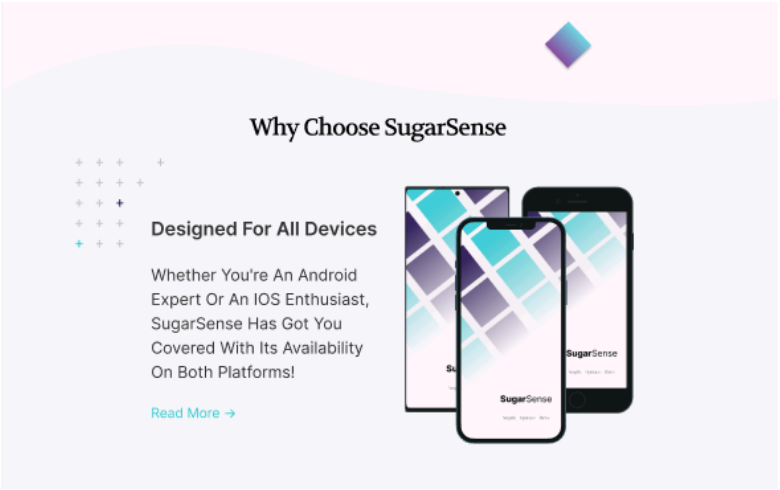


Figure 61: SugarSense website applications' best features section

Continuing their journey through the website, users are introduced to the section that emphasizes the user-friendly interface and encouraging features of the app that answers why choose SugarSense such as availability of the application on different devices whether IOS or android and accessibility to a wide range of informative articles, instilling trust and confidence in potential users.



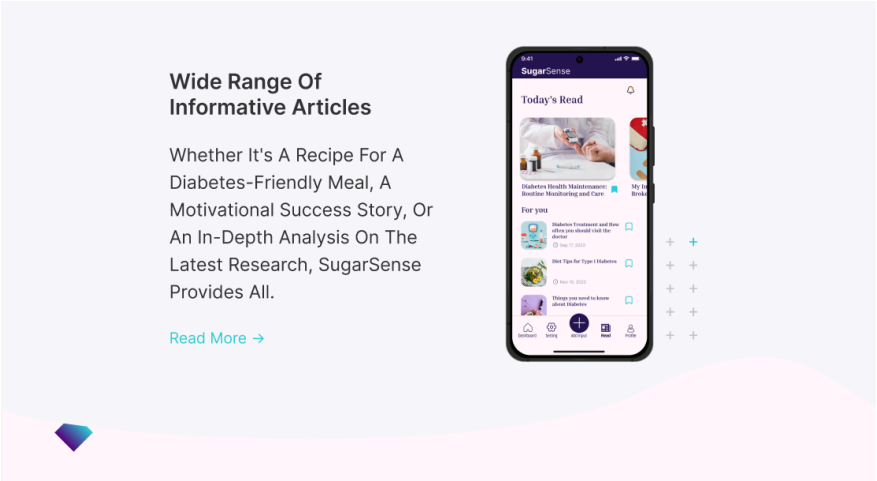


Figure 62: SugarSense website reasons to choose SugarSense section

The About Us section is strategically placed to provide users with insights into the team behind SugarSense, their dedication and hard work in creating the application, and their vision and mission of aiding in managing diabetes easier for users. This section not only showcases the expertise and passion of our team but also builds a connection with potential users by conveying our commitment to making their lives more efficient and organized. Users can read more when they click on the Read More button which will redirect them to the about us page.

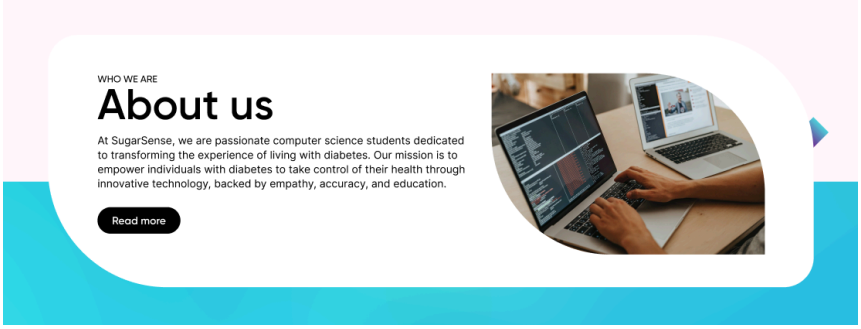


Figure 63: SugarSense website About Us section

Users can view SugarSense's subscription plans by clicking on the View Available Plans button.

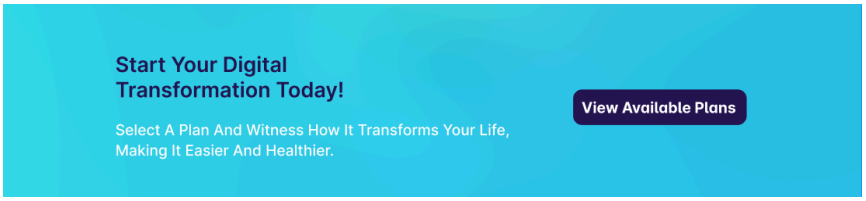


Figure 64: SugarSense website Available Plans section

To keep users engaged and informed, the website includes a Subscribe section that allows them to enter their email address and subscribe to our newsletter for updates on SugarSense. This section is strategically placed to capture users' attention and encourage them to stay connected with us for the latest information and developments.

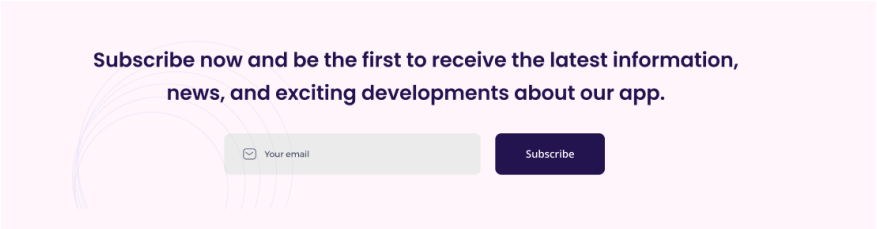


Figure 65: SugarSense website newsletter subscription section

The Testimonial section on the website showcases reviews from satisfied users of the SugarSense app, providing social proof and testimonials about the effectiveness and usability of the application. Users can read through the testimonials and gain insights from real-life experiences, building trust and confidence in the SugarSense app. A "Read More" button is also provided for users to access more reviews and further validate the credibility of SugarSense.

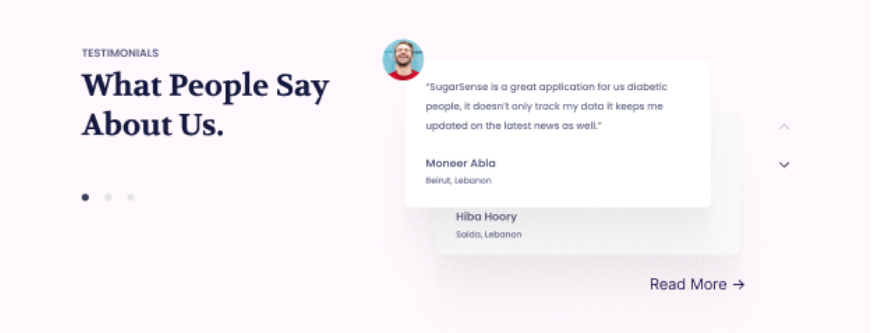


Figure 66: SugarSense website testimonials section

The User Manual page on the website will contain detailed instructions, tutorials, and information about using our product effectively. On the user manual page, users will find sections and chapters that cover various aspects of the application, feature descriptions, troubleshooting tips, frequently asked questions, and more.

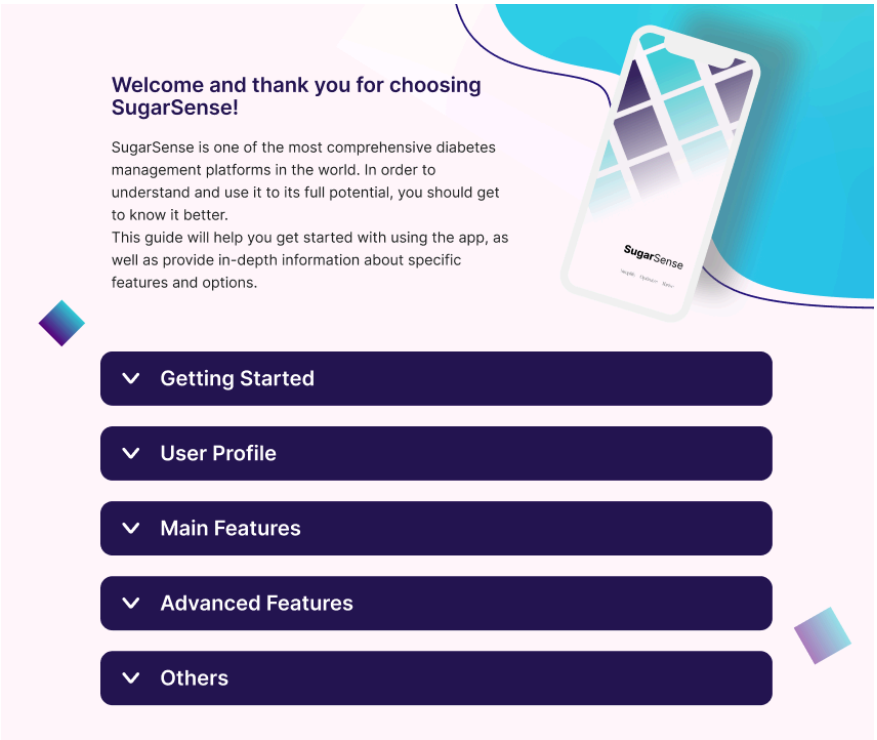


Figure 67: SugarSense website User Manual page

The Contact Us section serves as a means for users to connect with our team by sending messages. Users can provide their name, email address, subject, and message content, allowing them to reach out to us with any inquiries, feedback, or suggestions. This section is designed to be user-friendly and convenient, encouraging users to engage with us and establish a direct communication channel.

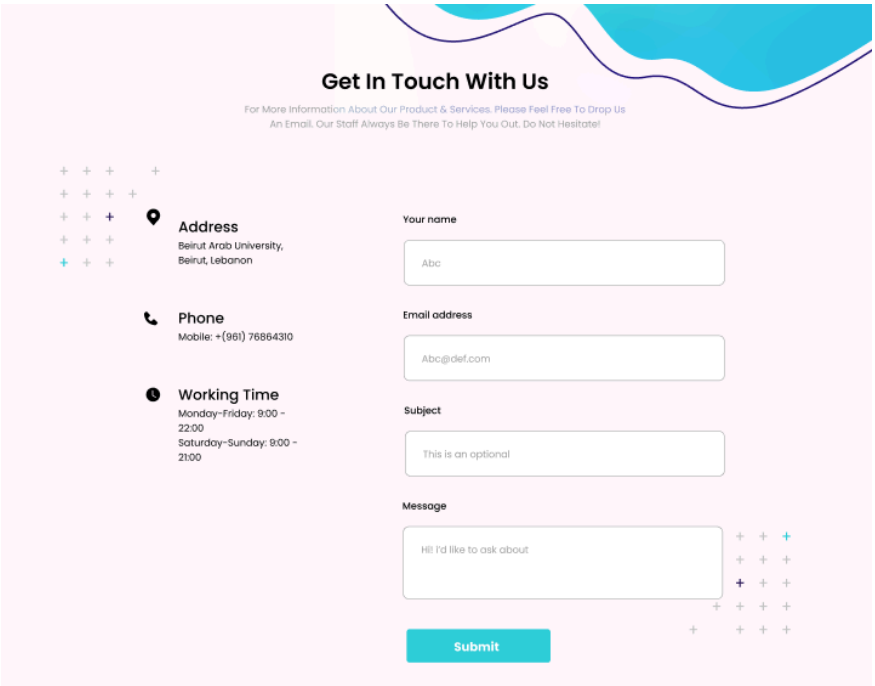


Figure 68: SugarSense website Contact Us page

Finally, the footer of the website summarizes important information, including links to different pages, contact details, newsletter subscription, and links to SugarSense's social media accounts.

This section is designed to provide users with easy access to relevant information and further opportunities to connect with us through different channels. The footer is strategically placed at the bottom of the website, ensuring that users can easily access important information without having to scroll back to the top.

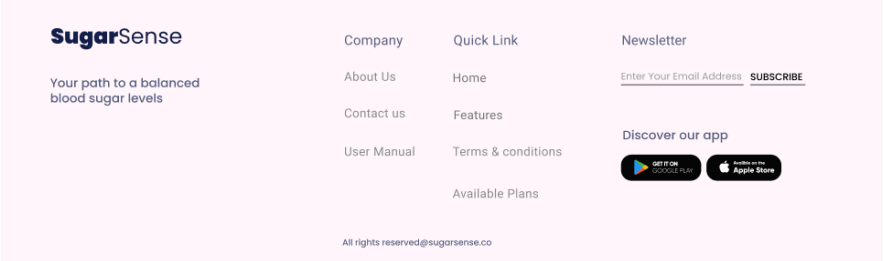


Figure 69: SugarSense website footer section

In addition to the various sections and pages, the overall design of the website is carefully crafted to create a visually appealing and engaging experience for users. The use of captivating visuals, including images and graphics, complemented by concise and impactful text, effectively conveys the key messages and benefits of SugarSense. The color scheme and typography are chosen to align with the brand identity of SugarSense, creating a cohesive and professional look and feel throughout the website.

Furthermore, the website is designed to be responsive, ensuring that it is accessible and functional on different devices, including desktop computers, laptops, tablets, and smartphones. This ensures that users can access the website and navigate through its pages seamlessly, regardless of the device they are using, providing a consistent and user friendly experience.

Overall, our website serves as a comprehensive and engaging platform for showcasing the features and benefits of SugarSense. It not only provides users with information about the app, but also offers them an interactive experience, allowing them to easily navigate through different sections, learn more about the app's features, read testimonials, subscribe to the newsletter, and connect with our team. The visually appealing design, intuitive user interface, and compelling content work together to create an informative and engaging website that effectively promotes the SugarSense app and encourages users to download and try it out for themselves.

As our team continues to refine and enhance the SugarSense app, we are committed to keeping the website up to date with the latest information and developments. We will continuously listen to user feedback, analyze website analytics, and make necessary improvements to ensure that our website remains a valuable resource for users interested in learning about SugarSense and its benefits. We are excited to share our project with the world through our website and look forward to welcoming new users to experience the convenience and efficiency of SugarSense for managing their diabetes and simplifying their lives.

E. Testing

Testing is a crucial part of software development because it ensures the usability and functionality of the application. Testing took place during the development cycle and after the implementation is complete. There are multiple factors that we considered when testing software which are:

- Devices:

The tests will be held on a set of physical Android and iOS devices to validate its cross platform functionality. To extend our coverage we used emulations to apply tests on devices that we don't own. This approach combines the reliability of physical testing with the scalability of emulation. Additionally, we focused on low end devices because that's the most commonly used by our target audience because it includes a lot of children and elderly.

- Unit testing:

Its main purpose is to confirm that each part of the app works as intended independently. This isolation allows for easier identification of the location of bugs. Unit testing can be achieved using test cases in which we give said unit a set of inputs expecting an exact output. The inputs include edge cases to make the app foolproof. (Appendix E)

Test Case ID:	add inputs -1A		Test Case Description:	calculate insulin dosage	
Pre-requisites:	user must be signed in		Test Priority:	very high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	total carbs	press calculate button	display total carbs of meals chosen	display carbs	pass
2	glucose level	input glucose	takes input of glucose	display glucose input	pass
3	select carb ratio	select from drop down list if mutliple	carb ratio selected successfully	display carb ratio chosen	pass
4	get insulin dosage	none	display correct insulin dosage	calculated dosagee successully	pass
5	create entry	press save button	navigate to dashboard	directed to dahsboard	pass
		if one input missing	pop up message "please input meals and calculate"	pop up displayed	pass
		if no connection	saves locally and waits for next input to sync on the server (1 sec)	new entry as saved the local databse and server	pass
Post-Conditions	a new entry has been saved into the database				

Figure 70: calculate insulin test case

Each test case describes a specific operation or feature in the application and holds the following factors: an test case ID, description, priority (low,mid or high), prerequisites, execution steps including all possible inputs, expected and actual output followed by the test result and finally post-conditions of the test case.

- Integration testing:

After confirming that a unit works as intended it's important to test its integration with the rest of the system, thus system integration testing was used to ensure that the application works as a whole from the user interface to the back end. In this step we focused on the connection between different layers, for example we checked if the user input is being accurately taken by the AI model and the output is being stored correctly in the SQL database.

- Usability testing:

We conducted sessions with potential future users to view and analyze their interaction with the software. This could allow us to discover bugs that may have gone unnoticed. Additionally we asked them to perform some tasks to confirm the user-friendliness of the app and redesign the tasks that were not intuitive. We also took all their feedback into consideration.

In conclusion the tests we deployed are essential and sufficient to make sure that the app meets all the software requirements specification including functionality, usability, ease of use and cross platform compatibility.

F. Maintenance

Making and releasing our application doesn't mean our work is over. Maintenance is in fact a significant task in the development cycle of the app; thus, we will try to provide a smooth user experience using it. To avoid confusion and manage the code, we will be using Git for version control, as it will make sure that everyone can work on the code simultaneously and will display changes and that all the team members are up to date. Since technology is always advancing, after we launch SugarSense, we will continue to update the application often and make sure that it is compatible with the latest devices. This will happen by working on the code and ensuring that it is always up to standards.

Chapter 6. Conclusion and Future Work

In conclusion, we realize after this detailed research that SugarSense will not be an easy project to implement. There were some other simpler paths or common ideas we could have taken but we wanted to make a significant change in people's life and develop an impeccable application that can be a safe guide for all diabetic patients all around the world.

SugarSense will ideally fulfill its duties and assist diabetic patients by providing them with trustworthy authenticate resources to further educate them on diabetes lifestyle; it will protect their health and well-being by generating an almost accurate estimation of meals and recommend the optimal insulin dosage for patients, reducing any medical emergencies or any decline in health; it will keep track the important data of patients over the days, month and years and give them a detailed representation that will be accessed directly with their doctor. Instead of scheduling frequent check ups, the doctors can continuously observe and treat their patients safely. SugarSense will be "Empowering Lives, One Dose at a Time".

Although, It is important to keep in mind that to launch SugarSense into the world, we had to go through a particular course of action. We went through a thorough research studying all the possible risks and obstacles we might face ahead of development. We organized weekly meetings with our advisor to keep track of our progress to ensure a transparent and productive workflow throughout the semester. And finally after making sure our application is up to the standard and testing it, we will launch it onto the market. When SugarSense becomes a successful application we will ensure continuous and constant maintenance of the system and develop updated features.

The first primitive version of SugarSense will be just the beginning. Several features will definitely be integrated into the application over the years. Our team will always work their best to improve newly diabetic's lives and guide them to safety. So, some of the feature we plan to add later on is the ability of doctors to rate or comments on articles and recommended them to their patients automatically, make in-app challenges to motivate users, peer support and social features, goal setting and progress tracking, connect with their glucose meter, a virtual assistant and possibly further extend our community to include also non-diabetic users such as bodybuilders that follow a strict carbohydrates diets or simply people who wanna keep track of their daily sugar intake. SugarSense can be a safe haven to all that wish to join.

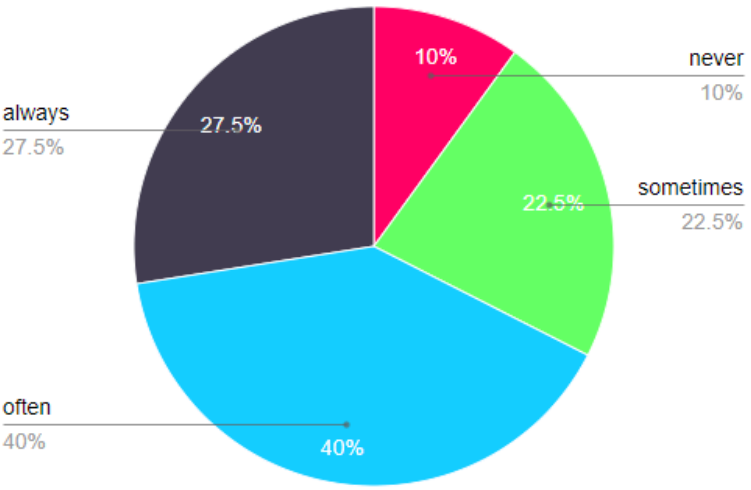
Chapter 7. References

1. https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sqlite
2. https://medium.com/intro-to-artificial-intelligence/soft-actor-critic-reinforcement-learning-algorithm-1934a2c3087f
3. https://radixweb.com/blog/importance-of-mobile-app-maintenance
4. https://www.baeldung.com/cs/layered-architecture
5. https://nishothan-17.medium.com/pbkdf2-hashing-algorithm-841d5cc9178d
6. https://articles.wesionary.team/tips-tricks-for-improving-the-performance-of-your-flutter-app-3f9b2dacdfe1
7. https://www.figma.com/community/search?resource_type=mixed&sort_by=relevancy&query=websites&editor_type=all&price=all&creators=all
8. https://www.beatdiabetesapp.in/
9. https://www.mysugr.com/en
10. https://diabetes-m.com/
11. https://www.techtarget.com/searchenterpriseai/definition/supervised-learning
12. https://www.ibm.com/topics/supervised-learning
13. https://serpapi.com/
14. https://erdplus.com/edit-diagram/188b3062-1b05-4186-9770-ee9bea812939
15. https://astah.net/

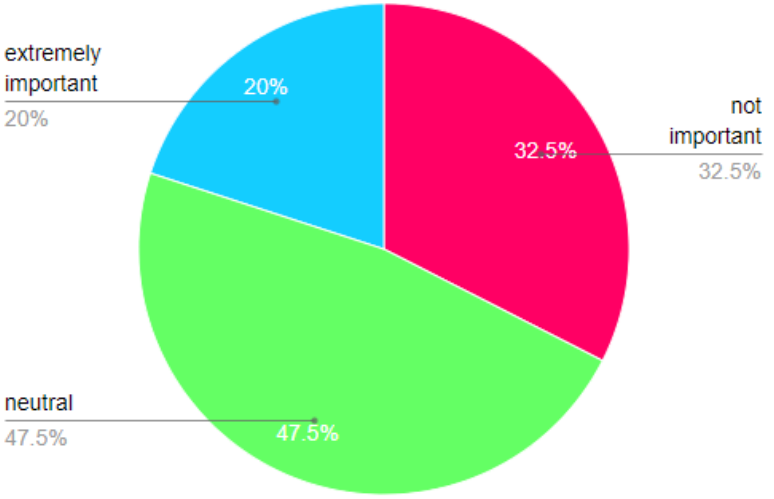
Appendices

1. Appendix A: online survey for operational feasibility.

Q. Do you struggle with estimating how much insulin to take for a specific meal?



Q.how important is it to have real time access to your history of blood glucose and insulin?



2. **Appendix B: links for online surveys of patients and doctors and the website.**

   patients: https://forms.gle/HVSkVJSn22zzbsRr5

   doctors: https://forms.gle/AoqSkYs7TB4CETBcA

   website: https://truebat.github.io/surveyWeb/

3. **Appendix C: SQLite implementation demo for meals.**
   - this is the function to edit a meal:

```
editNewMeal(int parentMealId, String mealName, String picture,
    List<Map> childMeals) async {
  List<Map> response = await getMealById(parentMealId);
  double totalCarbs = response[0]['carbohydrates'];

  childMeals.forEach((element) {
    totalCarbs += element['carbohydrates'] * element['quantity'];
  });

  if (mealName == "") {
    mealName = "My ${response[0]['mealName']}";
  }

  int newMealID = await createNewMeal(
      mealName,
      totalCarbs,
      response[0]['unit'],
      response[0]['mealPicture'],
      response[0]['tags'] + ', myMeals');

  if (newMealID != -1) {
    childMeals.forEach((element) {
      createMealComposition(
          newMealID, element['mealID'], element['unit'], element['quantity']);
    });
    logger.info("Meal has been edited successfully with id $newMealID.");
    return newMealID;
  } else {
    logger.info("Error meal wasn't edited.");
    return -1;
  }
```

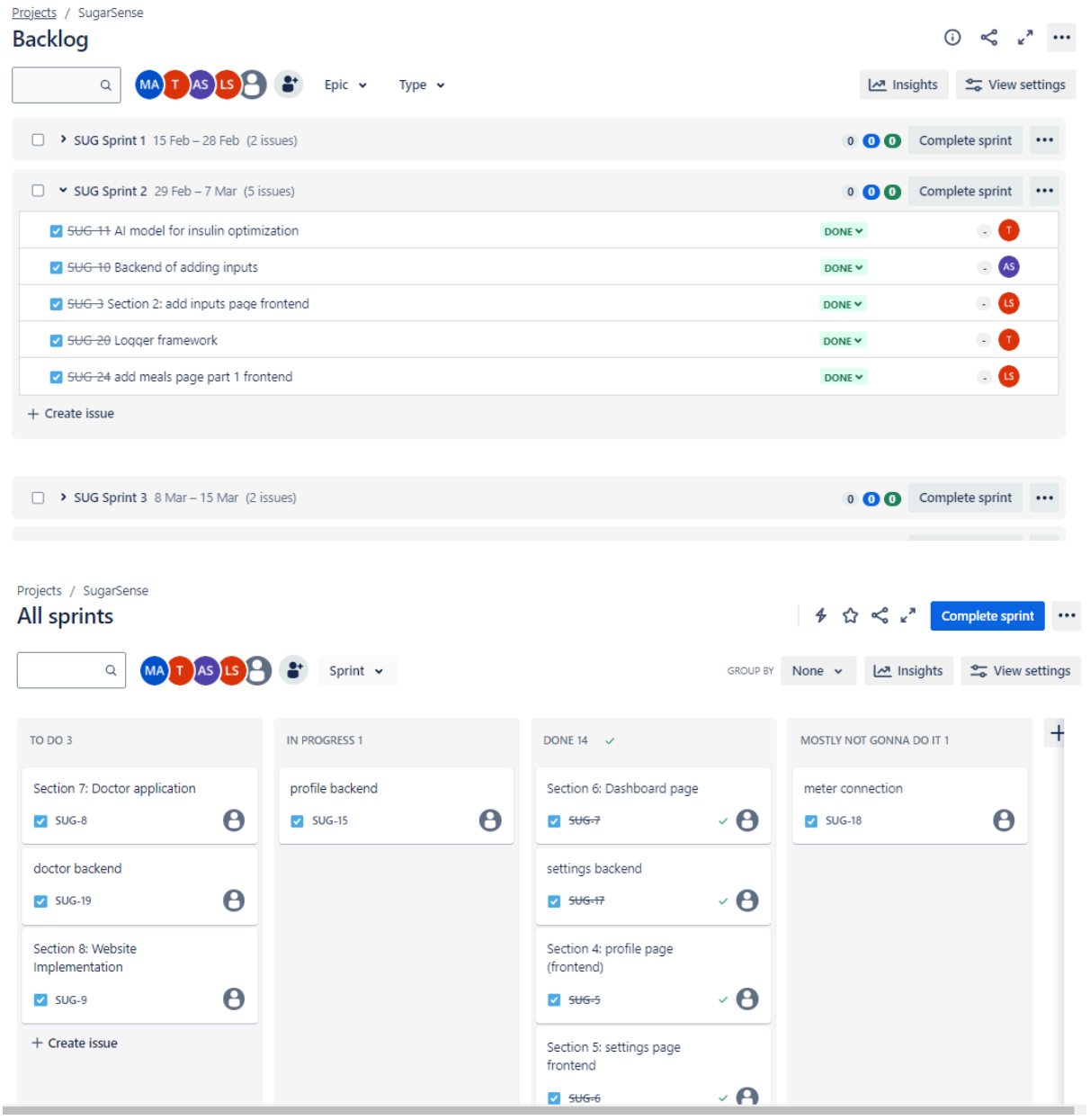
   - this is the function to sync the meals from the server to the local database:

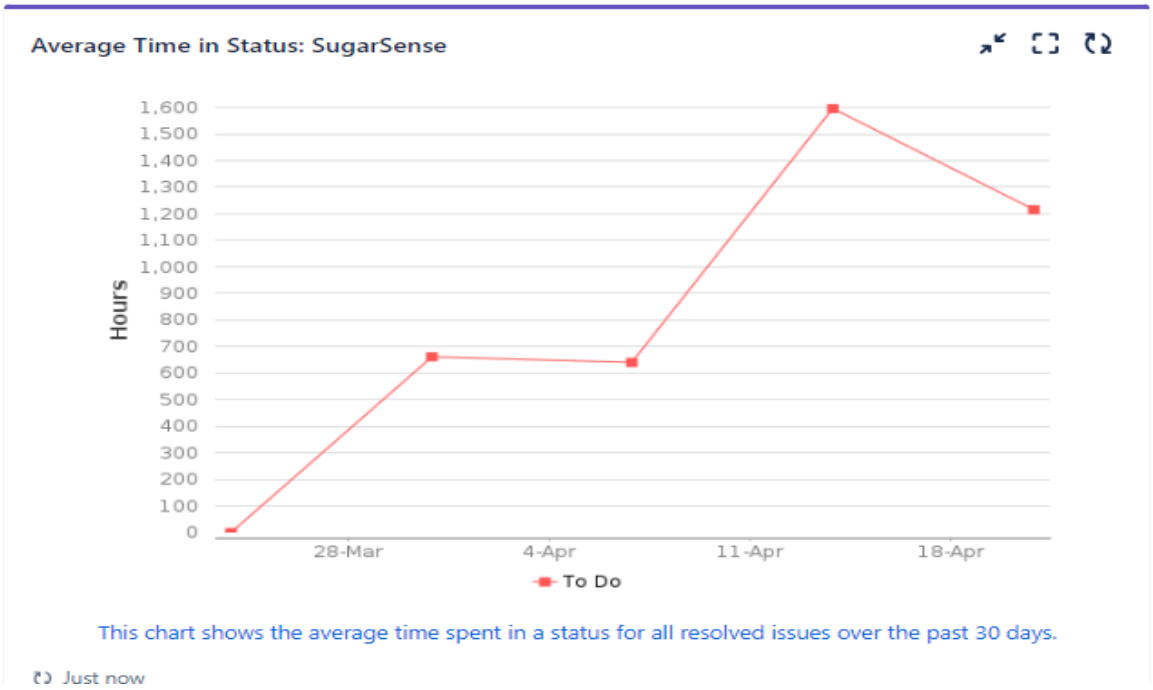
```dart
Future<void> syncMeals() async {
  logger.info("Syncing meals...");
  DBHelper dbHelper = DBHelper.instance;

// Fetch data from the server
  var response = await http.get(Uri.parse('http://$localhost:8000/meals'));
  if (response.statusCode == 200) {
    // If the server returns a 200 OK response, parse the JSON.
    var meals = jsonDecode(response.body);
    final Database? db = await dbHelper.db;
    for (var meal in meals) {
      await db?.insert(
        'Meals',
        meal,
        conflictAlgorithm: ConflictAlgorithm.replace,
      );
    }
    logger.info("Meals have been synced successfully.");
    // Now you can use meals to insert data into the database
  } else {
    // If the server returns an error response, throw an exception.
    throw Exception('Failed to load meals');
  }
}
```

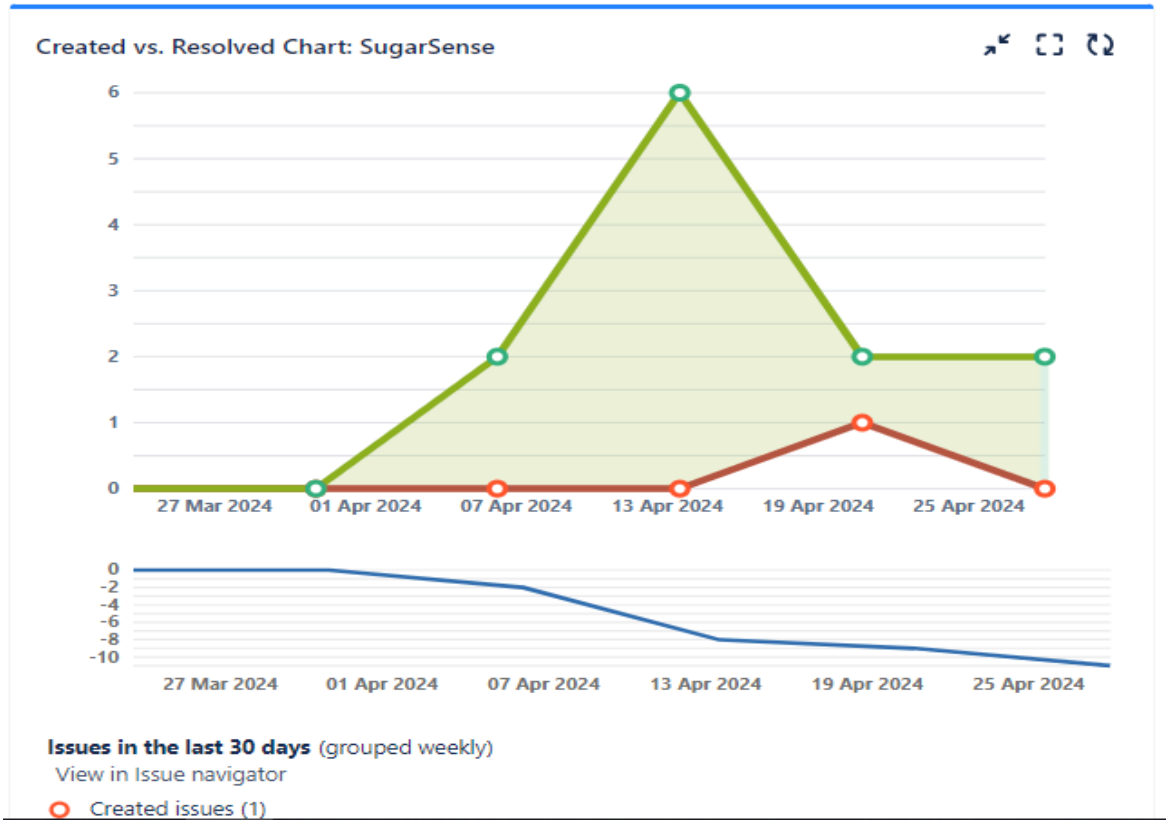
- this is the function to authenticate the user login:

```python
@app.post("/authenticate")
async def authenticate(user: User):
    try:
        # Query the database for the user
        cursor.execute("SELECT userPassword,userID FROM Users WHERE CAST(userName AS NVARCHAR(MAX)) = ?", (user.username,)
        rowUsername = cursor.fetchone()
        cursor.execute("SELECT userPassword,userID FROM Users WHERE CAST(email AS NVARCHAR(MAX)) = ?",(user.username,))
        rowEmail = cursor.fetchone()


        # If the user doesn't exist or the password is incorrect, return a 401 Unauthorized response

        hashed_password = hashlib.md5(user.password.encode()).hexdigest()


        if (rowUsername is None or hashed_password != rowUsername[0]) and (rowEmail is None or hashed_password != rowEmail
            if(rowUsername is None and rowEmail is None):
                raise HTTPException(status_code=401, detail="Invalid email or username")
            else:
                raise HTTPException(status_code=401, detail="Incorrect password")

        # If the email and password are correct, return a 200 OK response
        uid = getUserById(user.username)
        cursor.execute("SELECT * FROM Patients WHERE patientID = ?", uid)
        pid = cursor.fetchone()
        if(pid is None):
            raise HTTPException(status_code=400, detail="This user is not a patient")
        else:
            return {"message": "Authenticated successfully", "ID": uid}
            #rowUsername[1]
    except HTTPException as e:
```

4. Appendix D: the “Jira” project set up along with the progress of workflow.




Average Time in Status: SugarSense
1,600
1,500
1,400
1,300
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
Hours
28-Mar
4-Apr
11-Apr
18-Apr
To Do
This chart shows the average time spent in a status for all resolved issues over the past 30 days.
Just now


Created vs. Resolved Chart: SugarSense
6
5
4
3
2
1
0
27 Mar 2024
01 Apr 2024
07 Apr 2024
13 Apr 2024
19 Apr 2024
25 Apr 2024
0
-2
-4
-6
-8
-10
27 Mar 2024
01 Apr 2024
07 Apr 2024
13 Apr 2024
19 Apr 2024
25 Apr 2024
Issues in the last 30 days (grouped weekly)
View in Issue navigator
Created issues (1)

5. Appendix E: The test cases that helped organize our testing phase.

Test Case ID:	Login-A1		Test Case Description:	Login as existing user	
Pre-requisites:	user must have signed up already		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	
1	Launch application	none	display login page	login page	pass
2	Fill in the required fields				
	a- username not found	username: alisinnoe password: alisinno1	The snackbar pop "username not found" should be displayed.	snackbar message displayed	pass
	b- email not found	email: alisinno@gmail.comr password: alisinno1	The snackbar pop "email not found" should be displayed.	snackbar message displayed	pass
	c- incorrect password	username: alisinno password: alisinno123	The snackbar pop "incorrect password" should be displayed.	snackbar message displayed	pass
	d- correct login info	username: alisinno password: alisinno1	successful login (2-3 sec)	successful	pass
		email: alisinno@gmail.com password: alisinno1	successful login (2-3 sec)	successful	pass
3	no internet connection	none	The snackbar pop "no internet connection" should be displayed.	snackbar message displayed	pass
4	patient did not subsrcibe	none	navigate to membership plan	navigated to membership page	pass
5	user is not a patient	none	navigate to questions page	navigated to questions page	pass
6	User forgot password	press on forgot password	direct to forgot password page	forgot password page	pass
7	Second time login	none	immediately login	directed to dashboard	pass
Post-Conditions	User has access to the dashboard				

Test Case ID:	SignUp-1A		Test Case Description:	Sign up as a new user	
Pre-requisites:	none		Test Priority:	High	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	Launch application	none	display login page	login page	pass
2	click on sign up link	sign up link	display sign up page	sign up page	pass
3	Fill in the required fields:				
	a- One of the required fields is empty	first name: last name: username: email: password: confirm password:	The snackbar pop "all fields with * must be filled" should be displayed.	snackbar message showed	pass
	b- username already exists	username: bsmith	The snackbar pop "username already exists" should be displayed.	snackbar message showed	pass
	e- email already exists	email: bSmith@gmail.com	The snackbar pop "email already used" should be displayed.	snackbar message showed	pass
	d- invalid email	email: alisinnoemail.com email:alisinno@emailcom email: alisinnoemailcom	The snackbar pop "invalid email" should be displayed.	snackbar message showed	pass
	e- invalid password	password: 123456 password: 12345678 password: abcdefgh	The snackbar pop "password must be at least 8 character and contains at least a number" should be displayed.	snackbar message showed	pass
	f- confirm password does not match	password: 1234567E confirm: 1234567G	The snackbar pop "password and confirm do not match" should be displayed.	snackbar message showed	pass
	g- wrong doctor code	doctor ID: NULL/"ndjhf"	The snackbar pop "wrong doctor ID" should be displayed.	snackbar message showed	pass
	h- no internet connection	None	The snackbar pop "internet connection is required" should be displayed.	snackbar message showed	pass
	i- correct information	first name: ali last name: sinno username: aliSinno doctor ID: password: 12345678j confirm: 12345678j email: alisinno@gmail.com	Sign up is successful. User is directed to subscription page. (6-8 sec)	accept terms and conditions and directed to subscription plan page	pass
Post-Conditions:	New user is added to the database and the user to the subscription plans.				

Test Case ID:	Sign Up-1B		Test Case Description:	insput information about patient	
Pre-requisites:	successful sign up		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	input carb ratio:				
	a- one of the fields is empty	none	alert pop "please enter a value"	alert message displayed	pass
	b- input negative value	carbs: -1 uniy: 3	no input is displayed	no input showed	pass
	c- add another carb ratio	press + button	another textboxes are displayed	new carb ratio was displayed	pass
	d- delete a carb ratio	press delete button	the textbox assigned is removed	added carb ratio is removed	pass
2	input insulin sensivity & target glucose level:				
	a- one of the fields is empty	none	alert pop "please enter a value"	alert message displayed	pass
	b- input negative value	carbs: -1 uniy: 3	no input is displayed	no input showed	pass
3	input privacy	depends on user	loading sign pop up	alert message displayed	pass
4	no internet connection	none	alert pop "internet is required"	alert message displayed	pass
Post-Conditions	a new patient was created and user is directed to login page				

Test Case ID:	login- 1B		Test Case Description:	forgot password link	
Pre-requisites:	user must have signed in		Test Priority:	moderate	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	user inputs their email				
	a- email not found	email: "bob@gmail.com"	snackbar "email not found is displayed"	snackbar message displayed	pass
	b- email found	email: "bob23@gmail.com"	random code is sent to user	code sent through email	pass
2	user code sent				
	a- code is incorrect	correct code: FGKBHY input: FGKBHJ	snackbar "invalid code"	snackbar message displayed	pass
	b- code is not complete	correct code: FGKBHY input: FGKBH	snackbar "please enter code"	snackbar message displayed	pass
	c- input after 10 min	code: FGKBHY	snackbar "code expired"	snackbar message displayed	pass
	c-succesful	code: FGKBHY	redirected to input new password	user is allowed to change password	pass
Post-Conditions	password is successfully changed and user directed to login page				

Test Case ID:	dashboard- 1A		Test Case Description:	display history of patient	
Pre-requisites:	user must be signed in		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	display graphs of entries	none	graphs displayed correctly	graphs displayed correctly	pass
	filter them by day, month & 3 months	press the montly filtration inputs	graphs displayed correctly according to the choice	graphs displayed correctly according to the choice	pass
	pick one specific attribute to analyze	press the filtrating buttons	graphs displayed correctly according to the choice	graphs displayed correctly according to the choice	pass
2	display latest Entry & daily entries	none	display last entry recorded	lates entry displayed	pass
3	delete latest entries				
	internet available	press delete button	delete locally and remotely	delete entry locally	pass
	no connection	press delete button	delete locally and wait for connection to be deleted remotely	delete entry locally and remotely	pass
	2 hours has passed	none	delete button disappears	delte button disappeared	pass
4	navigate to add iputs page	press 'add your logs' box	nagivate to add new inputs	navigated to add inputs page	pass
5	display entries of the past week	press arrows for daily details	get insulin, glucose and carbs of day for the past week	displayed weekly entries	pass
Post-Conditions	patient can read and analyze				

Test Case ID:	settings- 1A		Test Case Description:	modify patient info	
Pre-requisites:	user must be sign in		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	chang units	click on unit option	convert all current unit	unit was converted	pass
2	alter target glucose & insulin sensivity & carb ratio & privacy				
	a- input negative value	target= -100	no input is displayed	no input shown	pass
	b- add carb ratio	press + button	carb ratio is added	carb ratio is added	pass
	c- remove carb ratio	press delete button	carb ratio is removed	carb ratio is removed	pass
	d- modify info	insulin sensivity: 23 privacy: 101 ...	all data was modified	all info has been edited in the server	pass
3	internet connection	carbRatio: 1/3 --> 1/3.5	updated in server	values updated	pass
	if no internet	carbRatio: 1/3 --> 1/3.6	wait for connection to update	values updated	pass
4	change doctor info	doctorCode: 'we120'	doctor found	new connection was made	pass
Post-Conditions	all patient info has been updated successfully				

Test Case ID:	profile- 1A		Test Case Description:	edit profile information	
Pre-requisites:	user must be signed in		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	edit username & email & phone				
	a- username already exists	username: alisinno	alert message pop "username already exits"	alert message displayed	pass
	b- email already exists	alisinno@gmail.com	alert message pop "email already exists"	alert message displayed	pass
	c- phone already exists	81854367	alert message pop "phone number already exists"	alert message displayed	pass
	d- invalid email	alisinnogmail.com	alert message pop "invalid email"	alert message displayed	pass
	e- invalid phone number	8743241m	alert message pop "invalid phone number"	alert message displayed	pass
	f- edit profile picture	image	new profile image is displayed	new image displayed	pass
	g- edit nothing	none	direct back to profile	pop to profile	pass
	h- edit any fields	input any fields	info has been updated	pop to profile	pass
2	change password				
	a-old password id wrong	input wrong password	alert message pop "old password is wrong"	alert message displayed	pass
	bnew password is invalid	password is < 8 char or has no letters	alert message pop "password must be at leat 8 character and has one letter"	alert message displayed	pass
	c-new password is the same as old password	old and new passowrd are the same	alert message pop "new passowrd s the same as old password"	alert message displayed	pass
	d- old and new password are verified	both old and new password check out	new password is updated	password updated	pass
Post-Conditions	user was able to edit all his information and save them in the server.				

Test Case ID:	add inputs -1A		Test Case Description:	calculate insulin dosage	
Pre-requisites:	user must be signed in		Test Priority:	very high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	total carbs	press calculate button	display total carbs of meals chosen	display carbs	pass
2	glucose level	input glucose	takes input of glucose	display glucose input	pass
3	select carb ratio	select from drop down list if mutliple	carb ratio selected successfully	display carb ratio chosen	pass
4	get insulin dosage	none	display correct insulin dosage	calculated dosagee successully	pass
5	create entry	press save button	navigate to dashboard	directed to dahsboard	pass
		if one input missing	pop up message "please input meals and calculate"	pop up displayed	pass
		if no connection	saves locally and waits for next input to sync on the server (1 sec)	new entry as saved the local databse and server	pass
Post-Conditions	a new entry has been saved into the database				

Test Case ID:	add inputs- 1B		Test Case Description:	add meals for specific entry	
Pre-requisites:	user must be signed in		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
			filtering meals		
	filter by category	click tags of categories	filter meals correctly	filter by category	pass
	search for meals	input meal name of differents spelling	get meals the user inputed	display correct meal	pass
1	display frequent meals in "myMeals"	navigate to myMeals categories	display frequent & personal meals if exists	display frequent meal and myMeals	pass
2			add meals		
a-	add meal directly	press on meal image	display pop up of meal	pop of meal chosen	pass
	1- input decimal/integer as an amount	amount: 2.0 amount: 1	no alert displayed	alert message displayed	pass
	2- display a negative Or 0 amount	amount: -3 amount: 0	alert message "amount must be positive non-zero"	alert message displayed	pass
	3- meal added to list	press okay button	navigate to add inputs page	direct to add inputs page	pass
b-	add through meal details	press on meals arrow	display meals details	display details of meals chosen	pass
	1- display tag, unit, carbs of the meal	none	display accordingly all meal's information	tags display correctly	pass
	2- input negative amount	amount: -2	pop up messgae "cannot input negative or zero values"	pop up message displayed	pass
	3- input 0 amount	amount: 0	pop up messgae "cannot input negative or zero values"	pop up message displayed	pass
	4- input empty amount	none	pop up messgae "specify the amount as a number"	pop up message displayed	pass
	5- meal added to list	press add to meals button	navigate to add inputs page	directed to add inputs page	pass
	6- edit meal	press edit button	navigate to edit meal page	directed to edit meal page	pass
3	create a meal	press create button	navigate to create meal page	directed to create meal page	pass
Post-Conditions	user will have list of meals ready to input				

Test Case ID:	add inputs -1D		Test Case Description:	user create his own meals	
Pre-requisites:	user must have access to meals page		Test Priority:	high	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	meal's picture	input user's image	saves user's image	user's image is saved for meal created	pass
		none	saves a default image	saves a default image	pass
2	meal's name	meal name:pancake	pop up message "meal name already exists"	pop up displayed	pass
		meal name: pancake2	saves new meal name	save name to meal created	pass
		none	pop up message "meal name can't be empty"	pop up displayed	pass
3	meal's categories	choose from lists of categories	meal's categories are added to the new meal	mutliple tags were added to meal created	pass
4	input amount of carbs	amount: 0 amount: -2	pop up message "meal can'y be negative or 0"	pop up message displayed	pass
		none	pop up message "meal carbs can't be empty"	pop up message displayed	pass
5	choose from meals	list of meals chosen	calculate total amount of carbs of meals	catbs were correctly calculated accodrding to quantity of each meal	pass
6	meal created sucessfully	press create button	navigate to myMeals section to add the meal	new meal was added to meal's section	pass
Post-Conditions	new has been created and can be accessed from myMeals section				

Test Case ID:	add inputs- 1C		Test Case Description:	customize a specific meal	
Pre-requisites:	chosen meal must be editable		Test Priority:	moderate	
Test Execution Steps:					
S.NO:	Description:	Inputs:	Expected Output:	Actual Output:	Test Result:
1	meal picture	input his own image	new image is taken	new image is saved to meal edited	pass
		keep default image	default image is taken	default image was taken	pass
2	meal name	meal name already exits	pop up message "meal name already exits"	pop up message displayed	pass
		new meal name	new meal name is saved	name is saved to edited meal	pass
3	meals ingredients	edit amount of each ingredients	total carbs is calculated based on new ingredients	carbs were calculated according to quantity & crabs of each ingredient	pass
4	saving edited meal	press save button	navigate to myMeals page to later add it	directed to myMeals section	pass
Post-Conditions	meal has been edited successfully and is displayed in myMeals section				