

Doctor Booking System

SDD Major Project

Samuel Gresham

Table of Contents

Table of Contents	1
Project Proposal	2
Defining the Problem – Project Brief	2
Design Specifications	2
Objectives	2
Developer Design Specifications	2
User Design Specifications	3
Interface/Interaction design	3
Social and Ethical Issues	4
Needs of the Client	5
Images of the Software	5
System Modelling	6
Storyboard.....	6
Screen Designs.....	7
Data Dictionary	8
IPO Charts	9
Data Flow Diagram.....	10
Structure Diagram	11
System Flowcharts	12

The current revision of the project is available at:

Client Module <https://samuelgresham12.github.io/Doctor-Booking-System/login.html>
Reception Module <https://samuelgresham12.github.io/Doctor-Booking-System/reception.html>

Documentation, such as the Gantt Chart, can be found at:

<https://github.com/samuelgresham12/Doctor-Booking-System/tree/master/DOCUMENTATION>

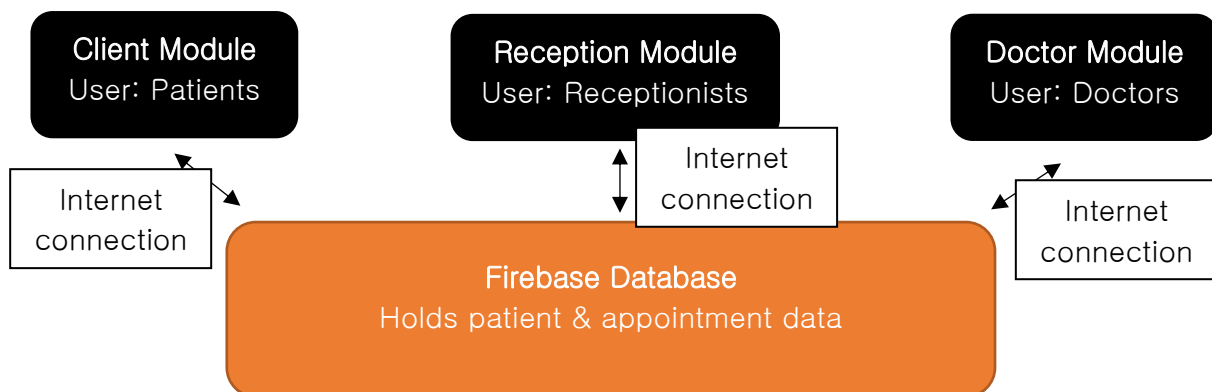
Project Proposal

Defining the Problem – Project Brief

'Doctor Booking System' is aimed to streamline and simplify the process of:

- (i) Booking Creation,
- (ii) Check In and
- (iii) Patient Consultation

in the context of a doctor's office/GP Clinic. It will consist of three modules, which will all interact with a centralised database (Google Firebase), which will hold all patient and appointment data. The general structure of the program is as follows:



A general overview of the functionality of each module is provided below:

1. The **Client Module** allows patients to manage their personal data and create/cancel bookings. A display of upcoming bookings is also provided.
2. The **Reception Module** allows receptionists to view the upcoming bookings on a certain day, as well as check patients in and create over-the-phone bookings.
3. The **Doctor Module** allows the doctors and medical professionals to see past diagnoses and treatments. The doctor can then assess the patient and update the information on the database as necessary.

To generalise, the project will concern itself with the creation, storage and retrieval of patient information from a database, as well as the manipulation of that data.

A modular approach will be used to increase ease of use, as well as to decrease the likelihood that sensitive data is accessed by those who do not require it. Despite this, security will be a considerable issue due to the lack of a secure backend.

Design Specifications

Objectives

The following objectives will be met:

1. Functional modules which are appropriate for the relevant users
2. Appropriate user interface design – easy to use
3. Efficient use of centralised database – minimal reads reduces loading time

Developer Design Specifications

The following design specifications have been set to create a standard development framework:

1. **Data Types** – mostly strings will be used (patient names, comments etc...), but selected integers will be used for things such as times etc...

2. **Data Structures** – the project will store data primarily in objects, with some embedded arrays. For example, each patient is an object, with attributes such as name, Medicare number etc...
3. **Algorithms** – the project will use many algorithms and routines in order to achieve its purpose. These will include sort routines, search routines and general access routines (accessing data from database, queries etc...)
4. **Variables** – many variables are to be created in this project. The naming of these variables should be descriptive of their purpose. Temporary variables, such as incrementors, should be given names that make this obvious (i, temp etc...).
5. **Design Approach** – the project will be a mix of agile and prototyping approach. Prototypes will be consistently made and reviewed. Agile methodologies such as an “ad-hoc” mentality will also be adopted.
6. **Quality Assurance** – it is important that the final software is reliable and easy to use. For this reason, QA will be performed to ensure that the software operates correctly in a range of use cases. QA will be performed using test data and load testing, as well as aesthetic and UX refinement.
7. **Modelling** – a range of modelling tools will be used, including:
 - a. Storyboards
 - b. Screen Designs
 - c. Data Dictionaries
 - d. IPO Charts
 - e. Data Flow Diagrams
 - f. Structure Charts
 - g. System Flowcharts
8. **Documentation** – documentation is very important for this program, since users from many backgrounds and abilities will be using it. Such, proper documentation in the form of an online how-to guide will be created.

User Design Specifications

1. All module interfaces are to be made through the Bootstrap CSS library for consistency
2. Consistent use of colour will be used:
 - a. Red to signify a significant or critical action (i.e. deleting a booking)
 - b. Green to signify a commonly used action (i.e. logging in)
 - c. Grey to signify a less commonly used action (i.e. creating an account)
 - d. Yellow to signify a dangerous but reversible action (i.e. logging out)
 - e. Blue to signify a harmless, reversible action (i.e. updating personal details)
3. The client module is to be simple and easy to navigate, as to make it as easy as possible for a variety of people to use.
4. Testing is to be done to ensure ease of use and functionality.
5. Security will be developed to ensure that sensitive patient data is not vulnerable.
6. See *social and ethical issues* for accessible interface information.

Interface/Interaction design

The interface is to be made easy to use and functional by:

1. Use of consistent elements such as colour (see *above*)
2. Use of appropriate input types (drop-down boxes, input boxes, date etc...) to make data entry easier, and also to restrict incorrect data input.
3. Grouping related elements together (such as the ‘personal data’, ‘upcoming bookings’ and ‘make a booking’ columns in the client module) to make use more ergonomic.
4. Creating a user guide, to help users navigate each module.

5. Identification and testing with relevant audiences. This project will engage with a wide audience, so testing must be done with a wide audience.
6. Appropriate feedback, such as dialogue boxes, colours and data changes should be made quickly and clearly, as to make the program more responsive and easy to use.

Social and Ethical Issues

Quite obviously, the management of sensitive patient data results in plentiful social and ethical issues. Patients will not want to use the system unless they are sure that their data is being stored responsibly and securely.

This project aims to reduce social/ethical issues relating to the **sensitivity of patient data/privacy** by:

- (1) Using the more secure `collection.doc().get()` firebase method, rather than solely relying on the `collection.get().then(()=>{ // })` method. This essentially means that specific patient records are sent rather than the entire database, making the database more secure. For example, when patient data is loaded, the patient cannot access other patient's data.
- (2) Using the encrypting JavaScript methods to store sensitive passwords. This means that even if passwords are fetched from the database, they are in a 64 bit encoded form, making password breaches harder. Despite this, it is still possible for users to decrypt passwords if they have the knowledge.
- (3) Storing patient data using an anonymous, random primary keys. This means that someone could not just run a command like `db.collection("patients").doc("//patient name").get()`, rather, they would have to find the primary key, which is considerably more difficult. This random key is also assigned by firebase, and so the computer never has the keys stored. Despite this, it is still technically possible to access patient data, given a 'brute force' approach.

The project also considers **ease of use** by:

- (1) Modularising the program to ensure that patient interfacing modules are simple and streamlined. For example, patients are only given a selection of options that they can use.
- (2) Using consistent UI elements (see *user design specifications* above).

The project considers **availability** by making the application web-based, and such centrally accessible. This means the only software necessary to run the system is a web browser. This is a big issue for a medical program, as medical institutions should be aware of how accessible their services are to the general public, who may not have the expertise to run complicated software.

The project considers **ergonomics** by grouping similar elements together to reduce wrist strain when navigating pages. This will be beneficial, since receptionists may be using this software for hours on end. The grouping of these components will reduce the impacts of RSI (repetitive strain injuries), producing a safer working environment.

Inclusivity is considered by making language clear and easy to read, and using large fonts for important elements (visual disabilities etc...). TTS (text to speech) software could also be employed to make the service accessible to those with visual disabilities.

Needs of the Client

The client has the following needs:

- (1) Allow patients to make bookings remotely, without interacting with a member of staff
- (2) Track bookings for a given day to determine when doctors are required
- (3) Hold patient data such as:
 - a. Healthcare data (Medicare, private health)
 - b. Patient data (name, sex etc...)
 - c. Health data (doctor's notes, medications etc...)
- (4) Manage payment and financial tracking
- (5) Secure, reliable and fast storage of patient data
- (6) Attractive interface for both clients and members of staff

This system, whilst not crucial to the operation of a healthcare facility, is a central part of the patient experience. Such, it can be considered an integral and critical piece of software. For this reason, it is a worthwhile investment for the company.

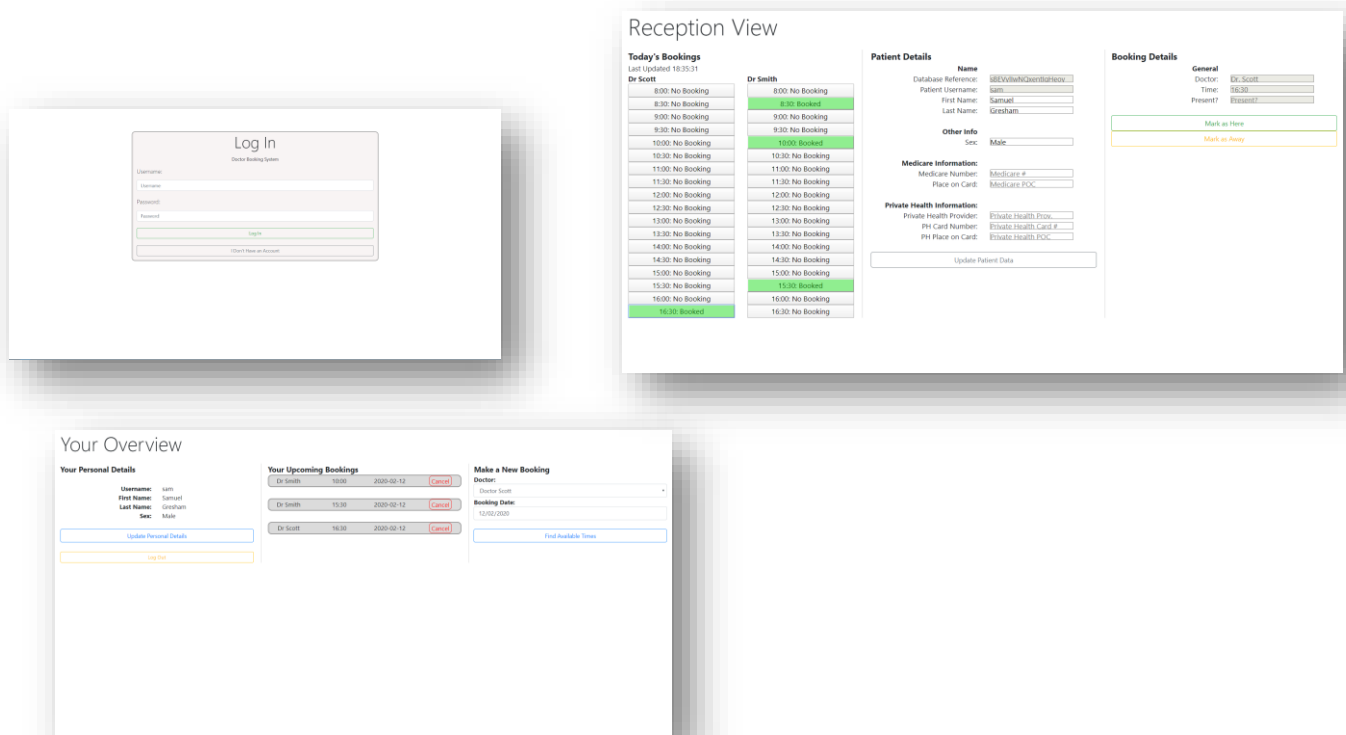
The system offers capabilities for expansion, making it future-proof. This means that the system can be adapted to offer extra functionality in the future. An example of this would be changing the times when which the doctors are available, to allow for extended operating hours.

For the above reasons, this software will satisfy the current needs of the client, and will be expandable for any future needs.

Unfortunately, there are some limitations to the Doctor Booking System that may not meet the needs of the client, a summary of which is below:

- It is difficult to add and remove doctors without editing source code
- There are some stability issues when internet connection is slow (optimisation issues)

Images of the Software

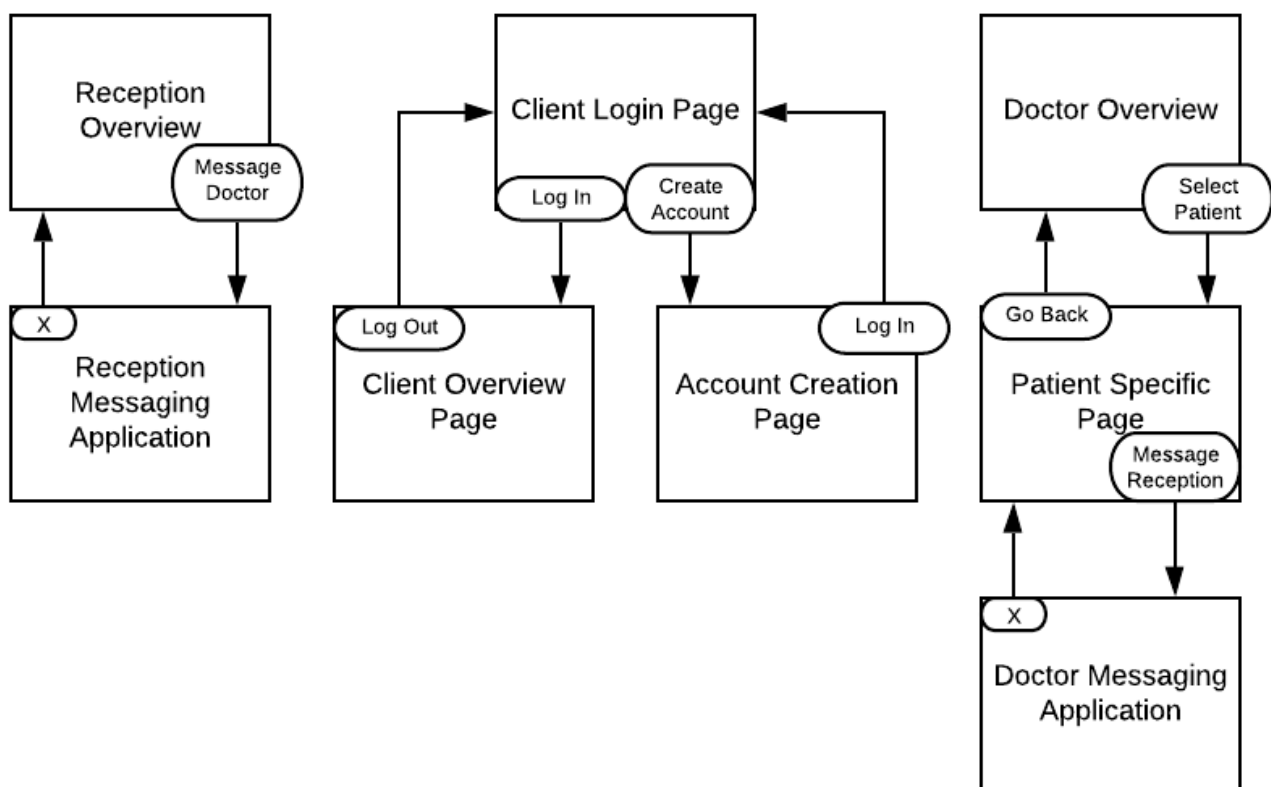


System Modelling

Storyboard

Doctor Booking System is designed to be intuitive and navigationally self-explanatory. For these reasons, the number of separate 'screens' or pages in each module has been intentionally limited, resulting in an ergonomic and seamless user experience.

DBS Storyboard With Navigation Buttons



Above is a storyboard of the three discrete modules, with the relevant navigation buttons supplied.

Screen Designs

Client Module
Login Page

Client Module
overview page

Reception Module
overview Page

Reception + Doctor
Messaging Page.

Doctor Module
overview Page

Above are the screen designs for the Reception, Client and Doctor Modules. Please note that some screens in the storyboard have not been included since they are visually identical to other screens. For example, the account creation screen is identical to the login screen, and the patient specific page is a permutation of the Doctor Overview Page. The messaging dialogue is shared between the Reception and Doctor modules.

Data Dictionary

Collection	Identifier Name	Data Type	Format	Description	Example	Validation	Scope
Bookings	bookingsScott	Array	false or ID	An array of timeslots for that day with booking status	["wJfKEvXZ1aBfBJwX", "false", "..."]	n/a	global
messaging	content	String	unlimited	The content of the message being sent (exists for each message)	"Hello!"	n/a	global
	sender	String	unlimited	The username of the user who sent the message (as above)	"Admin"	n/a	global
	timeSent	Integer	#####	The time in ms of the message being sent. (as above)	158388997208	n/a	global
	currentID	Integer	unlimited	The current message index	12	n/a	global
patients	presentClients	Array	unlimited	Contains all patients who are marked as present at the clinic	["wJfKEvXZ1aBfBJwX", "dT7aAfvA8Aadvb72waz"]	n/a	global
	appts*	Array	YYYY-MM-DD//H//DOCTORNAME	All current and past bookings with this patient	["2020-03-11//6//Dr. Smith", "2020-03-12//10//Dr. Scott"]	must fit format	global
	firstName*	String	unlimited	The first name of the patient.	"Samuel"	cannot be ""	global
	lastName*	String	unlimited	The last name of the patient.	"Gresham"	cannot be ""	global
	medicalDescription*	String	unlimited	Any critical medical information that should be readily available.	"Low blood pressure, susceptible to fainting"	n/a	global
	medicareNumber*	Number	####-####-#	The medicare number of the patient	1234-56789-1	must fit format	global
	medicarePOC*	Number	##	The place on card of the patient.	1	n/a	global
	password*	String	unlimited	A 64-bit encoded version of the user's personal password.	c2Ft	n/a	global
	privateHealthNumber*	Integer	unlimited	The private health number of the user (if the user has PH)	12345678910	n/a	global
	privateHealthPOC*	Integer	unlimited	The place on card of the patient (private health).	1	n/a	global
	privateHealthProvider*	String	unlimited	The private health provider that the patient uses.	"Bupa"	n/a	global
	sex*	String	unlimited	The sex that the patient identifies with.	"Female"	n/a	global
	userName*	String	unlimited	A unique username attributed to the patient's record and account.	"samGresham12"	n/a	global

Note: * indicates that this record exists for all patients

To the left, the data dictionary for the firebase database is provided. Below is a summary of each column:

- (1) **Collection** describes the database collection the data is in.
- (2) **Identifier Name** is the name of the variable.
- (3) **Data Type** is the data type of the variable.
- (4) **Format** provides the format the data is found in. Unlimited means that the data can be of any form.
- (5) **Description** describes the role of each identifier.
- (6) **Example** provides an example of the data.
- (7) **Validation** describes any data validation that the data goes through.
- (8) **Scope** provides the scope of the variable. They are all global since they are stored in a freely accessible database.

IPO Charts

IPO charts have been created for select critical processes.

IPO Chart: Login Procedure		
Input	Process	Output
Username + Password Entered by User	Get all accounts from the database	
	Find an account with the matching username	If there isn't, output an error message
	Encrypt the inputted password using 64 bit encryption	
	Check whether the inputted encrypted password matches the password in the DB	If it doesn't, output an error message
	Set the username to the uID identifier in session storage	Redirect the user to their personalised homepage

IPO Chart: Loading Personalised Page (function populate())		
Input	Process	Output
User ID from session storage	Find user ID in database	If not found, log user out and prompt re-login
First, Last, Sex, Username from DB	Load First, Last, Sex, Username onto page	If no data, output "No Data"
Bookings made by user	Build graphic for each booking made by user	Place graphics on page
	Wait for DB to be updated. When updated, run again.	

IPO Chart: Sending a Message		
Input	Process	Output
Sender and Recipient Names	Determine if a conversation already exists.	
	If a conversation does not exist, create one.	
Message Content	Check that new message is legal (i.e. contains something)	Error message if not
	Push to database, adding one to currentID.	Message to recipient's screen.

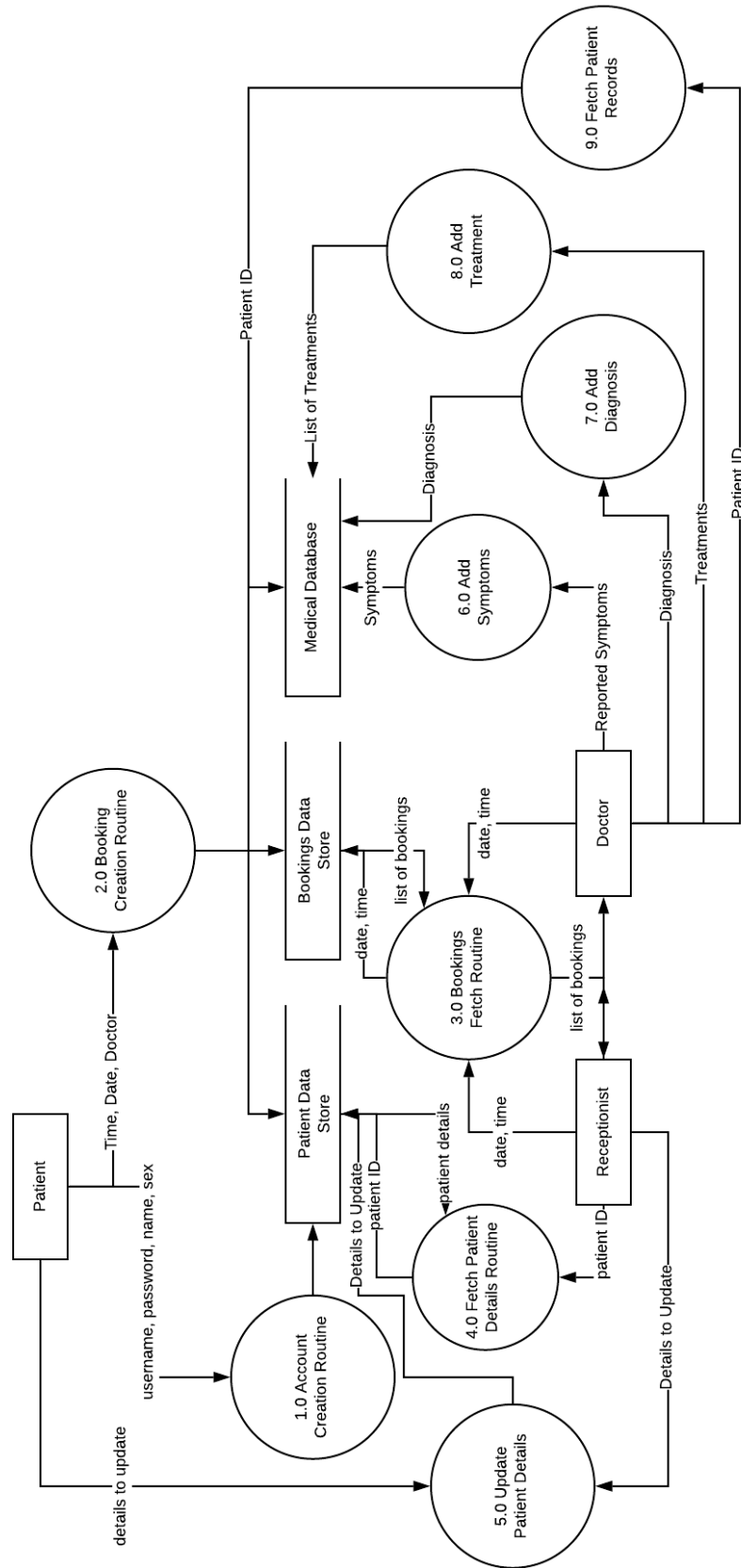
IPO Chart: Loading a Patient in Doctor View		
Input	Process	Output
Patient ID to be loaded	Search in client login database for patient	Patient name, username, sex
Patient Booking Data	Search PBD Database for patient ID, showing upcoming appointments	Upcoming appointments (in next two weeks)
Medical Database Data	Search Medical database for patient. If not found, create entry.	
	Load past unclosed symptoms from medical database.	Unclosed symptoms
	Load past diagnoses from medical database	Past diagnoses
	Load current treatments from medical database	Current treatments

These functions were chosen because they are some of the more nontrivial processes that are undertaken in DBS. Many functions in the DBS codebase are trivial (such as sorts, searches and string manipulation), and hence have not been included.

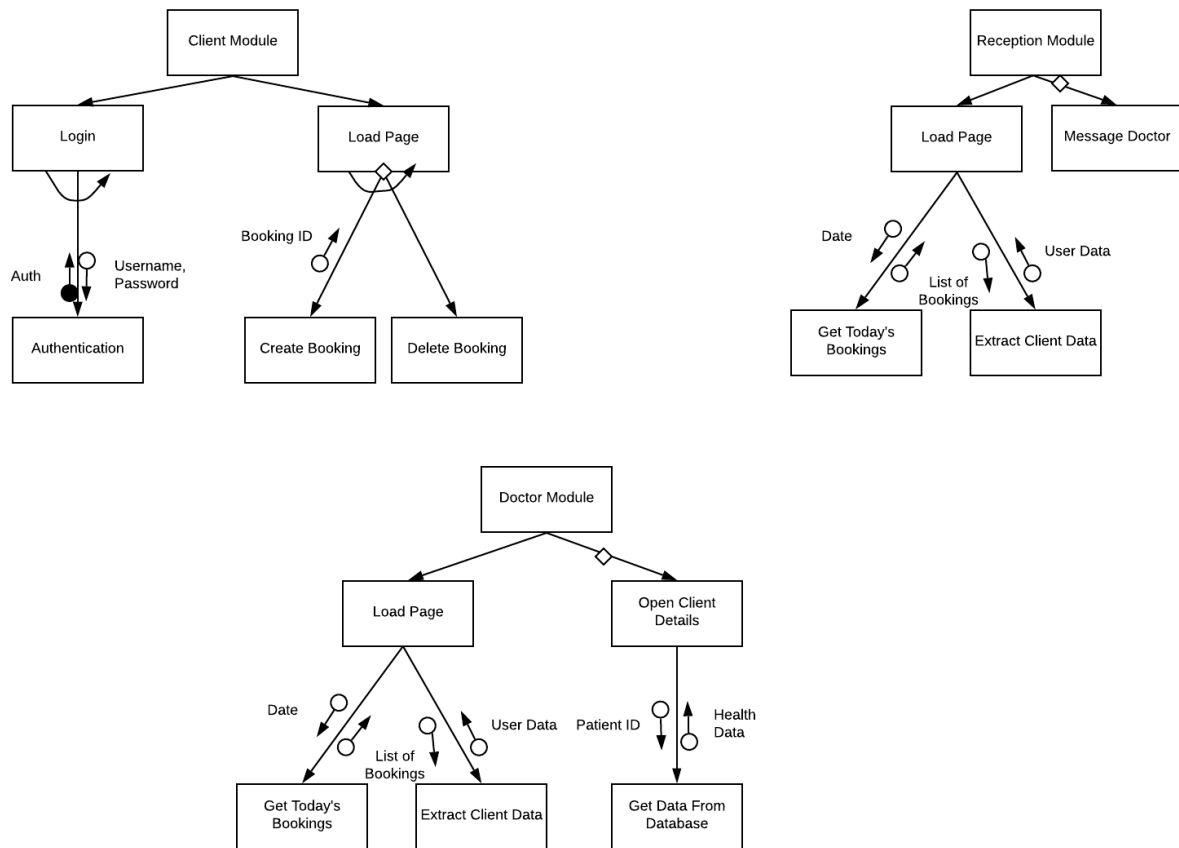
Data Flow Diagram

L1 Data Flow Diagram

Doctor Booking System



Structure Diagram



Above is a structure chart for each module in DBS.

System Flowcharts

