SCHEDULED SMILES

Presented by the Smile Team

The Smile Team

Scrum Master/Product Owner

Sammy Wong

Full Stack Developers

Keav'n Lor & Kyle Tran

Database Developer

Erds Mabilog

Lead Front-End Developer

Dann Manganti

Lead Back-End Developer

Kaylina Kwong

Back-End Developers

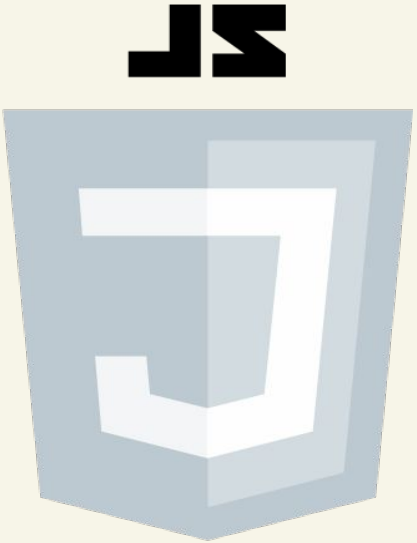
Brandon Casey & John Vue

- Objective
- Proposed Solution
- Tech Stack
- Functional & Non-Functional Requirements
- Stylistic Choices
- Design Patterns
- Highlights

- The purpose of this project is to develop a structurally organized website for its client that streamlines the gathering of clinical information for dental clinics.
- The goal of the project is to create a centralized system which manages patient records, appointment scheduling, treatment history, and dental staff performance.
- The goal for the website is to have user-friendly framework that is easily navigable for clinical staff and patients as well as provide an easy method to communicate with staff and access information for all users.

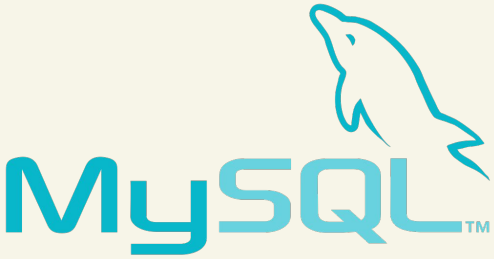
- The Scheduled Smiles website aims to make the process of scheduling a dental appointment as simple and efficient as possible.
- Users will be able to select whatever available date and time they wish and communicate with a dental assistant before and after appointments.
- Staff will be able to keep track of patient appointments to schedule and reschedule as needed, while keeping records of what sorts of treatments were given during said appointment.

Front-End



Back-End

Database



Operating System

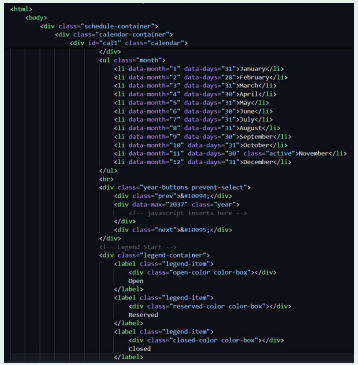
Tech Stack

Front-End

Back-End

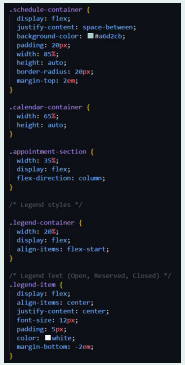
Database

HTML



```
<html>
    <body>
        <div class="schedule-container">
            <div class="calendar-container">
                <div id="cal1" class="calendar">
                    </div>
                    <ul class="month">
                        <li data-month="1" data-days="31">January</li>
                        <li data-month="2" data-days="28">February</li>
                        <li data-month="3" data-days="31">March</li>
                        <li data-month="4" data-days="30">April</li>
                        <li data-month="5" data-days="31">May</li>
                        <li data-month="6" data-days="30">June</li>
                        <li data-month="7" data-days="31">July</li>
                        <li data-month="8" data-days="31">August</li>
                        <li data-month="9" data-days="30">September</li>
                        <li data-month="10" data-days="31">October</li>
                        <li data-month="11" data-days="30" class="active">November</li>
                        <li data-month="12" data-days="31">December</li>
                    </ul>
                    <br>
                    <div class="year-buttons prevent-select">
                        <div class="prev">&#10094;</div>
                        <div data-num="2017 class="year">
                            <!-- javascript inserts here -->
                        </div>
                        <div class="next">&#10095;</div>
                    </div>
                        Legend Start -->
                    <div class="legend-container">
                        <label class="legend-item">
                            <div class="open-color color-box"></div>
                            Open
                        </label>
                        <label class="legend-item">
                            <div class="reserved-color color-box"></div>
                            Reserved
                        </label>
                        <label class="legend-item">
                            <div class="closed-color color-box"></div>
                            Closed
                        </label>
```

CSS



```
.schedule-container {
  display: flex;
  justify-content: space-between;
  background-color: #a6d2cb;
  padding: 20px;
  width: 85%;
  height: auto;
  border-radius: 20px;
  margin-top: 2em;
}

.calendar-container {
  width: 65%;
  height: auto;
}

.appointment-section {
  width: 35%;
  display: flex;
  flex-direction: column;
}

/* Legend styles */

.legend-container {
  width: 20%;
  display: flex;
  align-items: flex-start;
}

/* Legend text (Open, Reserved, Closed) */
.legend-item {
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 12px;
  padding: 5px;
  color: white;
  margin-bottom: -2em;
}
```

JavaScript

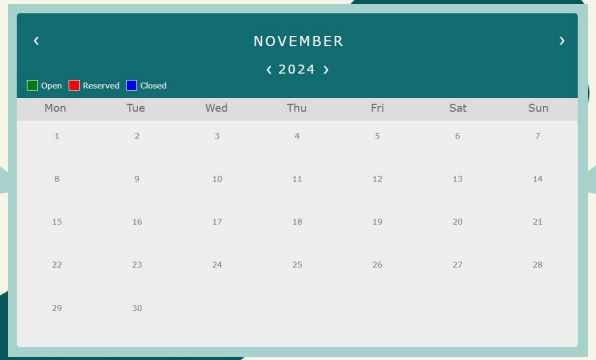
```
function loadCalendarYear(calendarElement) {
    // Our timestamps in backend only support dates up to Jan, 2038 -Kyle
    let yearElement = calendarElement.querySelector(":scope .year")
    yearElement.innerHTML = getYear()
}

function nextYear(calendarElement) {
    let yearElement = calendarElement.querySelector(":scope .year")
    const activeYear = parseInt(yearElement.innerHTML)
    yearElement.innerHTML = (activeYear >= yearElement.dataset.max) ?
        getYear() : activeYear + 1;
}

function prevYear(calendarElement) {
    let yearElement = calendarElement.querySelector(":scope .year")
    const activeYear = parseInt(yearElement.innerHTML)
    const currentYear = getYear()
    yearElement.innerHTML = (activeYear <= currentYear) ?
        yearElement.dataset.max : activeYear - 1;
}

function nextMonth(calendarElement) {
    let monthElementsContainer = calendarElement.querySelector(":scope .month")
    let oldActiveMonth = monthElementsContainer.querySelector(":scope .active")
    let newActiveMonth = (oldActiveMonth.nextElementSibling == null) ? monthElementsContainer.firstElementChild : oldActiveMonth.n
    oldActiveMonth.classList.remove("active")
    newActiveMonth.classList.add("active")
}

function prevMonth(calendarElement) {
    let monthElementsContainer = calendarElement.querySelector(":scope .month")
    let oldActiveMonth = monthElementsContainer.querySelector(":scope .active")
    let newActiveMonth = (oldActiveMonth.previousElementSibling == null) ? monthElementsContainer.lastElementChild : oldActiveMont
    oldActiveMonth.classList.remove("active")
    newActiveMonth.classList.add("active")
}
```



Front-End

Back-End

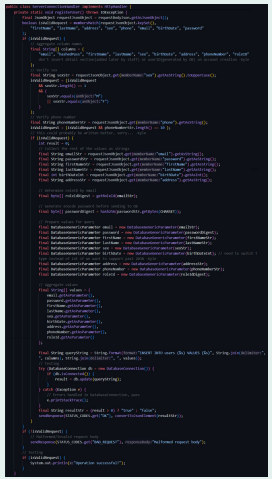
Database

No CSS

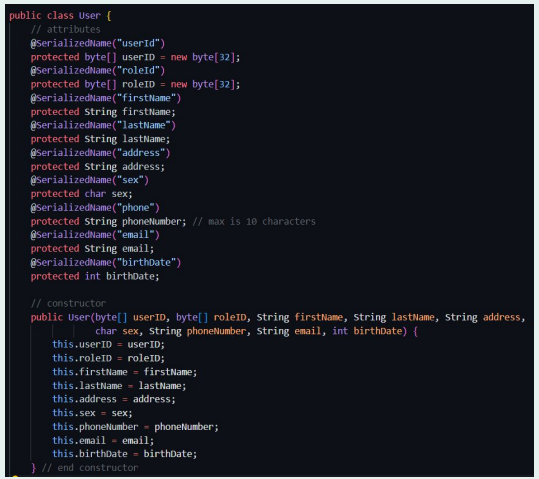
With CSS

Front-End | Back-End | Database

Registering a User



Server

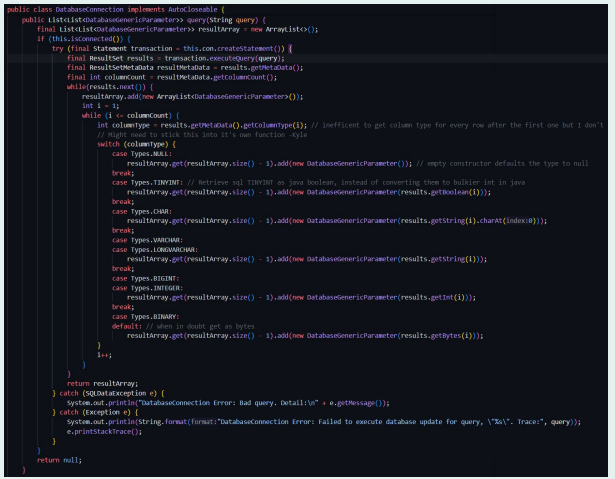


```java
public class User {
    // attributes
    @SerializedName("userId")
    protected byte[] userID = new byte[32];
    @SerializedName("roleId")
    protected byte[] roleID = new byte[32];
    @SerializedName("firstName")
    protected String firstName;
    @SerializedName("lastName")
    protected String lastName;
    @SerializedName("address")
    protected String address;
    @SerializedName("sex")
    protected char sex;
    @SerializedName("phone")
    protected String phoneNumber; // max is 10 characters
    @SerializedName("email")
    protected String email;
    @SerializedName("birthDate")
    protected int birthDate;

    // constructor
    public User(byte[] userID, byte[] roleID, String firstName, String lastName, String address,
                char sex, String phoneNumber, String email, int birthDate) {
        this.userID = userID;
        this.roleID = roleID;
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.sex = sex;
        this.phoneNumber = phoneNumber;
        this.email = email;
        this.birthDate = birthDate;
    } // end constructor
```

User Class

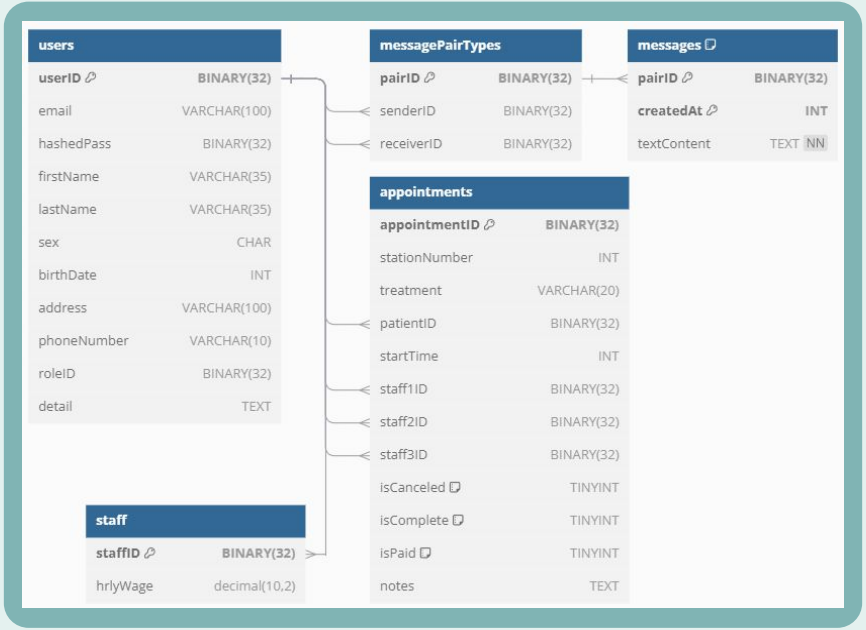
User Methods



```java
public class DatabaseConnection implements AutoCloseable {
    public List<List<DatabaseGenericParameter>> query(String query) {
        final List<List<DatabaseGenericParameter>> resultArray = new ArrayList<>();
        if (this.isConnected()) {
            try (final Statement transaction = this.con.createStatement()) {
                final ResultSet results = transaction.executeQuery(query);
                final ResultSetMetaData resultMetaData = results.getMetaData();
                final int columnCount = resultMetaData.getColumnCount();
                while(results.next()) {
                    resultArray.add(new ArrayList<DatabaseGenericParameter>());
                    int i = 1;
                    while (i <= columnCount) {
                        int columnType = results.getMetaData().getColumnType(i); // inefficent to get column type for every row after the first one but I don't
                        // might need to stick this into it's own function -kyle
                        switch (columnType) {
                            case Types.NULL:
                                resultArray.get(resultArray.size() - 1).add(new DatabaseGenericParameter()); // empty constructor defaults the type to null
                            break;
                            case Types.TINYINT: // Retrieve sql TINYINT as java boolean, instead of converting them to bulkier int in java
                                resultArray.get(resultArray.size() - 1).add(new DatabaseGenericParameter(results.getBoolean(i)));
                            break;
                            case Types.CHAR:
                                resultArray.get(resultArray.size() - 1).add(new DatabaseGenericParameter(results.getString(i).charAt(index:0)));
                            break;
                            case Types.VARCHAR:
                            case Types.LONGVARCHAR:
                                resultArray.get(resultArray.size() - 1).add(new DatabaseGenericParameter(results.getString(i)));
                            break;
                            case Types.BIGINT:
                            case Types.INTEGER:
                                resultArray.get(resultArray.size() - 1).add(new DatabaseGenericParameter(results.getInt(i)));
                            break;
                            case Types.BINARY:
                            default: // idrk in doubt get as bytes
                                resultArray.get(resultArray.size() - 1).add(new DatabaseGenericParameter(results.getBytes(i)));
                        }
                        i++;
                    }
                }
                return resultArray;
            } catch (SQLDataException e) {
                System.out.println("DatabaseConnection Error: Bad query. Detail:\n" + e.getMessage());
            } catch (Exception e) {
                System.out.println(String.format(format:"DatabaseConnection Error: Failed to execute database update for query, \"%s\". Trace:", query));
                e.printStackTrace();
            }
        }
        return null;
    }
```

DatabaseConnection

Database Schema



Front-End | Back-End | Database

User

userID	email	hashedPass	firstName	lastName	sex	birthDate	address	phoneNumber	roleID	detail
BLOB	JaySohn@email.com	BLOB	Jay	Sohn	M	948614400	123 Address Ct	9163597437	BLOB	NULL
BLOB	StephFu@scheduledsmiles.com	BLOB	Steph	Fu	F	567648000	456 Address Ave	9169678121	BLOB	NULL
BLOB	AnnieYeager@scheduledsmiles.adm.com	BLOB	Annie	Yeager	F	397526400	789 Address Dr	9167954329	BLOB	NULL
BLOB	StewartFerris@scheduledsmiles.com	BLOB	Stewart	Ferris	M	905990400	456 Address Blvd	9160258429	BLOB	NULL
BLOB	JohnDoe@email.com	BLOB	John	Doe	M	759283200	123 Address Lane	1234567890	BLOB	NULL
BLOB	AdamMinh@scheduledsmiles.adm.com	BLOB	Adam	Minh	M	169171200	789 Address Way	9166534124	BLOB	NULL
BLOB	EliseFlossmore@scheduledsmiles.com	BLOB	Elise	Flossmore	F	774705600	456 Address Dr	9165592063	BLOB	NULL
BLOB	JaneDoe@email.com	BLOB	Jane	Doe	F	826761600	123 Address Lane	3141592654	BLOB	NULL

Staff

staffID	hrlyWage
BLOB	80.00
BLOB	35.00
BLOB	80.00
BLOB	35.00
BLOB	80.00

Appointment

appointment...	stationNumb...	treatment	patientID	startTime	staff1ID	staff2ID	staff3ID	isC...	isComplete	isPaid	notes
BLOB	1	Emergency	BLOB	1732294800	BLOB	BLOB	BLOB	0	1	0	Operation Successful, Patient must...
BLOB	2	Checkup	BLOB	1731092400	BLOB	NULL	NULL	1	0	0	Patient cannot come in, rescheduled
BLOB	1	Checkup	BLOB	1731090600	BLOB	NULL	NULL	0	1	0	NULL
BLOB	1	Checkup	BLOB	1731693600	BLOB	NULL	NULL	0	1	0	Patient needs further care
BLOB	1	Filling	BLOB	1731690000	BLOB	BLOB	NULL	0	1	0	Operation Successful, Patient must...
BLOB	1	Checkup	BLOB	1731087000	BLOB	NULL	NULL	0	1	0	Patient needs further care

Sender & Receiver

pairID	senderID	receiverID
BLOB	BLOB	BLOB
BLOB	BLOB	BLOB
BLOB	BLOB	BLOB
BLOB	BLOB	BLOB
BLOB	BLOB	BLOB
BLOB	BLOB	BLOB

Messages

pairID	createdAt	textContent
BLOB	1732081671	MESSAGE1-1
BLOB	1732173270	MESSAGE1-2
BLOB	1732095086	MESSAGE3-1
BLOB	1732162807	RESPONSE1-1
BLOB	1732508806	RESPONSE1-2
BLOB	1732480152	MESSAGE2-1
BLOB	1732539566	RESPONSE2-1
BLOB	1732561661	RESPONSE3-1

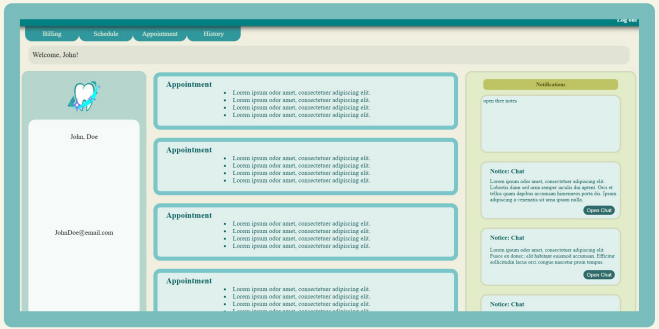
Functional Requirements & Non-Functional Requirements

- FR 1.0: User Based Functionality and Records
- FR 2.0: Patient-Doctor Chatbox
- FR 3.0: Patient Billing
- FR 4.0: Employee Payroll
- FR 5.0: Session Tracking Form
- FR 6.0: User Login
- FR 6.1: User Registration
- FR 7.0: Appointment Scheduler
- FR 7.1: Patient: View/Request Schedule Times
- FR 7.2: Staff: Creating/Managing Schedule Availability

- NFR 1.0 User Record Database
- NFR 1.1 Securing Database
- NFR 2.0 User Communication
- NFR 3.0 Patient Information Validation
- NFR 4.0 Staff Information Validation
- NFR 5.0 Session Tracking Form
- NFR 6.0 User Login
- NFR 6.1 User Insert to Database
- NFR 7.0 Appointment Scheduler

FR 1.0
User Dashboard

- **Purpose**: This requirement allows the user to access records relating to them (i.e scheduled appointments, previous treatments).
- **Processing**: Upon selecting a desired link, the website will redirect to the corresponding webpage. If such webpage requires displaying specific data, a SQL query will be sent via server request to read user specified information from the remote database on AWS.



NFR 1.0 User Record Database: ✕

The database facilitating these requests will utilize MySQL and hosted remotely on AWS server. Queries into the database will be sent using server calls initiated from the backend.

NFR 1.1 Securing Database: ✕

To prevent information breaches, sensitive information will be secured through the hash algorithm *sha256*.

FR 2.0
Patient-Doctor Chatbox

- **Purpose**: This allows patients to directly communicate with their doctor(s) and vice versa.
- **Processing**: Both the sender and receiver will have dedicated IDs, the message sent between users are stored in the database alongside the timestamp that it was sent to be ordered chronologically. Viewing a chat log will query the database to retrieve up to n^{th} number of message, decided by optimal runtime efficiency.

NFR 2.0 User Communication: ✕

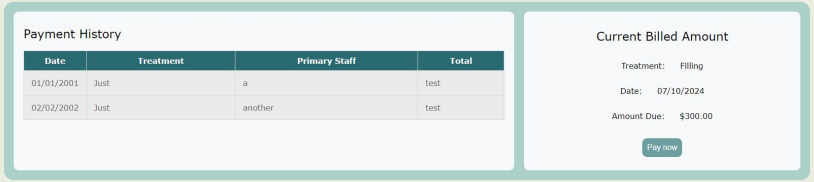
At account creation, all users will be given a unique ID for linking communication. Messages will be inserted into a log on the database, and on webpage load, the latest log of that chat will be presented on the page.

FR 3.0
Patient Billing

- **Purpose**: This requirement allows patients to receive, view and pay their billed amounts.
- **Processing**: Upon page opening, a server request will send a query into the database selecting all recent appointments for that user up the n^{th} recent appointment, decided by optimal runtime efficiency. All appointments will be displayed into the payment history alongside that appointment's respective cost, treatment, and date. Selected appointments with the "paid" status as *false* will be displayed separately with the option for user to select "pay".

NFR 3.0 Patient Information Validation: ✕

Patient (role) identification is initialized after cross-reference of database for matching credentials. Uses include validation of payment information and appointment scheduling.

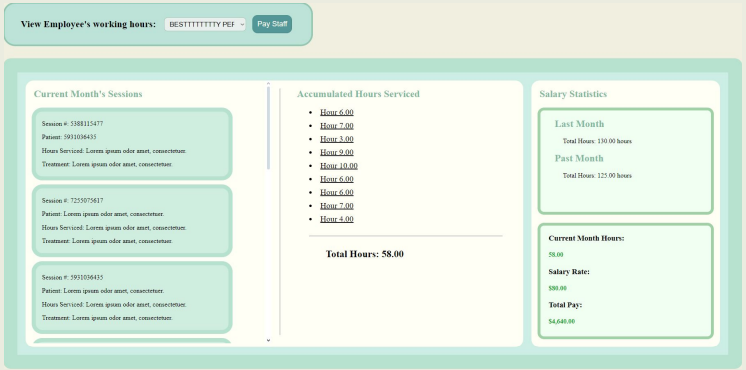


FR 4.0
Employee Payroll

- **Purpose**: This requirement allows the user (staff) pay to be calculated and viewed based on hours working.
- **Processing**: Each staff associated will have their hourly rate and hours worked (found by querying database for all recent appointments attached to that staff) used to calculate their expected pay.

NFR 4.0 Staff Information Validation: ✕

Staff (role) identification is initialized after cross-reference of database for matching credentials. Uses encapsulate staff specific view permissions such as payroll and alternate appointment page.

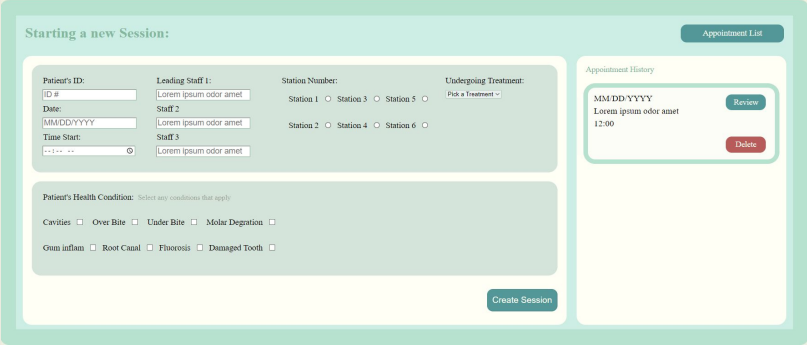


FR 5.0
Session Tracking Form

- **Purpose**: A web page (form) for storing records per appointment. The page contains patients' health information, staff's notes, treatment type, and any other relevant information to be stored in the database.
- **Processing**: This webpage will have pre-determined keywords and will be saved with a unique form identification based on its entry number (generated in backend) into the MySQL database, saving account IDs associated with the session (both staff and patient). Keywords and key information in this form will be tracked and parsed into separate pieces of information for data storing.

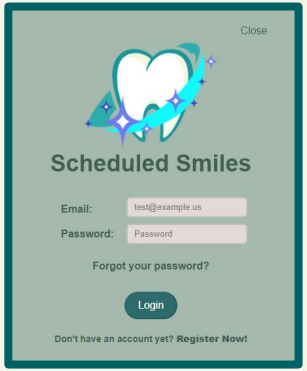
NFR 5.0 Session Tracking Form: ✕

Checks and verifies if scheduled appointment sessions are available and then completed.



FR 6.0
User Login

- **Purpose**: This requirement will allow users to log in and access their personal accounts.
- **Processing**: User inputted credentials will be cross-referenced with a hashed version of the password stored in the database to verify correct user login. On verification, return account key (backend data type) associated with that specific user.



NFR 6.0 User Login:

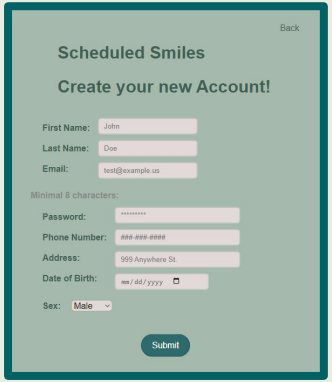
Logging in will compare credentials already stored in the database. The user will input their specific username and password that will be used in a search through the database. The database will check if a username matching the inputted value exists, it will then require hashing the password and matching it to a password hash stored under the corresponding username in the database. If it matches, the user will be logged in.

FR 6.1
User Registration

- **Purpose**: This requirement will allow users to generate a unique account key to store and access related information, so that history associated with the user is trackable.
- **Processing**: Upon user input, credentials are cross-referenced from database for a search of duplicate credentials by querying previously created accounts. If no such replica is found, initiate account creation and store all information in database, as well as generate non-occupied account ID to be associated with user account.

NFR 6.1 User Insert to Database:

Registering new users will add them as a new entry to the database. They will be inserted in an organized data structure based on a newly created and unique ID.



FR 7.0
Appointment Scheduler

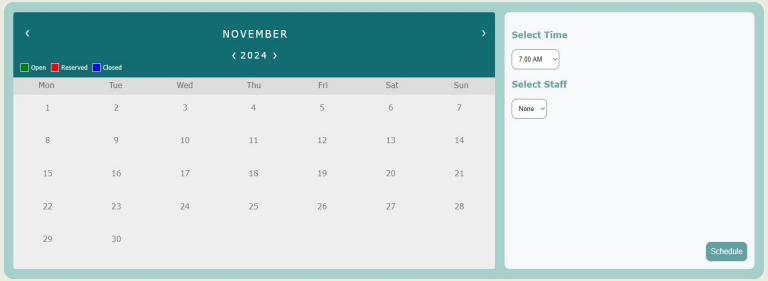
- **Purpose**: This requirement allows patients and staff to schedule an appointment at an available date and time directly through the website.
- **Processing**: Upon receiving the user's account key, a server request will query the MySQL database for the account's role, displaying role specific processes. A separate server request will query the database for all non occupied times available for the user to schedule their appointment.

NFR 7.0 Appointment Scheduler:

When scheduling an appointment, clear, visual user feedback will be given based on account privilege. We will display specific color schemes or patterns to determine which specific time frames are available and allowed. An easily accessible legend will also be displayed for clarification on the meanings of each color and pattern.

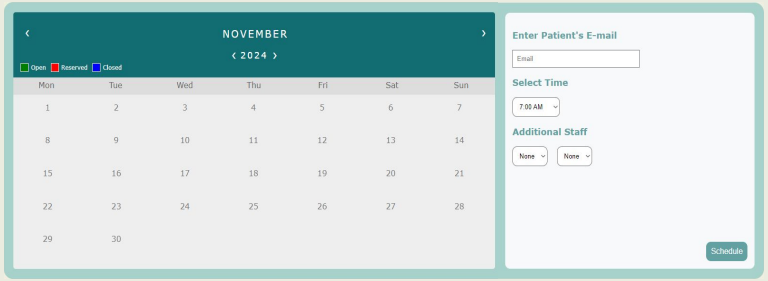
FR 7.1
Patient: View/Request Schedule Times

- **Purpose**: This view will allow patients to select from available time frames and request to reserve an appointment on the specified timeframe.
- **Process**: The timeframe will be entered into the website and database and be recorded. The reservation will occupy a time slot of 1 (one) hour. The account ID and user inputted information will be used to recognize who the appointment is for and retrieving history relating to that user.



FR 7.2
Staff: Creating/Managing Schedule Availability

- **Purpose**: Staff will be able to create availability of appointments by whitelisting timeframes allowed for appointment from a list and calendar.
- **Process**: Selected time frames will be sent to the website and processed. These timeframes will create new overwritable data entries and store them in the database. If there are duplicate entries, they will overwrite the old ones. Recorded data entries will then be retrieved and displayed on the Calendar pages for general view.



Stylistic Choices

Design Patterns

Highlights

HTML & CSS

- Indentation of 4 spaces
- Meaningful Class Names
- Class Names are Short and Brief

```html
<!doctype html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title>Scheduled Smiles</title>
        <link rel="stylesheet" href="./css/header.css">
        <link rel="stylesheet" href="./css/dashboard.css">
        <script src ="./js/navigationTab.js"></script>
        <script src="./js/overviewBox.js"></script>
    </head>
```

```css
.signInBox{
    display: none;
    position: absolute;
    left: 50%;
    top: 50%;
    -webkit-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
}

.wrapSignIn {
    background-color: #bdc2b8;
    height: 600px;
    width: 600px;
    text-align: center;
    padding-top: 30px;
    justify-content: center;
    border-radius: 10px;
}

.Logo{
    justify-content: center;
}
```

Java

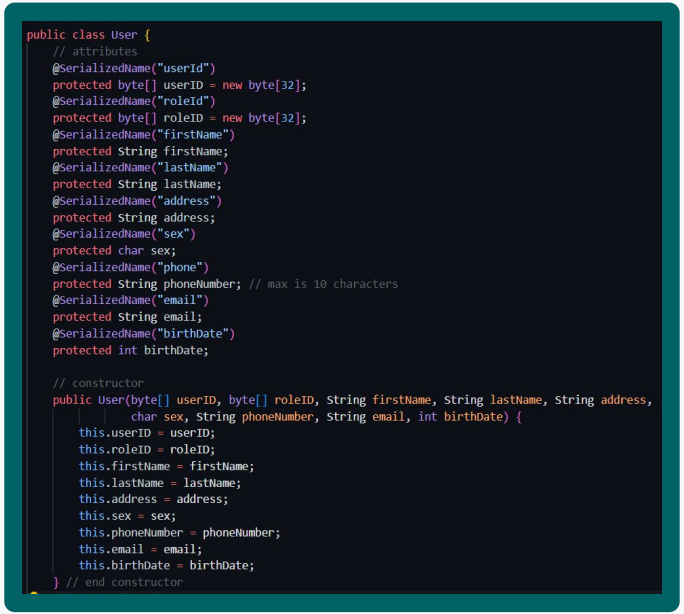
- Camel Case
- Kernighan & Ritchie Style for Curly Brackets
- Indentation of 4 spaces

```java
public String getAsParameter() {
    if(isBoolean()) {
        return String.format(format:"TINYINT(%s)", (Boolean.parseBoolean(stringValue)) ? 1 : 0);
    } else if(isBytes()) {
        return String.format(format:"UNHEX(\'%s\')", stringValue);
    } else if(isInteger()) {
        return stringValue;
    } else { // must be string
        return String.format(format:"\'%s\'", stringValue);
    }
}
```

```java
public boolean getAsBoolean() throws UnsupportedOperationException {
    if (this.isBoolean()) {
        return Boolean.parseBoolean(stringValue);
    }
    throw new UnsupportedOperationException(message:"Value is not of type boolean");
}
public String getAsString() throws UnsupportedOperationException {
    if (this.isString()) {
        return stringValue;
    }
    throw new UnsupportedOperationException(message:"Value is not of type int");
}
```

Creational Design Pattern

- Singleton
  - Guarantees the ability to create a single instance of the User object
    - Creates a single instance of User, then calls from other classes will reference that single instance
  - Eager Initialization
    - Creates the instance as soon as the class begins to run



```
public class User {
    // attributes
    @SerializedName("userId")
    protected byte[] userID = new byte[32];
    @SerializedName("roleId")
    protected byte[] roleID = new byte[32];
    @SerializedName("firstName")
    protected String firstName;
    @SerializedName("lastName")
    protected String lastName;
    @SerializedName("address")
    protected String address;
    @SerializedName("sex")
    protected char sex;
    @SerializedName("phone")
    protected String phoneNumber; // max is 10 characters
    @SerializedName("email")
    protected String email;
    @SerializedName("birthDate")
    protected int birthDate;

    // constructor
    public User(byte[] userID, byte[] roleID, String firstName, String lastName, String address,
                char sex, String phoneNumber, String email, int birthDate) {
        this.userID = userID;
        this.roleID = roleID;
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.sex = sex;
        this.phoneNumber = phoneNumber;
        this.email = email;
        this.birthDate = birthDate;
    } // end constructor
```

Structural Design Pattern

- Decorator
  - Modifies the User object functionality at runtime
    - Allows us to assign extra behaviors such as hourly rate to User objects without breaking the code

```java
public class Admin extends User {
    // Attributes
    @SerializedName("hourlyRate")
    private double hourlyRate;
    // constructor
    public Admin(byte[] userID, String firstName, String lastName, String address, char sex, String phoneNumber, String email, int
        super(userID, ROLE_IDS.get("Admin"), firstName, lastName, address, sex, phoneNumber, email, birthDate);
        this.hourlyRate = hourlyRate;
    } // end constructor

    // Getter for hourlyRate
    public double getHourlyRate() {
        return hourlyRate;
    }

     // Setter for hourlyRate
    public void setHourlyRate(double hourlyRate) {
        this.hourlyRate = hourlyRate;
    }
}
```

```java
public class Staff extends User {
    // Private Attributes
    @SerializedName("hourlyRate")
    private double hourlyRate;

    // Contructor
    public Staff(byte[] userID, String firstName, String lastName, String address,
                 char sex, String phoneNumber, String email, int birthDate, double hourlyRate){
      super(userID, ROLE_IDS.get("Staff"), firstName, lastName, address, sex, phoneNumber, email, birthDate);
      this.hourlyRate = hourlyRate;
    }

    // Getter for hourlyRate
    public double getHourlyRate(){
      return hourlyRate;
    }

    // Setter for hourlyRate
    public void setHourlyRate(double hourlyRate){
      this.hourlyRate = hourlyRate;
    }
}
```

- Minimal Overhead
- Inhouse Source Code
- Optimal Runtime
- Flexible server
- Secure Encryption
- User-Centric Experience