CSC 191

Fall 2013

Team Sierra

Alex Chernyak
Joubin Jabbari
Kyle Matz
Mike McParland
Scott Livingston
Serge Lysak

Software Design Specification Document

Tuesday September 19th, 2013

1. INTRODUCTION

This is the software design specification document for the Salon Scheduling System project sponsored by Lisa Sigurdson. This project is being undertaken by the Team Sierra development team. The team is comprised of undergraduate students majoring in Computer Science at California State University, Sacramento. The team members are enrolled in a two-semester senior project course required of all undergraduate majors. Successful delivery of the desired software product will fulfill the senior project requirement for the student team members.

PROJECT SPONSOR
Contact Person's Name: Lisa Sigurdson
Title: Owner and Hair Stylist
Organization Name: Dragonfly Salon and Boutique
Contact information: Alayna.Sigurdson@gmail.com
DEVELOPMENT TEAM:
"Team Sierra":

- Alex Chernyak
- Joubin Jabbari
- Kyle Matz
- Mike McParland
- Scott Livingston
- Serge Lysak

Team Contact Info: TeamSierra@googlegroups.com

1.1 Purpose

The purpose of this document is to introduce the reader to the design specifications of this system. As a general rule, it is assumed that the reader of this document has been introduced to our project and is familiar with overall system.

1.2 Scope

In this document, Team Sierra will outline the design of the system. Based on the design, a website will be developed to cover the scope of the operation of the business. Diagrams will be developed to show how the system is designed to meet requirements on the front end and back end. Resulting in a clear understanding on the behalf of the sponsor of how this system will be designed.

1.3 Definitions, Acronyms and Abbreviations

This section will include the various definitions, acronyms, and abbreviations that we will use throughout this document. This section can be referenced throughout reading the document to further explain unclear words.

1.3.1 Definitions

Actor	Anything that interfaces with the system
App Server	A server that runs the web application.
Database	Organized collection of data.
Entity	An organized object describing related data in the database.
Executable Code	The form of the software that can be ran on the computer.
Faculty Advisor	The senior project teams have a CSc faculty member assigned to them to advise the team on various aspects throughout the course of the project.
Gigabyte	A large unit of data storage.
Javadoc	Javadoc is a documentation standard that generates HTML files with descriptions of the code to be view using any web browser.
MySQL	A database program used to hold and organize data in a relational form.
Non static information	Information that changes.
Project Charter	Provides a detailed approach to the project.
Project Documentation	Includes all of the following documents that will be created for the software system over the course of the project.
Project Management Plan	Provides detailed information on how the project will be managed and resources will be obtained.
Sanitation	Process of validating and removing invalid input data.
Software Design Specification	Provides a high level view of the design and interfaces that the project is based on.
Software Requirements Specifications	Describe the behavior, scope and dependencies of the system to be developed. Provide explanation on how it will be used.
Source Code	A collection of computer instructions written in human-readable computer language.
Stylist	Refers to both hair stylist and nail technician

System Test Specification	Sequence of tests to be conducted to test the system.
System Test Report	Results from the System Test Specification.
Use Case	Describes things that an actor wants to do to the system.
User Manual	A complete guide on how to install and operate the system.

1.3.2 Acronyms

CD	Compact Disk
CSc	Computer Science
ERD	Entity Relation Diagram
GB	Gigabyte
IDE	Integrated Development Environment
MVC	Model View Controller
SPMP	Software Project Management Plan
SRS	Software Requirement Specifications
UI	User Interface

1.3.3 Abbreviations

App	Application
Mgt	Management
Mtg	Meeting
Req't	Requirement
Spec	Specification

1.4 References

Dr. Buckleys SDS template provided at CSU, Sacramento Fall 2013.

1.5 Overview of Contents of the Document

This section includes the overview of each main topic of this document. The overview explains what each section includes and any other details about the sections.

1.5.1 Architectural Design

This section is dedicated to the design of the software including all components. In this section, the components of the system are referred to as actors. The figures will give an easy to understand clear view of the architecture design.

1.5.2 Interface Design

This section provides some screenshots and a small caption detailing the features of a given page. The objective is to provide a general feel to the application from the user's view.

1.5.3 Database Schema

This section provides the translation of the informational model contained in the Software Requirements Specification (SRS) into a relational database.

1.5.4 Component Design Specifications

This section provides the detailed description for the design of the software. The section begins with a traceability matrix that map use cases to the web page and the components needed to serve the web pages. The following table shows a brief overview of how components within the system are related.

1.5.5 Performance Analysis

This section provides details on any performance issues or constraints that may have arisen during the design and implementation phase.

1.5.6 Resource Estimates

Provides estimates for and and all resources needed by the system.

1.5.7 Software Requirements Traceability Matrix

The matrix relates the design components to their associated requirements by listing the subsection numbers in this document and the associated subsection numbers in the Software Requirements Specification Document.

2. ARCHITECTURAL DESIGN

This section is dedicated to the design of the software including all components. In this section, the components of the system are referred to as actors. The figures will give an easy to understand clear view of the architecture design.

2.1 Hardware Architecture

Although we are using a specific set of hardware to test and set of the system, this exact

hardware is not required. Any hardware system that will run an App server, Web server, and a mongo database can be used to run our application. Figure 1 shows an overview of the software that will be running on the hardware system.

There will be one server that will host the application and will allow any internet connected client device to access it.

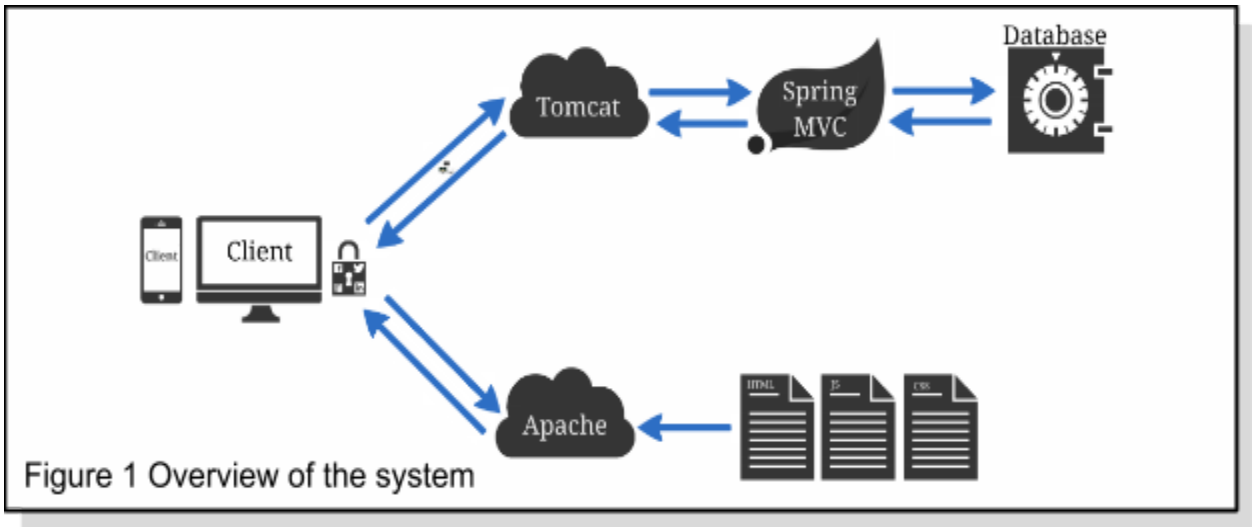


Figure 1 Overview of the system

2.2 Software Design Architecture.

- Backend
  - The backend will consist of two major components.
    - Spring MVC + MongoDB (Figure 2)
    - Apache Tomcat Server (Figure 3)

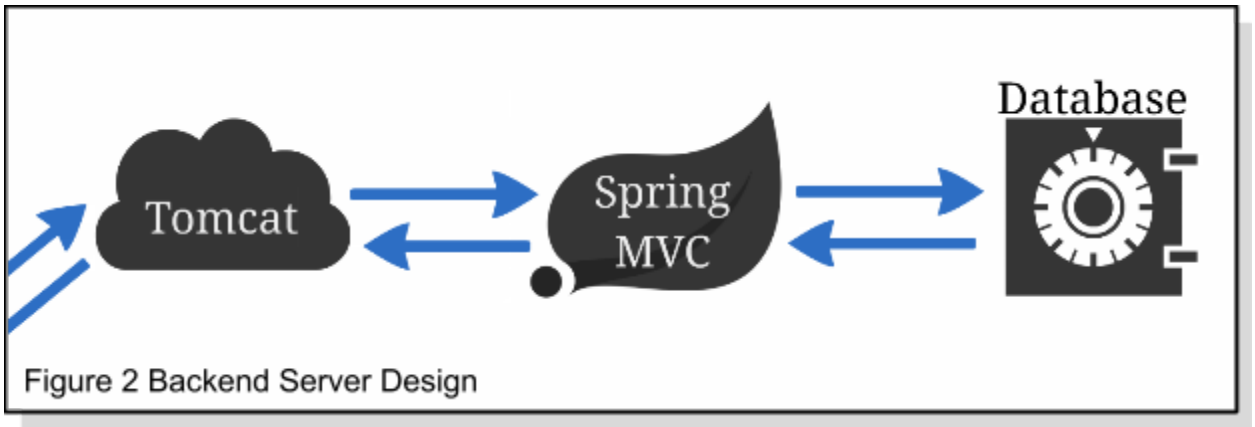


Figure 2 Backend Server Design

In this implementation, the backend will consist of an app server and a model view controller that will communicate with our database to provide business logic.

2.2.1 Presentation Layer

- FrontEnd
    - The presentation layer is the server that will provided a series of static pages that will be the styling of the application.

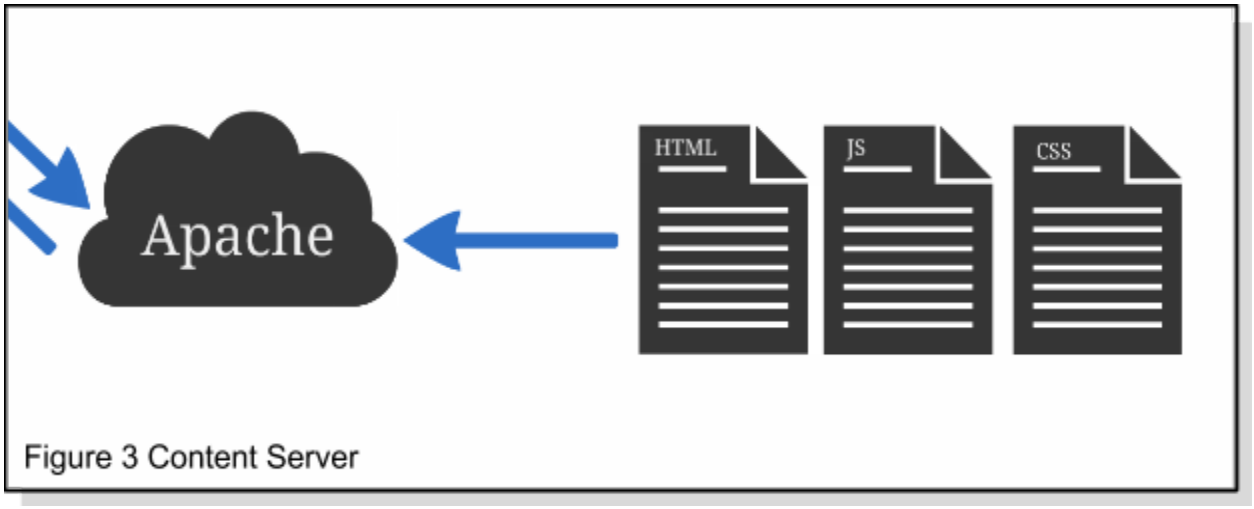


Figure 3 Content Server

2.2.2 Business Logic Layer

Once the backend (server side business logic) and the front end (styling of the application) have been delivered to the user. The business logic layer is an aggregated series of logic that allows the user to interact with the website in a logical manner.

2.2.3 Data Management Layer

The data management layer is a part of the backend described in the Software Design Architecture (2.2 Backend).

The model view control (MVC) provided by Spring MVC provides an abstraction layer for data management. The aggregation of the data from Mongo (database) perspective is a series of Key Value pairs defined in the Spring MVC code and retained in the business logic layer (2.2.2).

3. INTERFACE DESIGN

This section provides some screenshots and a small caption detailing the features of a given page. The objective is to provide a general feel to the application from the users view.

3.1 Login Interface

Users of the system will see this page first. It requires users to log in as a client or staff to proceed.

3.1.1 Staff

Staff will have their login username and password created for them by the admin of the system. This can be seen in Figure 4.

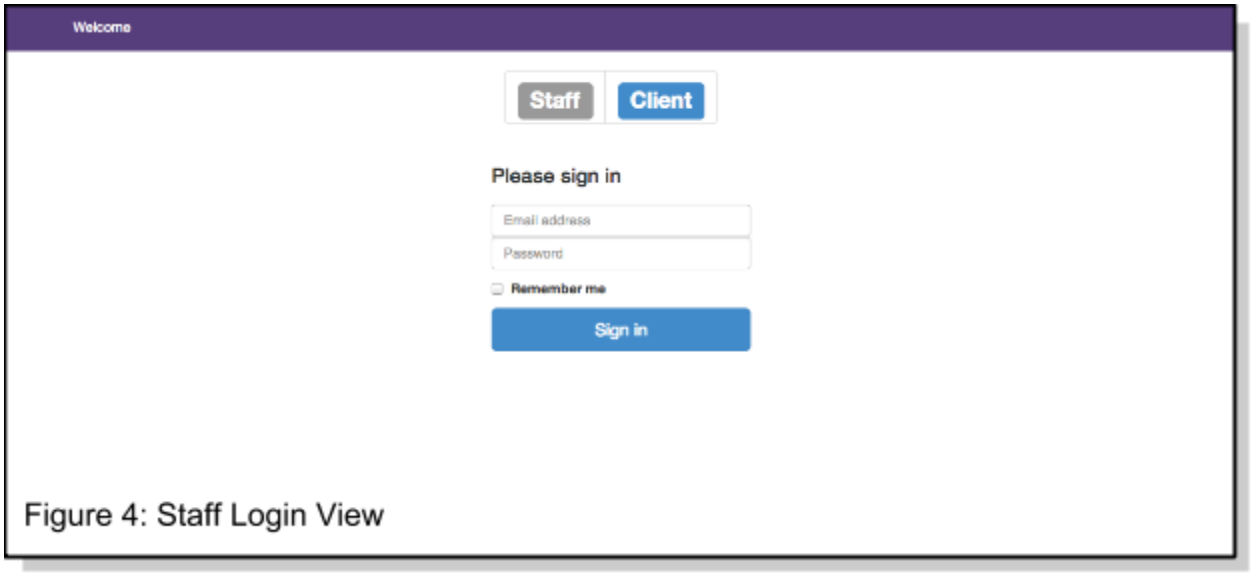


Figure 4: Staff Login View

3.1.2 Clients

Clients will use OATH to log in. OATH allows the use of a Facebook, Google or Yahoo! account to log in. This can be seen in Figure 5.

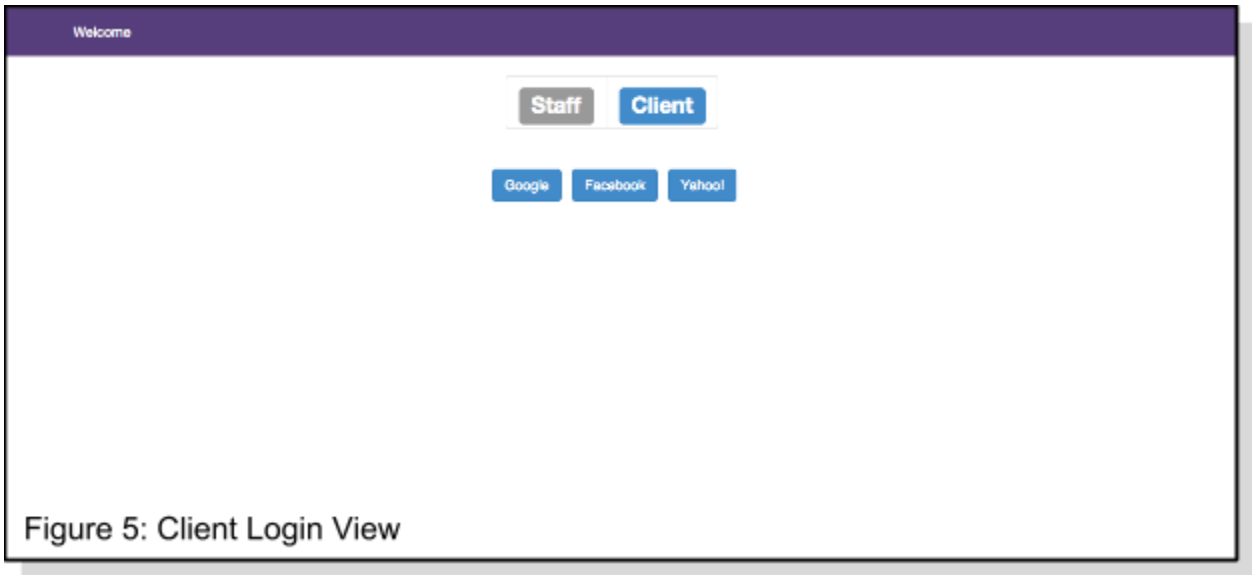


Figure 5: Client Login View

3.2 Landing Page

The landing page is where the users of the system get sent to upon logging in successfully to the system.

3.2.1 Staff Landing Page

Staff landing page displays all of the appointments booked for the given stylist. Of the left, each appointment has the attributes: Approved, Pending or Denied. The stylist can can change the status of these appointments. Clicking on an appointment will open a drawer with related details

to the appointment in question. This view can be seen in Figure 6.

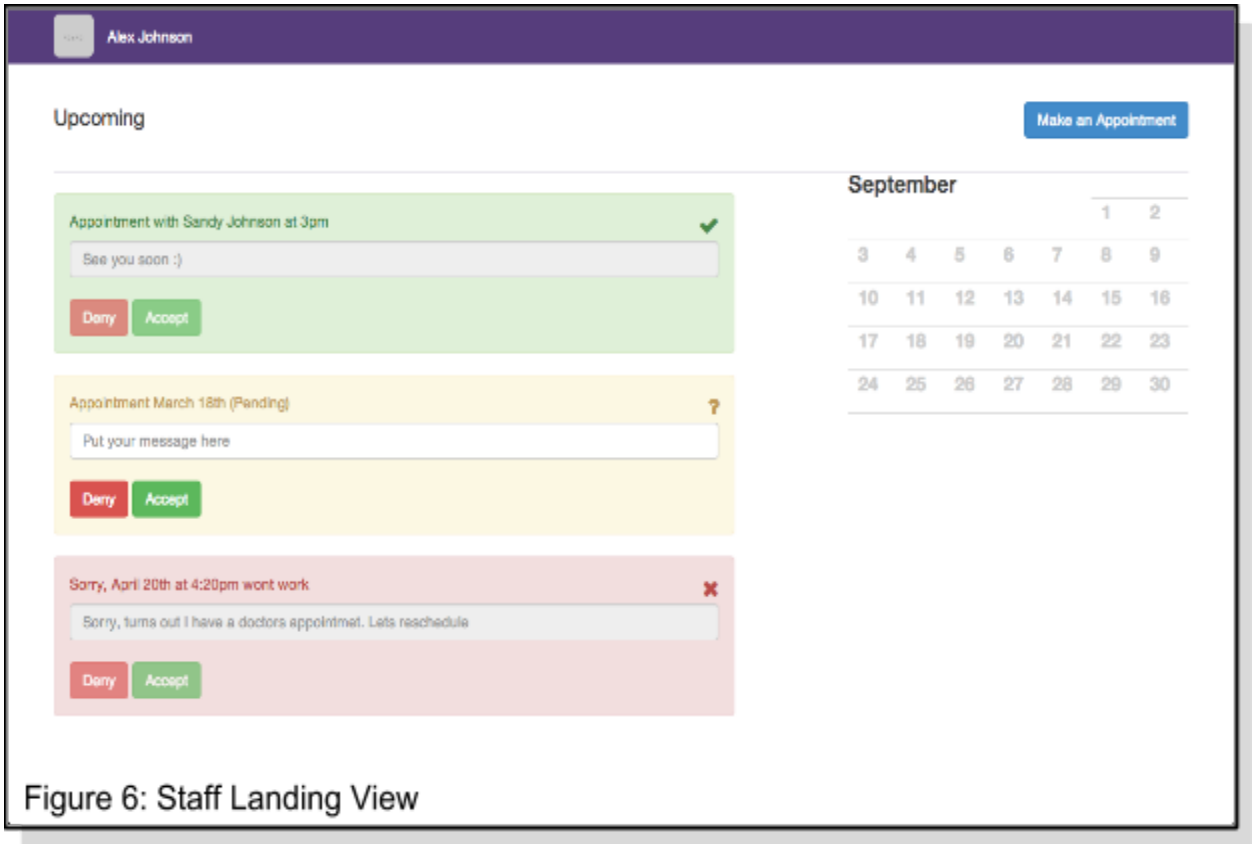


Figure 6: Staff Landing View

3.2.2 Client Landing Page

Client landing page displays the clients appointments on the left and a mini-calendar on the right to show the appointment in relationship to the rest of the month. The client's view can be seen in Figure 7.

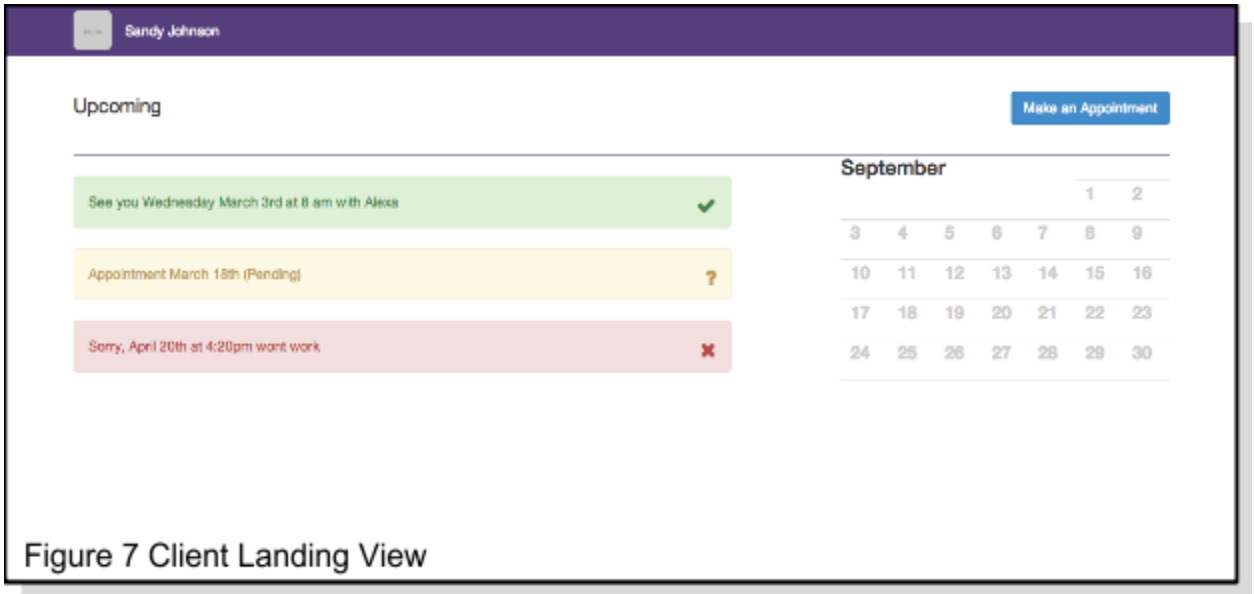


Figure 7 Client Landing View

3.3 Calendar Interface

The Calendar interface is used to make appointments by both clients for themselves as well as staff on behalf of a clients. This can be seen in Figure 8.

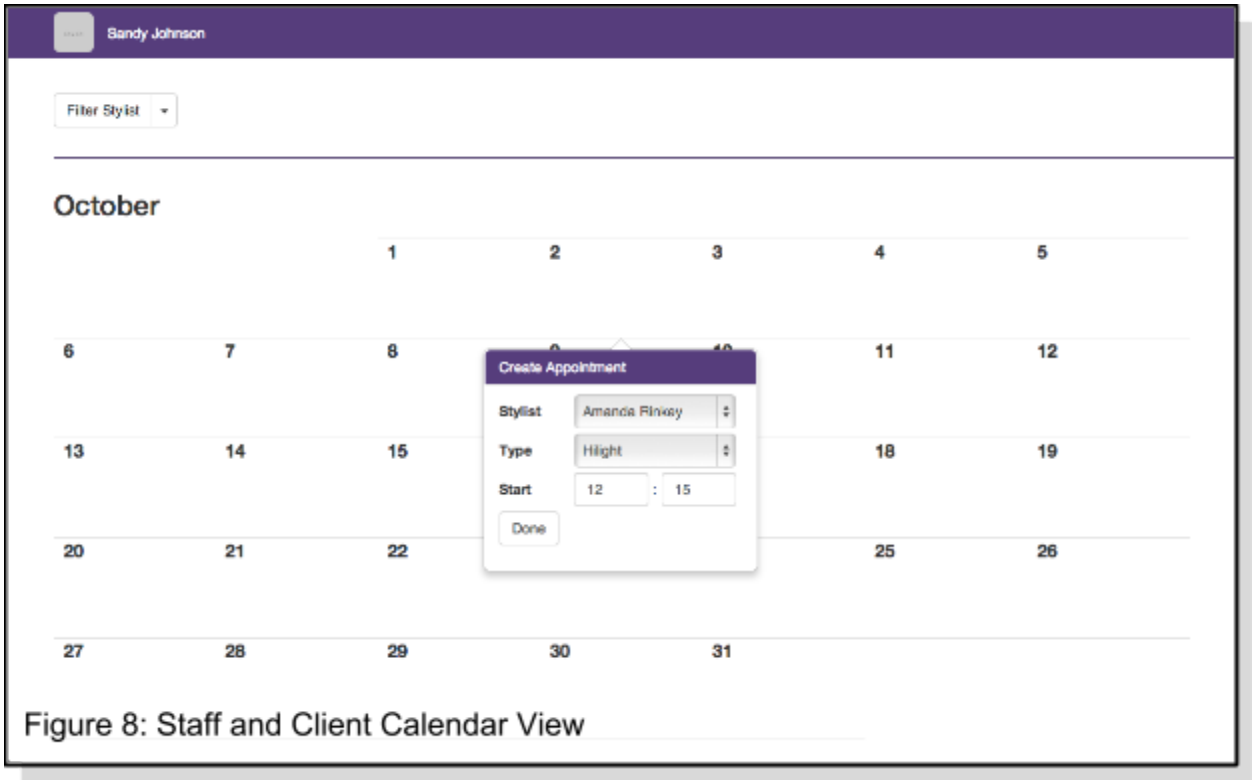


Figure 8: Staff and Client Calendar View

3.4 Edit User Profile

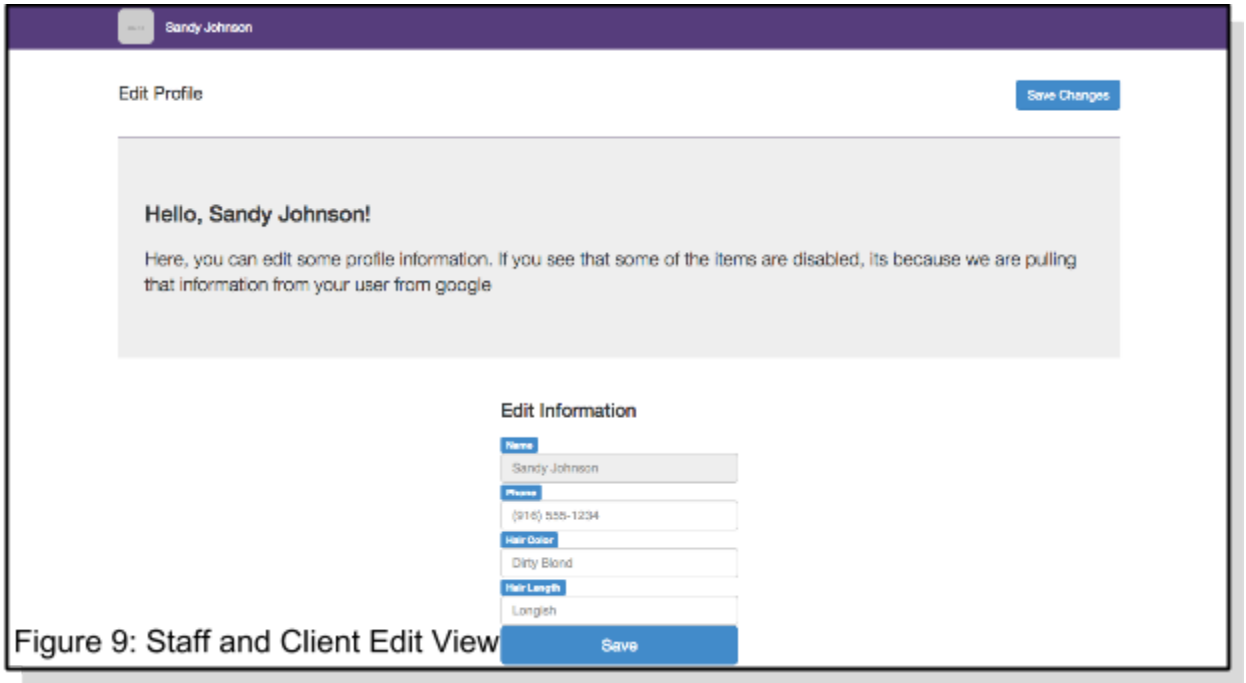


Figure 9: Staff and Client Edit View

4. DATABASE SCHEMA

This section provides the translation of the informational model contained in the Software Requirements Specification (SRS) into a relational database.

4.1 ERD Diagram

The ERD diagram below depicts the entire data used for the salon scheduling system in terms of the entities and relationships described by the data. This diagram is subject to change as the database type is still being discussed with the client.

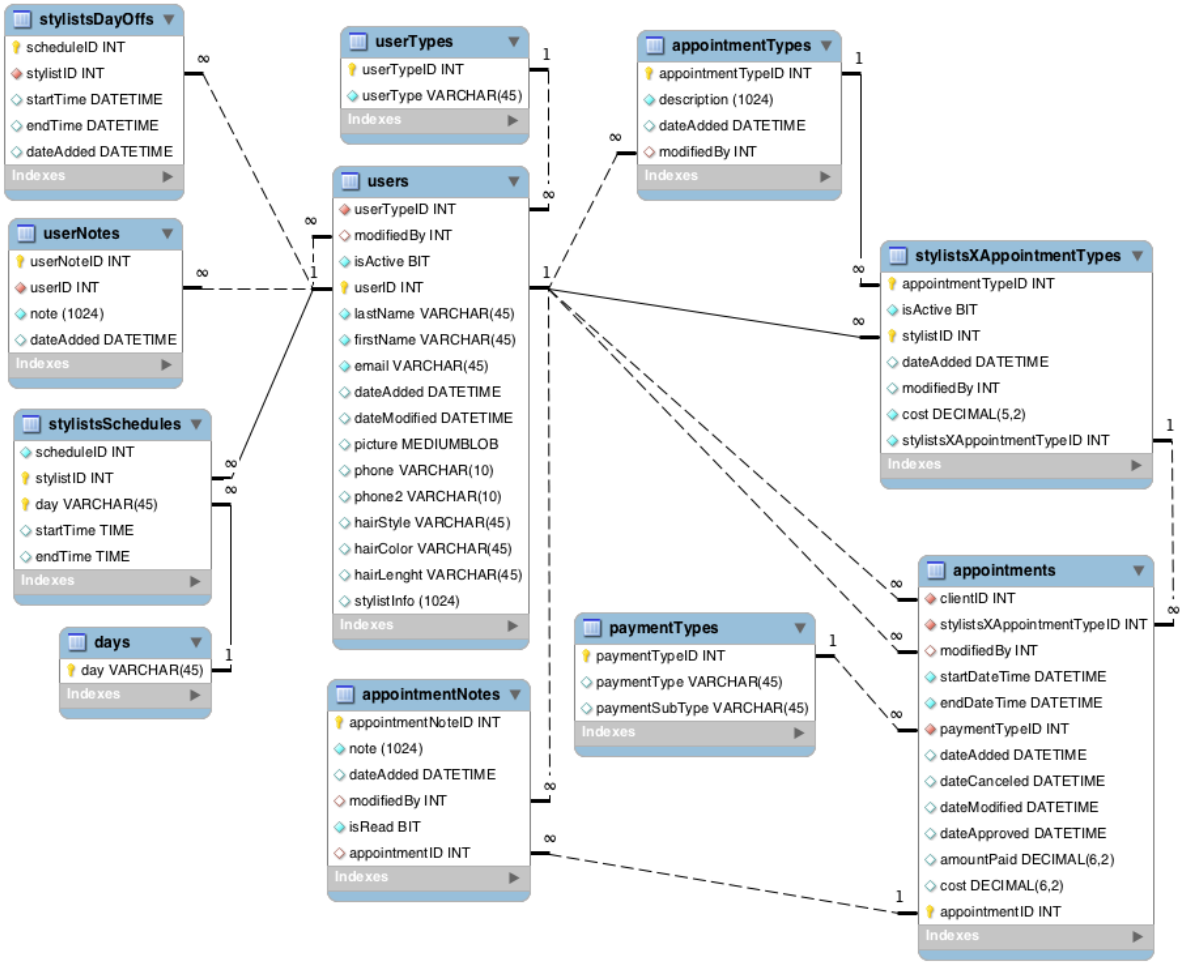


Figure 10: Database Layout

4.2 Creating the Database

Mongo provides an abstraction layer between the database architecture and the methods used to retrieve such information. The information is stored as key-value pairs and Mongo interface works with these pairs using Plain Old Java Objects (POJO) classes using its own query

language. Mongos table creation is synonymous with getter and setter methods. Per request, the library checks if the table exists; if not, the library creates the table and then provides the correct output accordingly. For example, the code below will create a table called “appointments” in the database with the schema described by an appointment model class:

```
@Repository
public class AppointmentRepository {
	@Autowired
	private MongoTemplate mongoTemplate;

	public Appointment insert(Appointment appointment) {
		mongoTemplate.insert(appointment);
		return appointment;
	}
}
```

4.3 Triggers and/or Procedures

The database schema design for this project does not use any triggers or procedures.

5. COMPONENT DESIGN SPECIFICATIONS

This section provides the detailed description for the design of the software. The section begins with a traceability matrix that map use cases to the web page and the components needed to serve the web pages. The following table shows a brief overview of how components within the system are related.

	Use Case Components		**Database**
Features	**Web Page**	**Use Case (function)**	**Table/Relations**
Clients			
Create Appointment	Login.html Index.html Calender.html	AppointmentController.java AuthenticationController.java UserController.java Appointment.java User.java AppointmentRepository.java UserRepository.java	Tables: users, appointments, stylistSchedule
Manage Appointments	Login.html Index.html	AppointmentController.java AuthenticationController.java UserController.java Appointment.java User.java AppointmentRepository.java UserRepository.java	Tables: users, appointments, stylistSchedule
Manage Profile	Login.html Index.html Edit-Profile.html	AuthentaicationController.java UserController.java User.java UserRepository.java	Tables: users
Stylists			
Manage Profile	Login.html Index.html Edit-Profile.html	AuthentaicationController.java UserController.java User.java UserRepository.java	Tables: users
Manage Appointments	Login.html Index.html	AppointmentController.java AuthenticationController.java UserController.java Appointment.java User.java AppointmentRepository.java UserRepository.java	Tables: users, appointments, stylistSchedule
Admin			
Manage Users	Login.html Index.html Manage-User.html	AuthenticationController.java UserController.java User.java UserRepository.java	Tables: users
Create Account	Login.html Index.html Manage-user.html	AuthenticationController.java UserController.java User.java UserRepository.java	Tables: users
System			
Send Email	Login.html Index.html send-email.html	AuthenticationController.java UserController.java User.java UserRepository.java	Tables: users

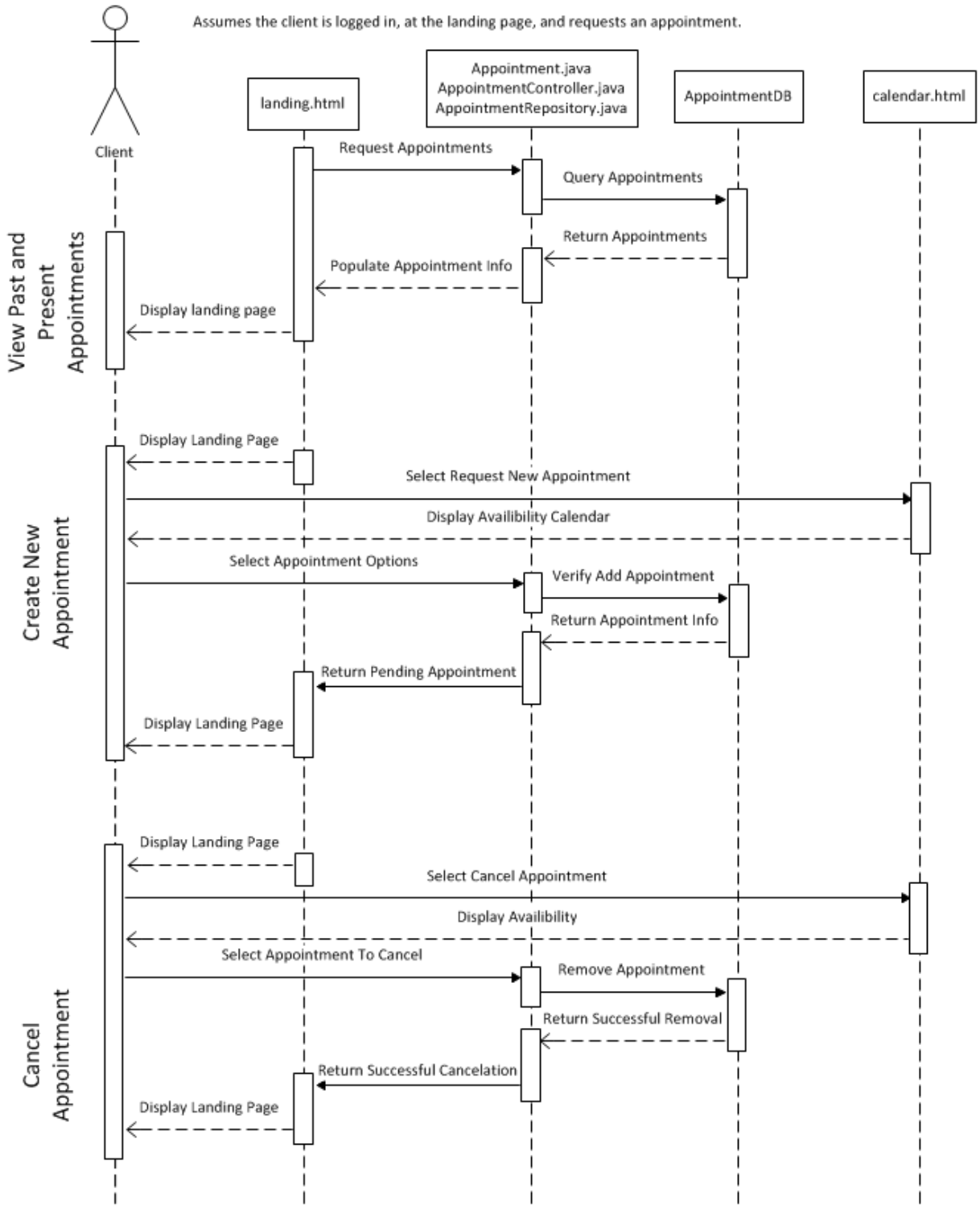
5.1 Sequence Diagrams

Diagrams for each function that illustrate the message interactions between the components that are required to deliver each specific use for that feature.

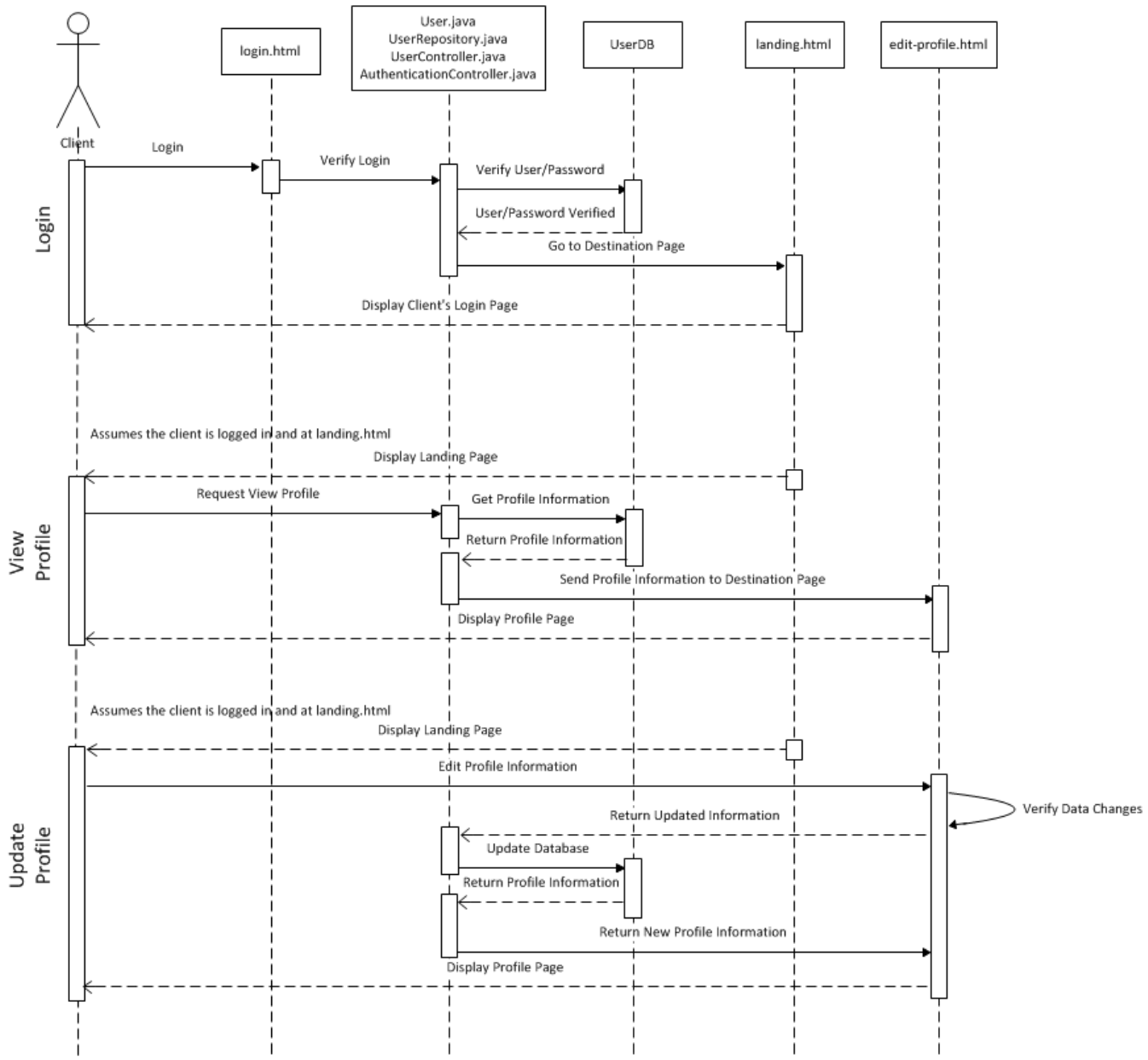
5.1.1 Client Diagrams

All of the sequence diagrams that are specific to a Client user.

5.1.1.1 Create/Manage Appointment



5.1.1.3 Manage Profile



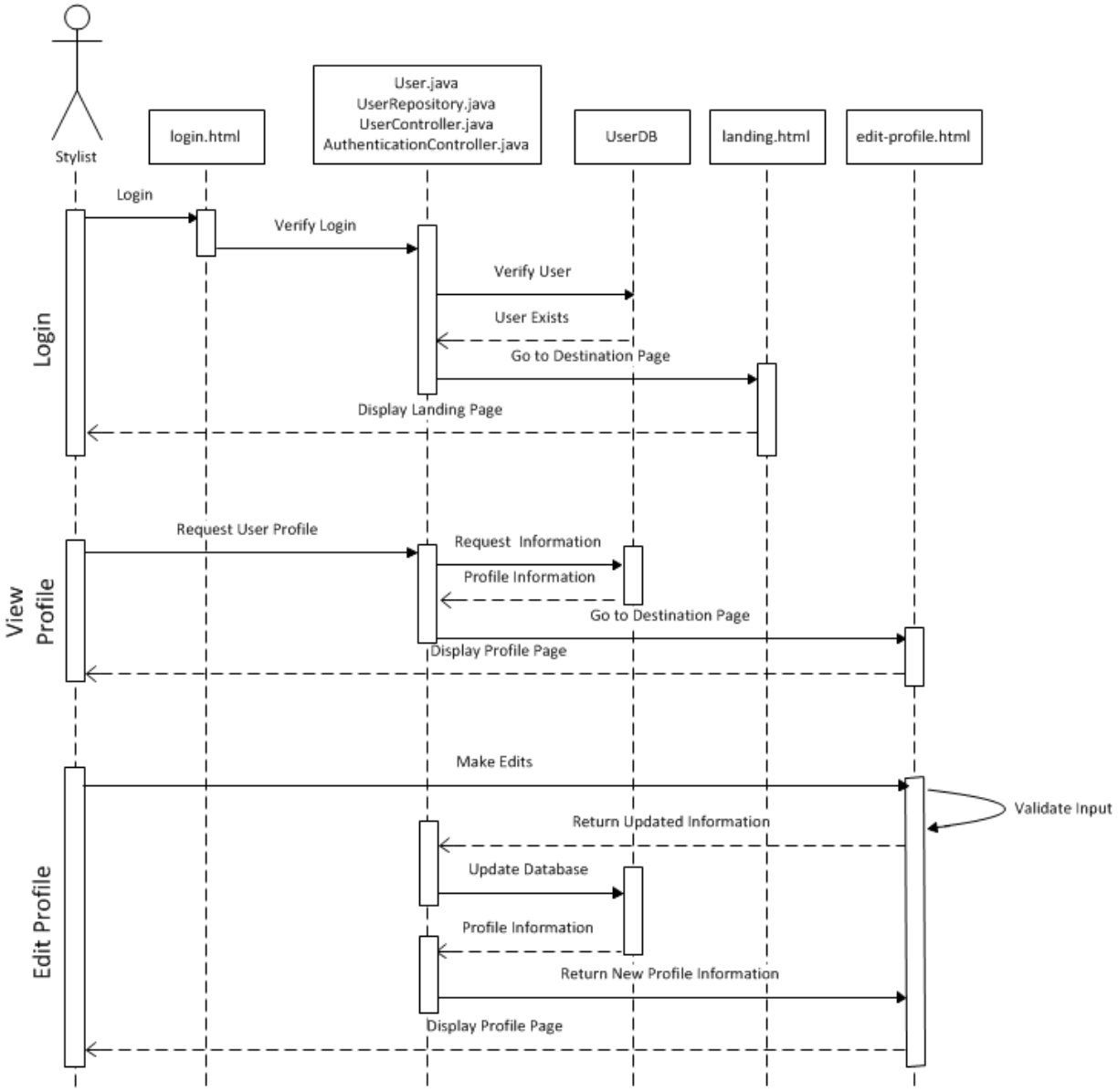
5.1.2 Stylist Diagrams

All of the sequence diagrams that are specific to a Stylist user.

5.1.2.1 Manage Profile

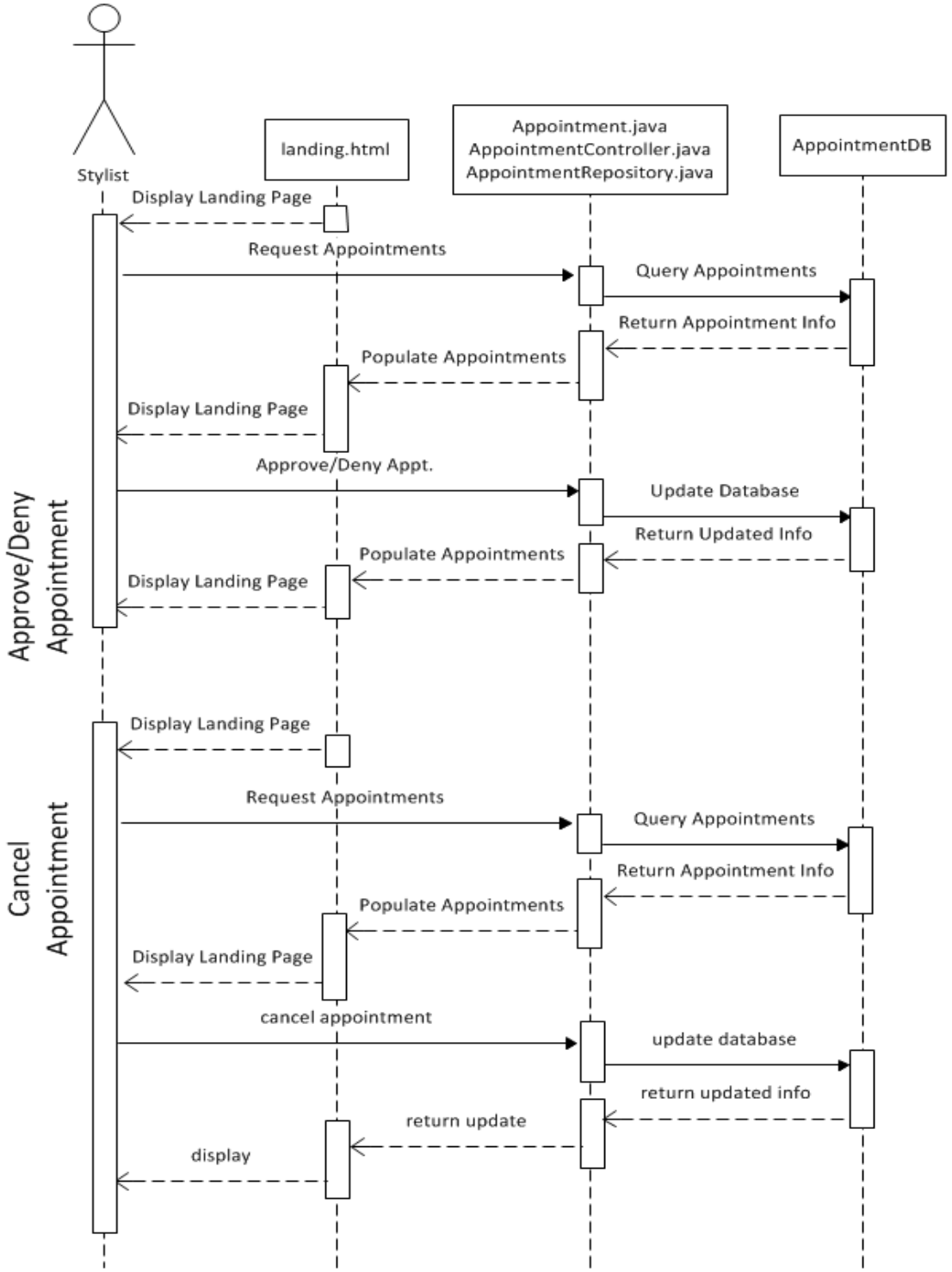
Stylists will be able to manage their profile much like a client can, however there are differences

in the types of information that the profiles contain. Due to this they are separate use cases and thus separate sequence diagrams.



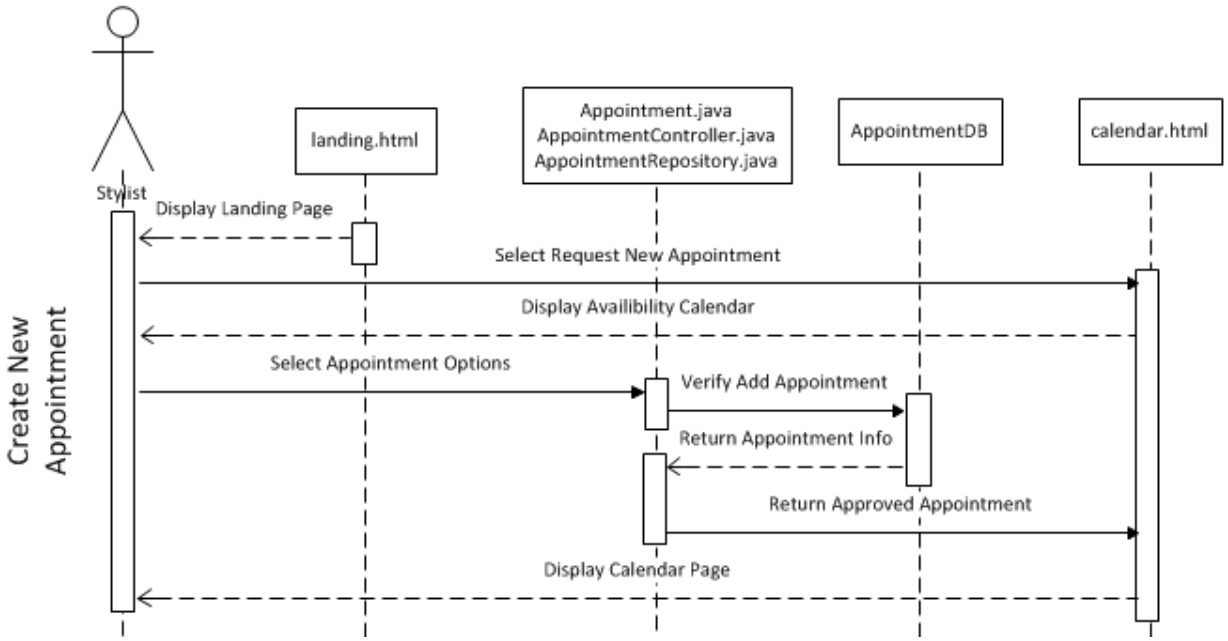
5.1.2.2 Manage Appointments

One of the main features available to stylists is being able to manage any appointments that clients have with them. In order to facilitate this stylists can approve or deny any pending appointments and cancel any already approved appointments.



5.1.2.3 Create Appointment

In addition to managing their appointments, stylists can create appointments on behalf of clients that are already preapproved when entered into the system.

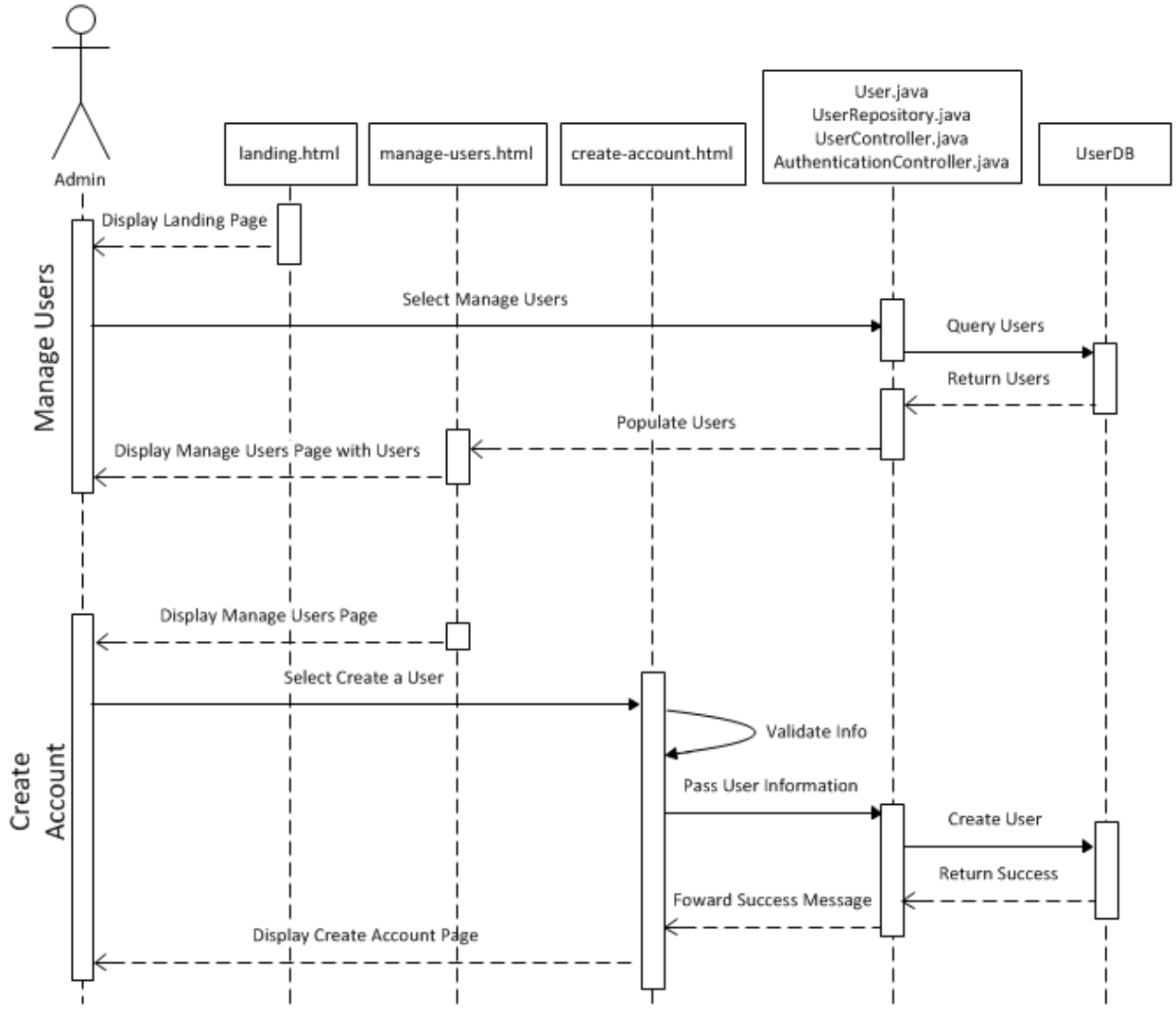


5.1.3 Admin Diagrams

All of the sequence diagrams that are specific to an Administrative user.

5.1.3.1 Manage Users and Create Account

Administrative users will be able to manage users from a manage user page, which will just consist of a list of users and an option to block them. Administrative users will also be required to create other administrative accounts and all stylist accounts.



5.2 Design Specifications by Entity

Specific details for each of the entities used in the sequence diagrams in section 5.1.

5.2.1 login.html

A login page for clients and staff to enter their credentials for our system. A tab at the top of the page will allow the user to select either client or staff.

- **Preconditions:** The user has to have an open web browser directed at our web site. The server also has to be up.
- **Interface:** HTTP GET the web page, the CSS and Javascript. The method verifyUser(String token).
- **Processing Specifications:** Login credentials will be validated to verify the user. In the case of an admin or staff member this will consist of a username and a password for our system. For normal clients this will be a token provided through the use of OAuth.
- **Screen Print:** see section 3.1.1
- **Database Requirements:** Tables - users.
- **Postconditions:** With valid login credentials the user will be logged into the system and be directed to their landing page, *landing.html*.

5.2.2 landing.html

There are several main events that can take place on or from this page. 1) The user can view and manage their appointments. 2) The user can be directed to their profile. 3) The user can select the calendar to view it and request an appointment time.

- **Preconditions:** The user is logged into the system.
- **Interface:**
    - Appointment[] findAllAppointments(User user).
    - Boolean updateAppointment(Appointment appointment).
- **Processing Specifications:** In the case of event 1 above, the user could opt to change an already existing appointment, the processing for this would require updating the appointment to reflect the changes. The other two do not require processing.
- **Screen Print:** see section 3.2.1
- **Database Requirements:** Tables - appointments and users.
- **Postconditions:** In the case of event 1, the page will simply update to reflect the new changes. In the case of event 2 the user will be redirected to *edit-profile.html*. In the case of event 3 the user will be redirected to *calendar.html*.

5.2.3 calendar.html

A page displaying a calendar with pop-up tabs that will allow users to request appointment times.

- **Preconditions:** The user must be logged into the system.
- **Interface:**
    - DateTime[] getStylistAvalibility(int stylistID, DateTime startDate, DateTime

endDate).

- **Processing Specifications:** Upon landing on the page the database will be queried to populate the calendar. If the user makes a new appointment request that information will be validated and then inserted into the database.
- **Screen Print:** see section 3.3
- **Database Requirements:** Tables - users, appointments, stylistSchedule.
- **Postconditions:** If changes are made by the user the page will update to reflect the new changes.

5.2.4 edit-profile.html

A profile page for each user that is also editable so that a user can change the information displayed on their profile.

- **Preconditions:** user is logged into the system.
- **Interface:**
  - ResponseEntity<Void> SetUser(User user)
- **Processing Specifications:** Validate the new information entered on the page.
- **Screen Print:** see section 3.4
- **Database Requirements:** Tables - users.
- **Postconditions:** Modify the content for each of the customers as requested on non-static information. Users will not be able to change their name since their names are associated with their oauth provider.

5.2.5 manage-user.html

An admin only page that will allow an admin to block and unblock users.

- **Preconditions:** user is logged into the system.
- **Interface:**
  - User[] getAllUsers();
  - void setBlockUser(blooean blocked);
- **Processing Specifications:** Queries the users of the system. uses setBlockUser() to change the flag determining if the user is blocked or not.
- **Database Requirements:** Tables - users.
- **Postconditions:** If a change to the user is made through blocking or unblocking them, the system will update the database to reflect the change.

5.2.6 create-account.html

Allows admins to create new users.

- **Preconditions:** This only applies to staff users. A Staff member must create an account for them
- **Interface:** String createUser(String Name, int level);
- **Processing Specifications:** Create a new user for a given staff. Assign each user that

is created a level of access
- **Database Requirements:** Tables - users.
- **Postconditions:** A new user is made with a given access level. Their password is randomly generated and returned to the user who created the account.

5.2.7 send-email.html

Allows stylists and admins to send an email to any user registered with our system.

- **Preconditions:** User is logged in.
- **Interface:** Boolean[] sendMail(User to, User from, String message)
- **Processing Specifications:** Send the content of the message to the recipient and mark it sent from the current user.
- **Database Requirements:** Tables - users, appointments
- **Postconditions:** Displays message has been sent once the information has been submited.

5.2.8 AppointmentController.java

- **Preconditions:** The user has logged in and been authenticated by the system. They have requested to schedule, edit, or cancel an appointment.
- **Interface:**
    - Appointment[] findAllAppointments(User user).
    - Boolean updateAppointment(Appointment appointment).
    - DateTime[] getStylistAvalibility(int stylistID, DateTime startDate, DateTime endDate).
- **Processing Specifications:** When the user requests to create, edit, or cancel an appointment the system will use the AppointmentController class to add or edit an Appointment entity. Those changes will then be added to the database using AppointmentRepository.
- **Screen Print:** N/A
- **Database Requirements:** Tables - users, appointments, stylistSchedule.
- **Postconditions:** A new appointment or existing appointment will then be added or updated in the database and the appropriate users and stylists will be notified.

5.2.9 AuthenticationController.java

- **Preconditions:** The user has opened a web browser and navigated to the login page and has initiated a request to log into the system.
- **Interface:**
    - authenticate(LoginForm login)
- **Processing Specifications:** Upon requesting a login the system will use the AuthenticationController in order to ensure that an authorized user is accessing the system. Once authorized the user will be redirected to the page they have requested
- **Screen Print:** N/A
- **Database Requirements:** Tables - users.

- **Postconditions:** The user will be authenticated and allowed to access the pages for users, stylists, or admins.

5.2.10 UserController.java

- **Preconditions:** The user has logged in and been authenticated by the system. They have requested to update user profiles.
- **Interface:**
    - int AddUser(User user)
    - Boolean UpdateUser(User user)
- **Processing Specifications:** After the user has been authenticated by the system they will be able to add and update their profiles by calling UserController methods.
- **Screen Print:** N/A
- **Database Requirements:** Tables - users.
- **Postconditions:** User profile will be created or updated.

5.2.11 Appointment.java

- **Preconditions:** A user, whether client or stylists, has requested a new appointment and the AppointmentController class has already validated the requested appointment.
- **Interface:**
    - String getId()
    - void setId(String id)
    - String getClientID()
    - void setClientID(String clientID)
    - String getStylistID()
    - void setStylistID(String stylistID)
    - Integer getStartTime()
    - void setStartTime(Integer startTime)
    - Integer getEndTime()
    - void setEndTime(Integer endTime)
    - String getStatus()
    - void setStatus(String status)
- **Processing Specifications:** AppointmentController will create the Appointment entity based on the data entered by the user. That Appointment entity will then be added to the database.
- **Screen Print:** N/A
- **Database Requirements:** Tables - appointments, stylistSchedule.
- **Postconditions:** A new Appointment entity will be added into the database that represents a particular Appointment for a given client and stylist.

5.2.12 User.java

- **Preconditions:** An Admin that is logged in has requested to create a new account. The UserController will then get the the request and process it.
- **Interface:**

    - String getId()
    - void setId(String id)
    - String getGroup()
    - void setGroup(String group)
    - String getFirstName()
    - void setFirstName(String firstName)
    - String getLastName()
    - void setLastName(String lastName)
    - String getEmail()
    - void setEmail(String email)
    - String getPassword()
    - void setPassword(String password)
    - String getToken()
    - void setToken(String token)
    - String getAvatarURL()
    - void setAvatarURL(String avatarURL)
    - getPhone()
    - void setPhone(String phone)
    - String toString()
- **Processing Specifications:** The UserController will process the request send the information to the UserRepository which will create a new User entity with the data provided by the Admin. The entity will then be added to the database.
- **Screen Print:** N/A
- **Database Requirements:** Tables - users.
- **Postconditions:** A new user account will be added to the users database. A user can now use that new account to login and access the site.

5.2.13 AppointmentRepository.java

- **Preconditions:** A user is logged in has requested to create or update an appointment with the specified information.
- **Interface:**
    - public AppointmentRepository(MongoTemplate mongoTemplate)
    - public Appointment insert(Appointment appointment)
    - public Appointment findByID(String id)
    - public List<Appointment> findAll(User user)
    - public void save(Appointment appointment)
- **Processing Specifications:** The AppointmentCollection will get the users input for a new appointment request. That request will then be processed and the AppointmentRepository will create a new Appointment entity to be added to the database with the data the user has entered.
- **Screen Print:** N/A
- **Database Requirements:** Tables - appointments.
- **Postconditions:** A new Appointment entity will be added to the collection and will be

able to be accessed by users on the website.

5.2.14 UserRepository.java

- **Preconditions:** An admin that is logged in has requested to create or update user information. They have entered the new user information or updated an existing user's information. The information has already been verified and validated by the UserController.
- **Interface:**
    - void insert(User user)
    - User findByEmail(String email)
    - User findById(String id)
    - User findByToken(String token)
    - void save(User user)
- **Processing Specifications:** The UserRepository will take the given information and create a new User entity or find and update an existing User entity. It will then save the new or updated User entity into the users tables in the database.
- **Screen Print:** N/A
- **Database Requirements:** Tables - users.
- **Postconditions:** A new or updated User entity will be added to the database that can be used to log in to the website with the given credentials.

6. PERFORMANCE ANALYSIS

This section provides details on any performance issues or constraints that may have arisen during the design and implementation phase.

6.1 System Performance Requirements

Based on our sponsors needs, we have determined that only one server is needed that will support a virtually unlimited number of connections for the users to use. Our software does not limit the number of active users, however, based on our sponsor's needs we have limited the hardware that will be supporting our software. Based on that, our application will support up to 100 active connections.

6.2 System Implementation Constraints

The only constraint the sponsor has for our system is its hosting costs. They have stated they do not wish to spend a lot of money on hosting. Due to this we know that the system running our application will not be extremely powerful. All of the options they can choose from support MySQL for a database and have at least 20GB of hard disk space.

6.3 System Implementation Details

The application is designed to separate presentation layer (frontend) from business logic and data management layers (backend). Both frontend and backend are platform agnostic. Frontend can be managed by any Web Server such as Apache or IIS. Backend can be managed by any Application Server that supports J2EE such as Tomcat. Backend data layer will be managed by

MySQL and MongoDB database servers. Backend business logic will be implemented in Spring MVC framework using Java. Our architecture allows for low-cost, performance efficient, single server deployment.

7. RESOURCE ESTIMATES

This software requires a dedicated server intended for web hosting. However, it is required for it to be elastic, able to adapt to the traffic of the application.
Per our research of available services in the industry, we estimate that a server that could serve such needs will cost roughly $20 per month.
Once the software is finished, we do not foresee a need for upkeep as the application is self regulating.

8. SOFTWARE REQUIREMENTS TRACEABILITY MATRIX

Some of the requirements have changed since the SRS. The requirements that are no longer required are marked as Not Applicable in the table below.

Use Case/Section Number	Use Case Name	SDS Section
UC1/3.1.1	Send Email	Not Applicable
UC2/3.1.2	View Profile	5.1.1.2
UC3/3.1.3	Create Account	Not Applicable
UC4/3.1.4	Request Password Recover	Not Applicable
UC5/3.1.5	Make Appointment	5.1.1.1
UC6/3.1.6	View Stylist	5.1.1.1
UC7/3.1.7	Change Password	Not Applicable
UC8/3.1.8	Update Profile	5.1.1.2
UC9/3.1.9	Cancel Appointment	5.1.1.1
UC10/3.1.10	View Past/Present Appointment	5.1.1.1
UC11/3.1.11	Create Profile	5.1.2.1
UC12/3.1.12	View Profile	5.1.2.1
UC13/3.1.13	View Client List	5.1.2.1
UC14/3.1.14	View Client Profile	5.1.2.1
UC15/3.1.15	Manage Appointment	5.1.2.2
UC16/3.1.16	Set Availability	Not Applicable
UC17/3.1.17	Manage Services	Not Applicable
UC18/3.1.18	Password Change	Not Applicable
UC19/3.1.19	Update Profile	5.1.2.1
UC20/3.1.20	Create Appointment	5.1.2.3
UC21/3.1.21	Manage Pending Appointment	5.1.2.2
UC22/3.1.22	Cancel Appointment	5.1.2.2
UC23/3.1.23	Create Stylist/Admin Account	5.1.3.1
UC24/3.1.24	Reactivate/Deactivate User	5.1.3.1

9. APPROVALS

Project Sponsors

By signing this document you agree that this document represents a shared understanding of the design requirements for the project and the system described will satisfy your needs. Any future changes in this baseline specification will be made through the project's defined change process. You understand that approved changes might require the renegotiation of the scope of the work that can be completed within the allotted time. Approval also constitutes an agreement between you and the development team that the requirements, as stated, provide all that is necessary for the development team to proceed with the design and implementation of the software system.

Lisa Sigurdson _____

Advisor

By signing this document you are approving the agreements and design specifications defined within this document.

Ahmed M Salem _____

Team Sierra

By signing this document you are agreeing to the design specifications defined within this document and complete the project as it is has been specified.

Alex Chernyak _____

Joubin Jabbari _____

Kyle Matz _____

Mike McParland _____

Scott Livingston _____

Serge Lysak _____

APPENDICES

APPENDIX A. Database Tables and Attributes

Please reference Section 4.1 Database Schema ERD Diagram for a complete listing of database tables and its attributes details.

APPENDIX B. Alphabetic Listing of Each Attribute and its Characteristics

Please reference Section 4.2 Database Schema Creating the Database for a complete listing of each attribute and its characteristics.