

Online System Design pack for Rolsa Technologies



Contents

Design.....	5
Entity Relation (database) Diagram	5
Figma / Interface Design.....	11
Splash Page	11
User pages	12
Admin Pages	17
Colour Scheme, Font Pack & Justification.....	20
Pseudocode.....	21
Flow Charts	Error! Bookmark not defined.
Decomposition Flow Chart.....	30
Data Flow Diagram	30
Tables	Error! Bookmark not defined.
Data Dictionary	Error! Bookmark not defined.
Data Structures / Variable List	Error! Bookmark not defined.
Test Strategy	33
Test Types Being Implemented.....	Error! Bookmark not defined.
Logic Tests.....	Error! Bookmark not defined.
Erroneous, boundary and correct data tests.....	Error! Bookmark not defined.
Testing Table Design.....	Error! Bookmark not defined.
Examples of tests that could be performed and written up	Error! Bookmark not defined.

Functional and Non-Functional Requirements

Functional Requirements

1. Account Systems

- a. Account Registration to allow the end user to receive a personalised dashboard containing appointment data and usage metrics.
- b. Account Login System to allow the user to access the account they have just registered
- c. Password updating system so that the end user can always change the password if in fear of compromise or has forgotten it.
- d. Account Deletion/Termination allowing the user to delete their account and the associated metrics and appointments should they wish to exit the Rosla Technologies market.

2. Metric Gathering

- a. Have the Metrics of a household / specific item be available to the user at any time over any period not exceeding the period the user has been with the company to avoid misconception of “blank” data.
- b. Be able to set usage limits on certain items of tech such as an EV charger or smart thermostat / heating solution to limit cost and consumption
- c. Email / smart contacting solution to the user if the item with a limit on it is close to the limit and provide them with options to increase OR remove the limit they have set.

3. Ability to book Appointments

- a. To be able to book appointments directly through the app for a variety of tasks
- b. To be able to set notes at point of booking detailing your issue from the end user Point of View (POV)
- c. Cancellation and rescheduling of appointments up to the date of the current appointment.

4. Admin Panel

- a. Be able to see overall usage metrics in a geographical area
- b. Be able to see usage metrics for one household or business facility
- c. Be able to procure contact details for a client to contact them if necessary

Non-Functional Requirements

1. Performance and Speed

- a. What are the loading times of the solution, are they within expected values defined as between 2 and 5 seconds in line with the average for the software space this will operate in
- b. Efficiency of pulling through user data when they arrive at the dashboard. Again, asking are they waiting just those 2-5 seconds for the Database query or is the data being tied up for far too long.

2. Security and Protection of Sensitive Data

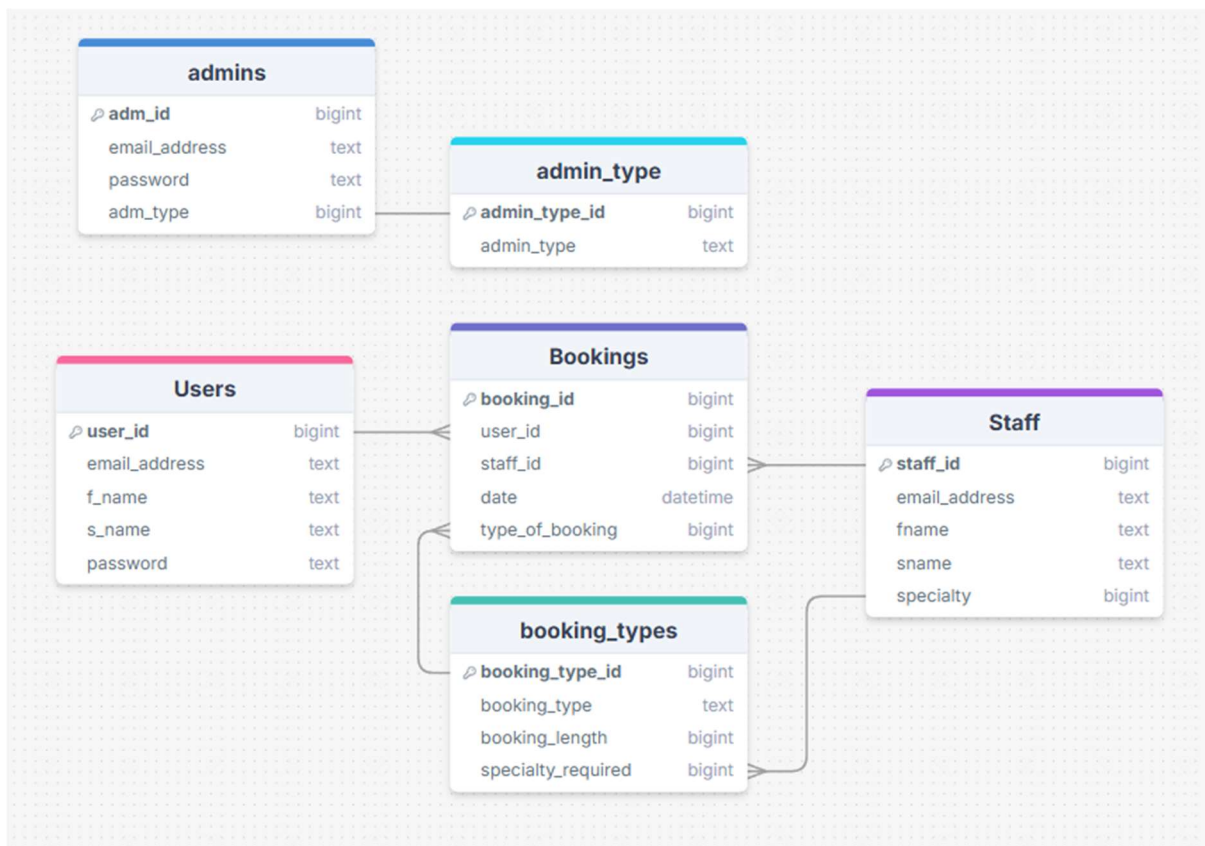
- a. Usage of SHA-3 encryption system defined by NIST as: the most up to date and secure hashing algorithm working alongside and in place of the SHA-2 based as and when systems are upgraded or replaced. This SHA-3 system will be used for all password and sensitive data storage.
- b. Storing only data that is needed in the database and only the necessary data as to avoid potential data breaches being worse than they need to be should they occur.

3. Usability

- a. Is the system comprehensive and easy to understand. i.e. can any user pick up the website and understand its overall flow
- b. Is the website compliant to WCAG and W3C in terms of being accessible to all user groups regardless of impairments to a certain point

Design

Entity Relation (database) Diagram and Data Dictionary / variable list



The Entity Relationship Diagram (ERD) I've designed strikes a balance between keeping things simple and making sure it works well for all required functions. It may look straightforward, but it's built with smart relationships that handle multiple needs at the same time. The diagram includes user details (FR1.a), staff details, booking information (FR3.a), different booking types, and a separate admin system (FR4) for better security.

The goal of this database design was to keep it streamlined while focusing on the most important features. This makes it easier to develop and allows quick implementation of essential functions while keeping things flexible for future improvements. One example of this is the decision to leave out an audit logging system in the first version. This wasn't an oversight—it was a conscious choice to avoid extra complexity that wouldn't provide enough value at this stage.

The relationships between tables are carefully designed. For example, the one-to-many link between user_id in the users table and user_id in the bookings table ensures that each user can have multiple bookings. This setup not only meets booking requirements (FR3.a) but also keeps data consistent by using foreign key constraints. This prevents

issues like lost or orphaned records. A similar structure is used for staff members, linking `staff_id` in the `staff` table to `staff_id` in `bookings`—this makes development and maintenance more intuitive and consistent.

The `booking_types` table follows a key database design principle called normalization. Instead of repeating booking type details across multiple records, this table keeps them separate, linking each `booking_type_id` to specific appointment durations. This prevents data duplication and makes it easier to add new appointment types in the future without changing the database structure. It also improves performance when analysing booking trends.

The admin system is intentionally separated from the main data to improve security. Instead of mixing admin credentials (such as `adm_id`, email, and password) with regular user data, I've placed them in a separate section. This helps prevent security risks like unauthorized access through SQL injection attacks. The admin system also includes role-based access control by linking `admin_type_id` to `adm_type`, which ensures that each admin only has the level of access they need (FR4).

Overall, this design keeps the database efficient, secure, and flexible. It avoids unnecessary complexity while making sure future features—such as audit logging, advanced reports, or new services—can be added without major changes to the structure.

Admin Table Dictionary

Database Field	Field Type	Constraints / Dependencies	Justification	Example Data
<code>Adm_id</code>	INT	Primary Key with auto increment enabled	Needs to always be unique, auto incrementation (AI) makes this an easy process	1, 21, 300, 8008
<code>Email_address</code>	Varchar	Not Null and must be unique	Will contain more than just text, MUST include “@” etc.	example@example.co.uk
<code>Password</code>	varchar	Not Null	Will contain more than just text, MUST include special characters etc.	Unhashed : Steve123!! Hashed : 9b1663bb79c9d08f8d5ac5
<code>Adm_type</code>	INT	FOREIGN KEY (references <code>admin_type.admin_type_id</code>), NOT NULL	This is a value drawn from <code>admin_type</code> . It will always be a number	1, 2, 3

Admin_type Dictionary

Database Field	Field Type	Constraints / Dependencies	Justification	Example Data
Admin_type_id	INT	Primary key , auto incrementing	Will always be INT as it is auto incrementing and is the quick reference for the admin table to check, actual role info not included in this field	1, 2,3
Admin_type	Varchar	Not Null	Will sometimes need special chars. But is mainly the varchar type for redundancy and insurance	Super(SUDO) Creator Editor

Users Dictionary

Database Field	Field Type	Constraints / Dependencies	Justification	Example Data
User_id	INT	Primary Key with auto increment enabled	Needs to always be unique, auto incrementation (AI) makes this an easy process	1, 21, 300, 8008
Email_address	Varchar	Not Null and must be unique	Will contain more than just text, MUST include “@” etc	example@example.co.uk
Password	varchar	Not Null	Will contain more than just text, MUST include special characters etc	Unhashed : Steve123!! Hashed : 9b1663bb79c9d08f8d5ac5
f_name	Varchar	Not null	May contain non ASCII characters with accents etc	Paul
s_name	Varchar	Not null	May contain non ASCII characters with accents etc	Blart

Staff Dictionary

Database Field	Field Type	Constraints / Dependencies	Justification	Example Data
Staff_id	INT	Primary Key with auto increment enabled	Needs to always be unique, auto incrementation (AI) makes this an easy process	1, 21, 300, 8008
Email_address	Varchar	Not Null and must be unique. Will need to contain “@RTECH.co.uk”	Will contain more than just text, MUST include “@” etc	example@RTECH.co.uk

Password	varchar	Not Null	Will contain more than just text, MUST include special characters etc	Unhashed : Steve123!! Hashed : 9b1663bb79c9d08f8d5ac5
f_name	Varchar	Not null	May contain non ASCII characters with accents etc	Adam
s_name	Varchar	Not null	May contain non ASCII characters with accents etc	Sandler
Specialty	INT	Foreign key, not null	A staff member will always have a specialty so field will never be null	1, 2, 3

Booking Dictionary

Database Field	Field Type	Constraints / Dependencies	Justification	Example Data
Booking_id	INT	Primary Key with auto increment enabled	Needs to always be unique, auto incrementation (AI) makes this an easy process	1, 21, 300, 8008
User_id	Int	FOREIGN KEY (references Users.user_id), NOT NULL	Needs to always be unique, auto incrementation (AI) makes this an easy process.	1, 21, 300, 8008
Staff_id	Int	FOREIGN KEY (references Staff.staff_id), NOT NULL	Needs to always be unique, auto incrementation (AI) makes this an easy process	1, 21, 300, 8008
Date	Datetime	EPOCH / POSIX / UNIX TIME NOT NULL	Easiest way to gather a date and time in one variable using POSIX / EPOCH / UNIX time	1742293690 = Tuesday, March 18, 2025, 10:28:10 AM 2147483647 = Tuesday, January 19, 2038, 3:14:07 AM
Type of Booking	INT	FOREIGN KEY (references booking_types.booking_type_id), NOT NULL	Links to type of booking table	1, 2, 3

Booking Type Dictionary

Database Field	Field Type	Constraints / Dependencies	Justification	Example Data
Booking_type_id	INT	PRIMARY KEY, AUTO_INCREMENT	Needs to always be unique, auto incrementation (AI) makes this an easy process.	1, 21, 300, 8008
Booking_type	VARCHAR	NOT NULL	VARCHAR is easiest way to gather the info for the booking type	"Installation", "Upgrade", "maintenance"
Booking_length	INT	NOT NULL	Will always be a number	20, 60, 120
Specialty_required	INT	FOREIGN KEY , NULL	Link to specialty required in staff table	1, 2, 3

PHP Session Variables

Database Field	Field Type	Justification	Example Data
\$_SESSION['user_id']	INT	Stores a logged in user Identifier	1, 21, 300, 8008
\$_SESSION['admin_ssnlogin']	BOOLEAN	If an admin ssn login is not detected on an admin only page then a redirect is needed	true, false
\$_SESSION['ERROR']	STRING	To store error messages	"ADM EXISTS"
\$_SESSION['SUCCESS']	STRING	To store a success message	"ADM REGISTERED"

PHP FORM VARIABLES

Database Field	Field Type	Justification	Example Data
\$_POST['username']	STRING	Username from login form	"jsmith"
\$_POST['password']	STRING	Password from login form	"SecurePW123!"
\$_POST['confirmPassword']	STRING	Password confirmation	"SecurePW123!"

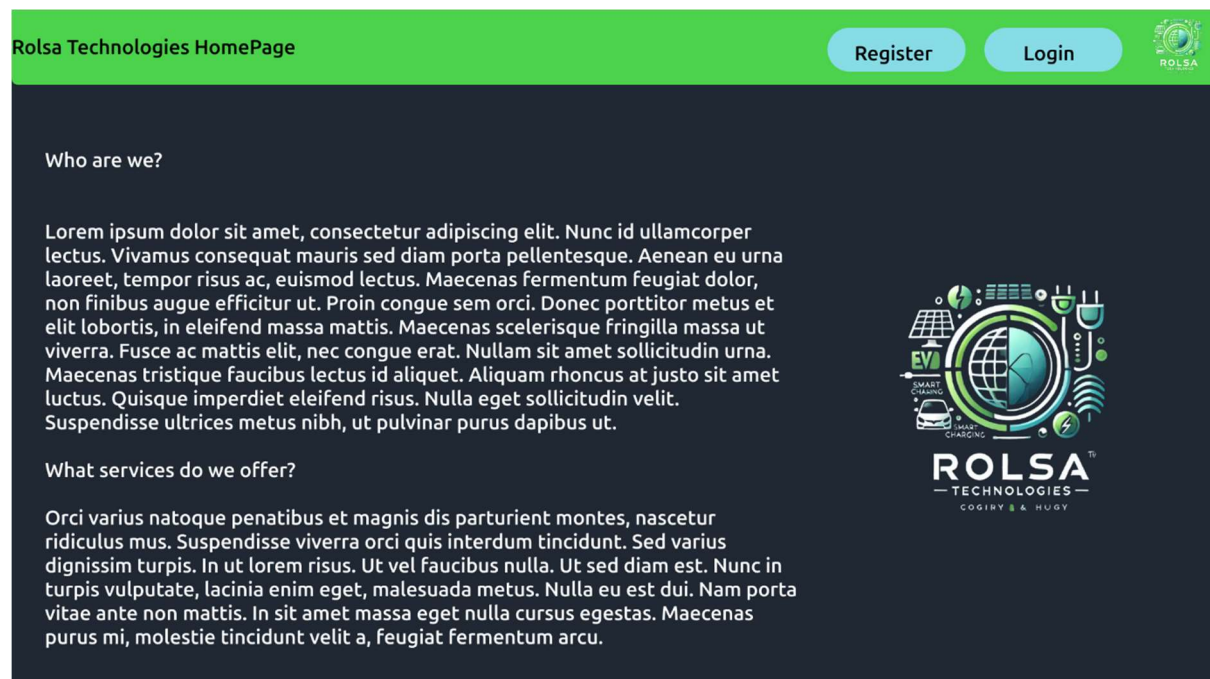
<code>\$_POST['firstName']</code>	STRING	First name from registration	"John"
<code>\$_POST['surname']</code>	STRING	Surname from registration	"Smith"
<code>\$_POST['email']</code>	STRING	Email from registration	" <u>user@example.com</u> "
<code>\$_POST['booking_type']</code>	INT	Selected booking type	3
<code>\$_POST['appointment_date']</code>	STRING	Selected appointment date	"2025-04-15"
<code>\$_POST['appointment_time']</code>	STRING	Selected appointment time	"14:30"

PHP Function Return Variables

Database Field	Field Type	Justification	Example Data
<code>\$connection</code>	OBJECT	PDO database connection	PDO Object
<code>\$result</code>	ARRAY	Database query results	<code>[["user_id" => 42, "email" => "user@example.com"]]</code>
<code>\$hashedPassword</code>	STRING	SHA-3 hashed password	"9b1663bb79c9d08f8d5ac5..."
<code>\$sanitizedInput</code>	STRING	User input after sanitization	" <u>john.smith@example.com</u> "
<code>\$signupDate</code>	STRING	Current date in YYYY-MM-DD	"2025-03-24"
<code>\$destination</code>	STRING	Redirect URL	"/admin/admin_login.php"

Figma / Interface Design

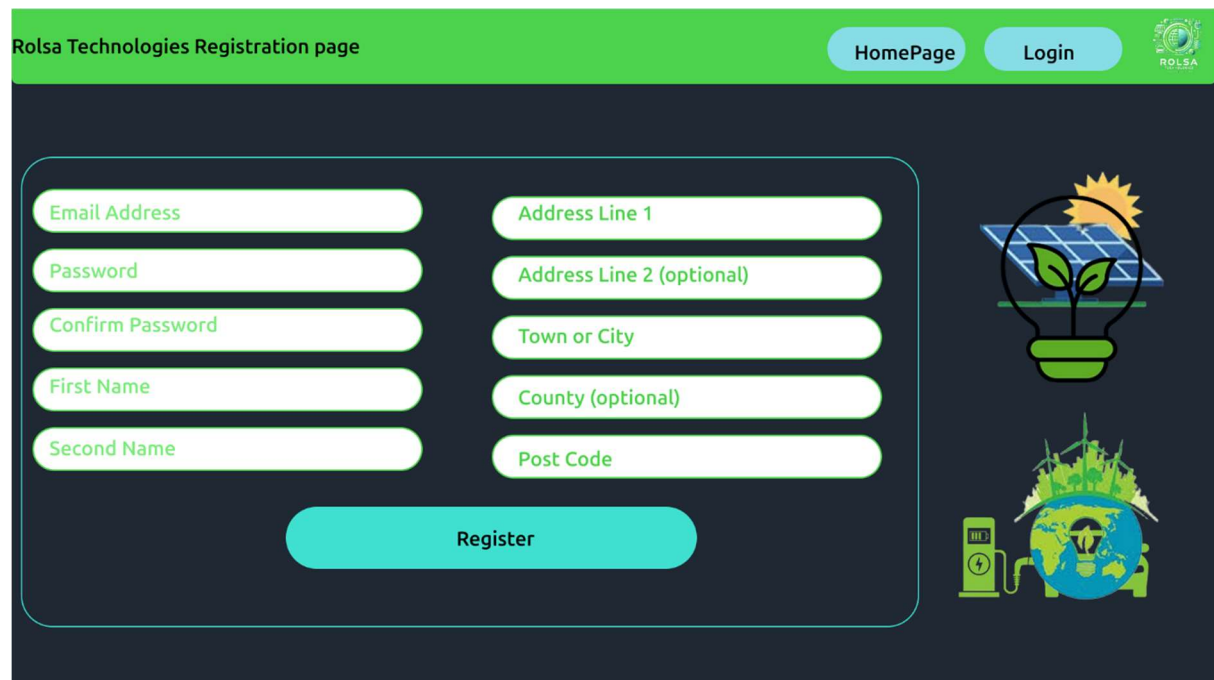
Splash Page



The splash page for Rolsa Technologies software solution that I have designed is simple and static. This is a deliberate decision both based on my own user experience and that of others in cases where a complex, overloaded website often makes people concerned that they may be missing out on information. Therefore a simple, minimalist design maintains that the information that is needed is presented and not 6 different side scrolling pop ups and over the top graphics. Another factor for the simple design is that I know it is an easier development than that of an overcomplex system and given the development time constraints I am held to, I believe this is the better solution as it CAN be delivered on effectively.

User pages

User Registration



Rolsa Technologies Registration page

HomePage Login

Email Address

Password

Confirm Password

First Name

Second Name

Address Line 1

Address Line 2 (optional)

Town or City

County (optional)

Post Code

Register

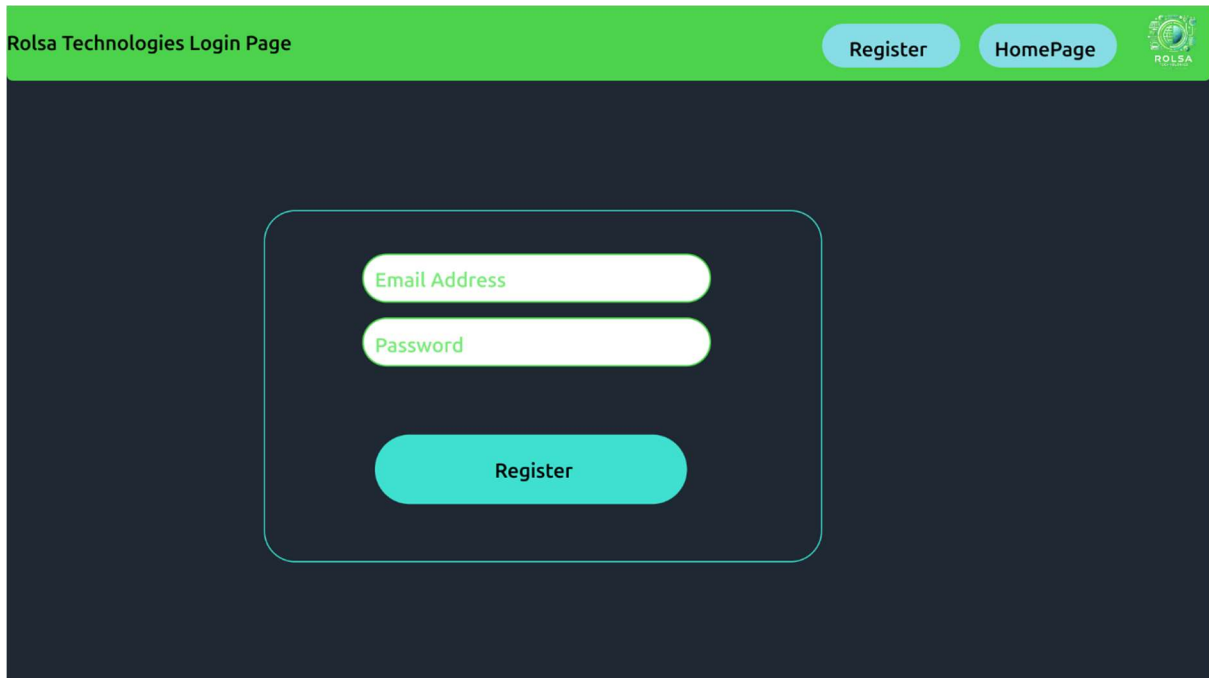
User Registration page includes the same simple and minimalist design as my splash page and indeed every other page of the system. It also contains some clipart photos of solar panels, EV charging, a green energy lightbulb and another associated tree hugger emblems.

The User Registration asks for:

- Email Address (FR1.A)
 - For contact information
- Password (FR1.A & FR1.C)
- Confirm Password (FR1.A & FR1.C)
 - Password MFA
- First Name(FR1.A)
- Second Name(FR1.A)
- Address (FR1.A)
 - Line 1
 - Line 2 (optionally)
 - Town or City
 - County (optionally)
 - Post Coded

The Details gathered are those needed and ONLY those needed. The address has been taken in using the official gov.uk format in respect to the fact that people moving to solar panels may be eligible for a tax return or other “reward” for going green. Compliance with gov.uk address storage from day one prevents need to redevelop later and change the entire database schema.

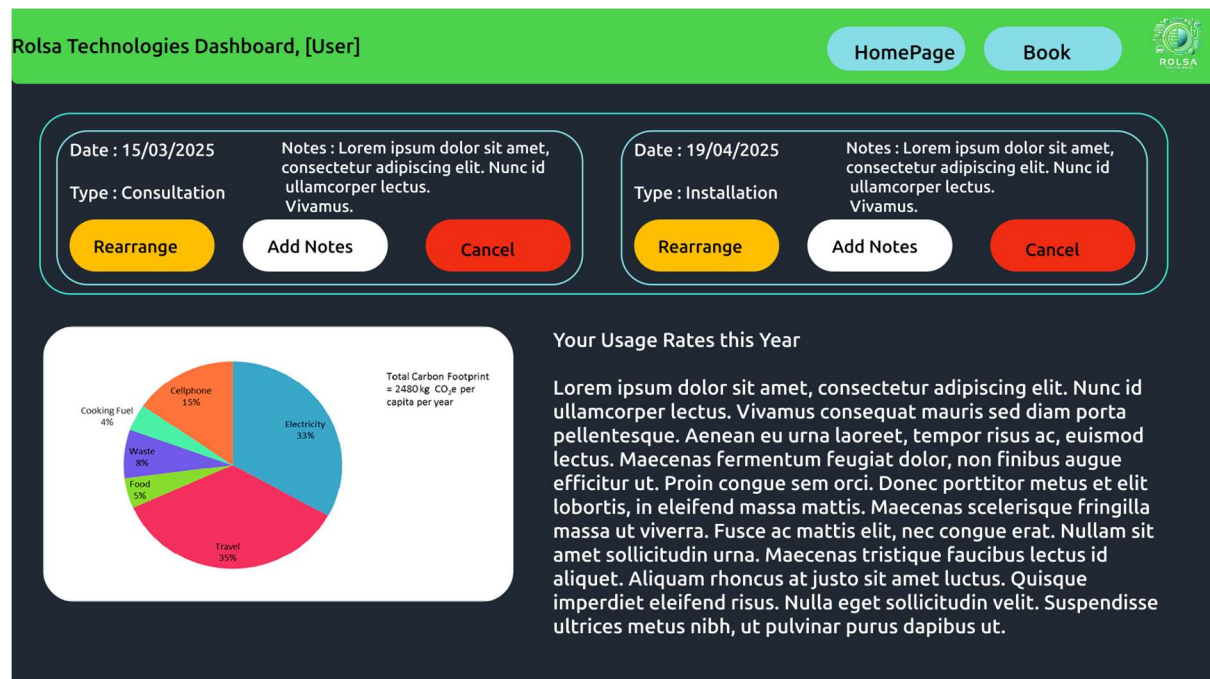
Login Page



The screenshot shows the Rolsa Technologies Login Page. The header is green and contains the text "Rolsa Technologies Login Page" on the left, a "Register" button on the right, and a "HomePage" button further right. The main content area has a dark blue background. In the center, there is a white rounded rectangle containing two input fields: "Email Address" and "Password", both with green placeholder text. Below these fields is a large green "Register" button.

The login page is very simplistic, asking for only an email address and password to search for in the database (FR1.b). Again, these pages are as simplistic as they can be while still maintaining professional look and feel that suits the project and company (NFR3.a)

Logged In Dashboard



The logged in dashboard is where the user lands after a successful login (FR1.b). The dashboard contains information about upcoming appointments, if the user has any (FR3.a), and some information about their product usages and spendings both in a graph and broken down into text (FR2.a). There should not be a situation where the user is met by a blank dashboard as I understand this software is to be rolled out to customers already in the company in which case we can import pre-existing data into the new solution surrounding appointments and statistics.

The user will be able to add notes to the system (FR3.b), rearrange or cancel an appointment at the push of a button (FR3.c). They will be able to see their next two appointments at any one time from this page. To schedule however they will need to use the navigation bar to proceed to the booking page.

Booking Page

Rolsa Technologies Bookings

Homepage Dashboard Logout

ROLSA

Type Of Booking

Address

Date (as YYYY/MM/DD)

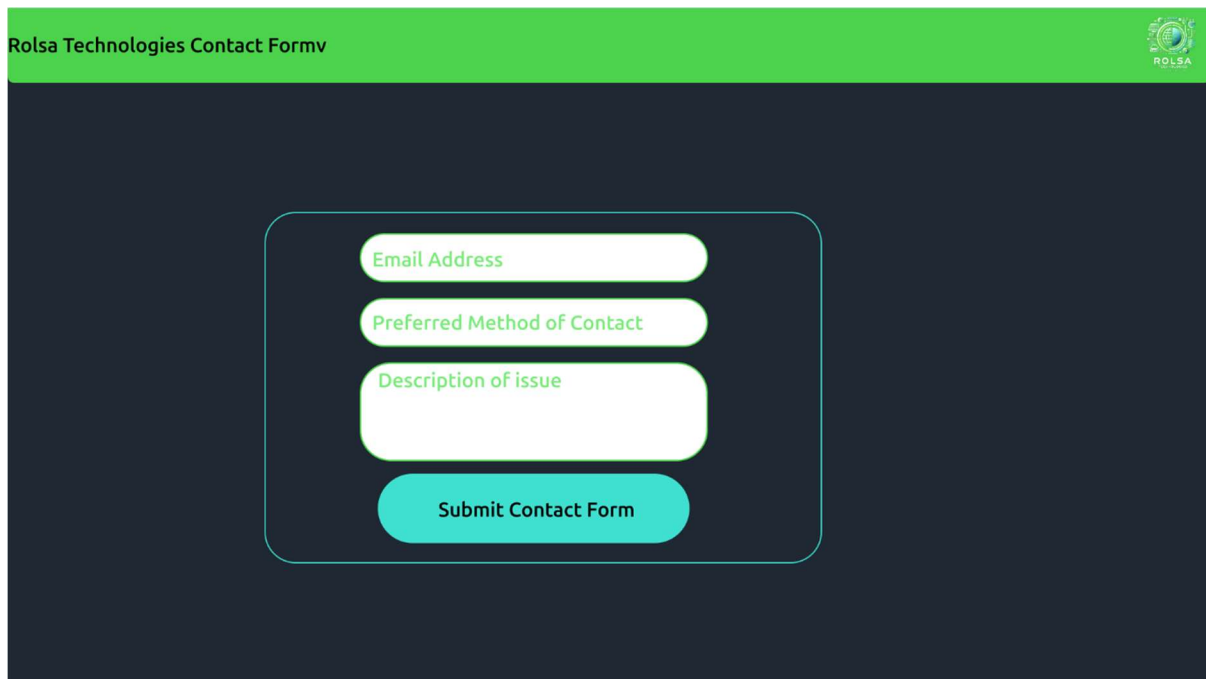
Product

S/N if known

Register

The booking page for the users follows a very similar styling to the registration page in using a centered div to portray information (NFR3.a). Asking for the type of booking (FR3.a), reconfirmation of address (to negate billing & home/site address discrepancies and just to ensure correct details) alongside the product menu that will be a drop down menu to select products identified as owned by the user (these will be tied to their account in the database) or if it is an installation the entire product catalogue will be available. The S/N or serial number input box will be an optional fill however filling it in will allow us to see if the product is within warranty of its manufacturing date.

Contact us



The image shows a contact form titled "Rolsa Technologies Contact Form" with a green header bar. The form is centered on a dark blue background. It consists of four input fields stacked vertically, each with a light blue border and a light blue placeholder text: "Email Address", "Preferred Method of Contact", "Description of issue", and a "Submit Contact Form" button. The button is light blue with dark blue text. A small Rolsa Technologies logo is in the top right corner of the header bar.

Rolsa Technologies Contact Form

Email Address

Preferred Method of Contact

Description of issue

Submit Contact Form

The contact us page again follows the centred div style approach (NFR3.a). It requests the user enter their email address (incase registered email address differs from contact address), a preferred method of contact drop down with phone, email and even in person and a description of the issue box in which users will be expected to produce a small description of the reason for contacting and ideally list hours of availability to be contacted back.

Admin Pages

Super User one-time Registration

Rolsa Technologies one time SUDO Registration

Be aware this is the SUPER USER Creation Screen. If you are seeing this then you are the only user that will ever see this. Your account will be responsible for creating and monitoring all other administrators that may use this system

Email Address

Password

Confirm Password

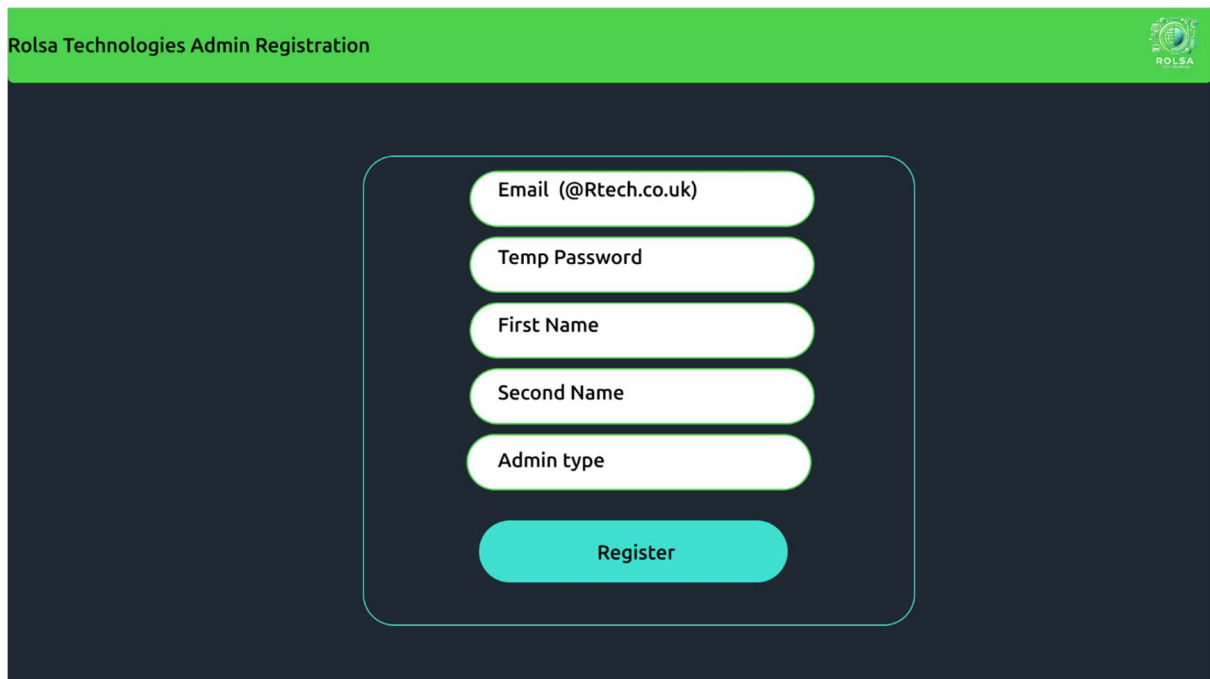
First Name

Second Name

Register

At first loading of the website at the Rolsa Tech office there will be a prompt to create a "SUDO" user or "super" user. This user is essentially the be all and end all of the system. They have the sole power to create other administrators, upgrade and downgrade their level of access and Delete administrators all together (FR4). The screen will only ever appear once, and the credentials would be better off as a specialist "super user" email address rather than tied to an employee who could leave the business.

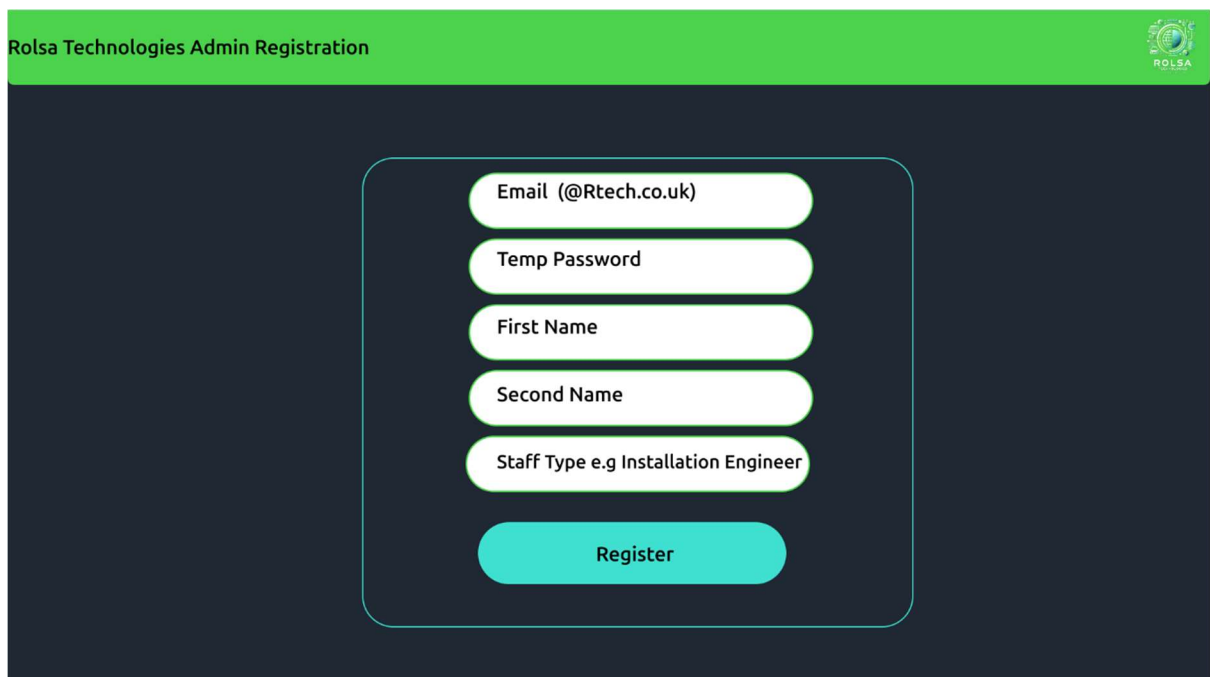
Super User Sub Admin Registration



The screenshot shows a web form titled "Rolsa Technologies Admin Registration" with a green header bar. The Rolsa logo is in the top right corner. The form is centered on a dark blue background and contains the following fields: "Email (@Rtech.co.uk)", "Temp Password", "First Name", "Second Name", and "Admin type". A red "Register" button is at the bottom of the form.

This is yet another screen available to only the "SUDO" user where they can register "sub" administrators (FR4). The admins being registered must have an "@Rtech.co.uk" email address and not having one of these will bounce the admin creation regardless of the "SUDO" doing it (NFR2.a, NFR2.b). This does cause an issue if the company wants to bring a contractor into the system, but this is not something outlined in the design brief and the company is large enough to have a specialist in each area.

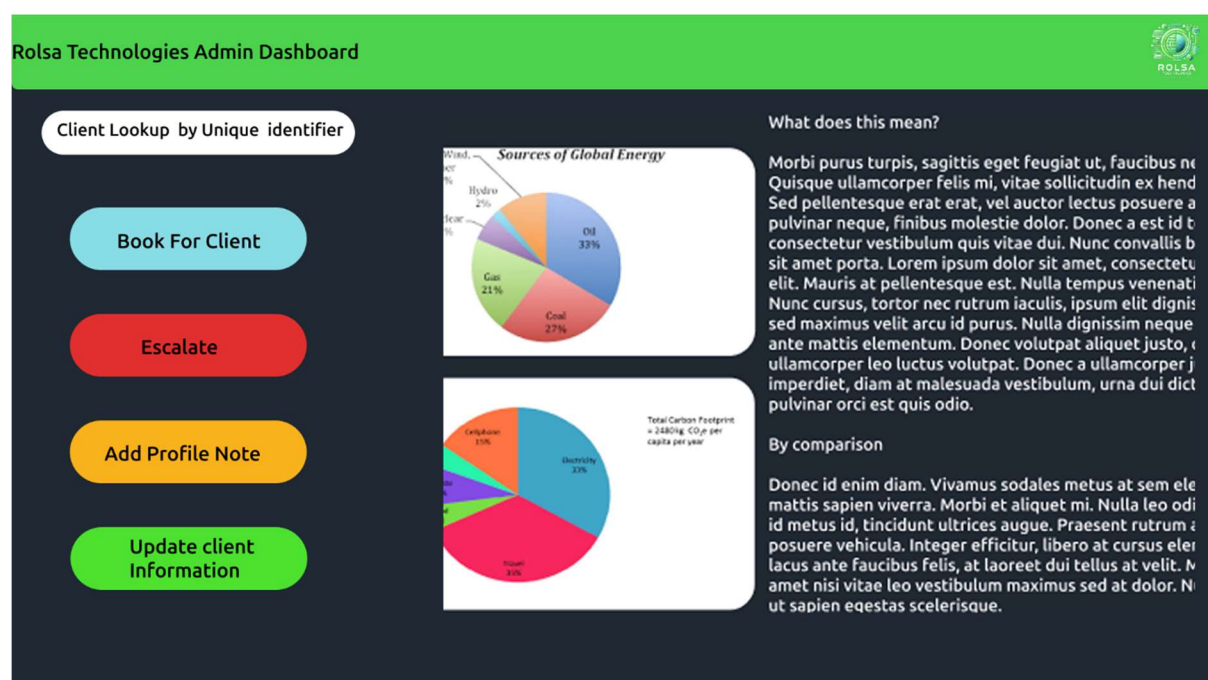
Staff Registration



The screenshot shows a web form titled "Rolsa Technologies Admin Registration" with a green header bar. The Rolsa logo is in the top right corner. The form is centered on a dark blue background and contains the following fields: "Email (@Rtech.co.uk)", "Temp Password", "First Name", "Second Name", and "Staff Type e.g Installation Engineer". A red "Register" button is at the bottom of the form.

The staff creation / registration portal a portal available to all administrators to create a staff member such as an "installation engineer" or "consulting Practitioner" or "Maintenance Technician" (FR4). These again need an "@Rtech.co.uk" email address to allow the creation of the staff member to occur (NFR2.a, NFR2.b). They will have to set up a temporary password for the staff member that will be generated randomly and obfuscated from the administrator to be sent directly to the "@Rtech.co.uk" email associated with the new account.

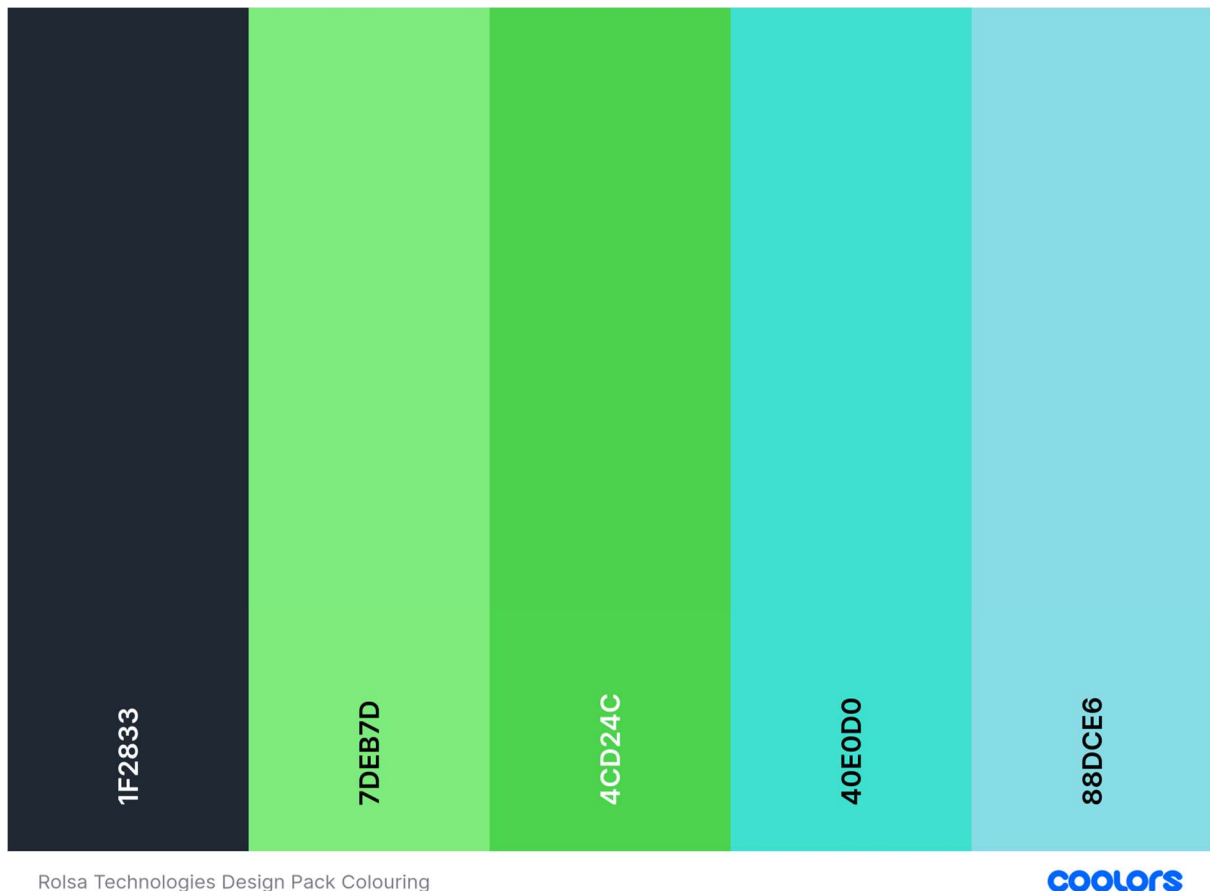
Admin Support Dashboard



The admin support dashboard is a dashboard available to any level of admin as a means of supporting users over the phone (FR4). The user will have to provide their unique identifier (ideally a client number) that when entered onto the system will allow the support staff to see the usages for the client over a period they dictate and other associated things to view (FR4.b, FR2.a).

There will also be buttons where the admin can book an appt for a user on their end if that is necessary (FR3.a), escalate a "support" call if the assistance needed is not in the realm of understanding for the current support member. A way to add profile notes i.e. "large dog, provide 15 mins notification prior to arrival to allow for secure working environment". And finally update client information (FR4.c).

Colour Scheme, Font Pack & Justification



Colour Palette

- Primary Dark: #1F2833 (Dark slate blue - background)
- Light Green: #7DEB7D (Bright mint green - primary accent)
- Medium Green: #4CD24C (Medium green - secondary accent)
- Teal/Turquoise: #40E0D0 (Bright teal - tertiary accent)
- Light Blue: #88DCE6 (Soft sky blue - complementary accent)

Having a heavy focus on green colours and tones helps to frame the website in a “green” and “eco” lighting which is very fitting for the scenario.

Font pack will be the “Sans Serif” family for readability as it is considered one of the more accessible fonts by comparison to others available.

Pseudocode

Registration

```
    IF form is submitted via POST method THEN  
    TRY  
        Include database connection for SELECT operations  
    CATCH DatabaseException  
        Display "Connection error: " + error message  
        Exit program  
    END TRY  
  
    // Sanitize user inputs  
    username = sanitize(POST.username)  
    password = sanitize(POST.password)  
    confirmPassword = sanitize(POST.confirmPassword)  
    firstName = sanitize(POST.firstName)  
    surname = sanitize(POST.surname)  
    email = sanitize(POST.email)  
    signupDate = current date in YYYY-MM-DD format  
  
    // Validate password  
    IF password != confirmPassword THEN  
        Display "Your passwords do not match"  
    ELSE IF password length < 8 THEN  
        Display "Password must be at least 8 characters long"  
    ELSE  
        TRY
```

```
// Check if username already exists

PrepareStatement "SELECT username FROM user WHERE username = ?"

BindParameter 1 with username

ExecuteStatement

result = FetchResult


IF result exists THEN

    Display "User exists, try another name"

ELSE

    TRY

        // Include database connection for INSERT operations

        Include database connection for INSERT


        // Hash the password

        hashedPassword = HashPassword(password)


        // Insert the new user

        PrepareStatement "INSERT INTO user (username, password, f_name, s_name,
email, signup) VALUES (?, ?, ?, ?, ?, ?)"

        BindParameter 1 with username

        BindParameter 2 with hashedPassword

        BindParameter 3 with firstName

        BindParameter 4 with surname

        BindParameter 5 with email

        BindParameter 6 with signupDate

        ExecuteStatement


        Display "Successfully registered"
```

```
        Redirect to "login?success=registered"
    Exit program
CATCH DatabaseException
    Display "Error: " + error message
END TRY
END IF
CATCH DatabaseException
    Display "Error: " + error message
END TRY
END IF
END IF
```

Login

```
TRY
    Include database connection for SELECT operations
    Display "Connected successfully"
CATCH DatabaseException
    Display "Connection error"
END TRY
```

```
Include database connection for SELECT operations
Start session
```

```
IF form is submitted via POST method THEN
    // Sanitize input values
    username = sanitize(POST.username)
    password = sanitize(POST.password)
```

```

// Validate inputs

IF username is not empty AND password is not empty THEN

    TRY

        // Prepare SQL query

        PreparedStatement "SELECT * FROM user WHERE username = :username"

        BindParameter ":username" with username

        ExecuteStatement

        result = FetchResult


        // Verify user credentials

        IF result exists AND VerifyPassword(password, result.password) THEN

            // Store user data in session

            SESSION.user_id = result.id

            SESSION.username = result.username


            // Redirect to dashboard

            Redirect to "dashboard "

            Exit program

        ELSE

            // Invalid credentials

            Redirect to "login?error=invalid_credentials"

            Exit program

        END IF

    CATCH DatabaseException

        Display "Error: " + error message

    END TRY

ELSE

    // Missing credentials

```


Redirect to "login?error=missing_credentials"

Exit program

END IF

END IF

Database Connection

FUNCTION dbconnect_delete()

SET servername = "localhost"

SET dbusername = "delete_user"

SET dbpassword = "delete_password"

SET dbname = "database_name"

TRY

CREATE new PDO connection using servername, port 3306, dbname, dbusername, dbpassword

SET connection error mode to EXCEPTION

RETURN connection

CATCH DatabaseException

LOG "Database error in function: " + error message

THROW exception

END TRY

END FUNCTION

FUNCTION dbconnect_insert()

SET servername = "localhost"

SET dbusername = "insert_user"

SET dbpassword = "insert_password"

SET dbname = "database_name"

TRY

CREATE new PDO connection using servername, port 3306, dbname, dbusername, dbpassword

SET connection error mode to EXCEPTION

RETURN connection

CATCH DatabaseException

LOG "Database error in function: " + error message

THROW exception

END TRY

END FUNCTION

FUNCTION dbconnect_select()

SET servername = "localhost"

SET dbusername = "select_user"

SET dbpassword = "select_password"

SET dbname = "database_name"

TRY

CREATE new PDO connection using servername, port 3306, dbname, dbusername, dbpassword

SET connection error mode to EXCEPTION

RETURN connection

CATCH DatabaseException

LOG "Database error in function: " + error message

THROW exception

END TRY

END FUNCTION

FUNCTION dbconnect_update()

Jack Witney LL-000018212

```

SET servername = "localhost"

SET dbusername = "update_user"

SET dbpassword = "update_password"

SET dbname = "database_name"


TRY

    CREATE new PDO connection using servername, port 3306, dbname, dbusername,
dbpassword

    SET connection error mode to EXCEPTION

    RETURN connection

CATCH DatabaseException

    LOG "Database error in function: " + error message

    THROW exception

END TRY

END FUNCTION

```

Logout

```

Start session

IF admin_ssnlogin exists in SESSION THEN

    SET destination = "/admin/admin_login"

ELSE

    SET destination = "index"

END IF

Destroy session

Redirect to "index"

```

Super User Creation

```
// Start session
```

```

session_start()

// Include required files

require_once '../dbconnect/db_connect_master

require_once 'admin_functions

require_once '../common_functions


// Check if super admin already exists
if (super_checker(dbconnect_select())) :

    // Set error message

    $_SESSION['ERROR'] -> "ADMIN ALREADY EXISTS, LOGIN or ASK FOR to be registered"

    // Redirect to login page

    header('location: admin_login)

    exit

:

// Check if form was submitted

elseif ($_SERVER['REQUEST_METHOD'] === 'POST') :

    try :

        // Attempt to register admin and create audit log

        if (reg_admin(dbconnect_insert(), $_POST)

            // Set success message

            $_SESSION['SUCCESS'] -> "ADMIN REGISTERED"

            // Redirect to login page

            header("Location: admin_login ")

            exit

        :

    else :

        // Set error message for unknown failure

```

```

    £_SESSION['ERROR'] -> "SUPER FAIL, UNKNOWN ERROR"

    // Redirect back to registration page
    header("Location: one_time_super ")
    exit

:

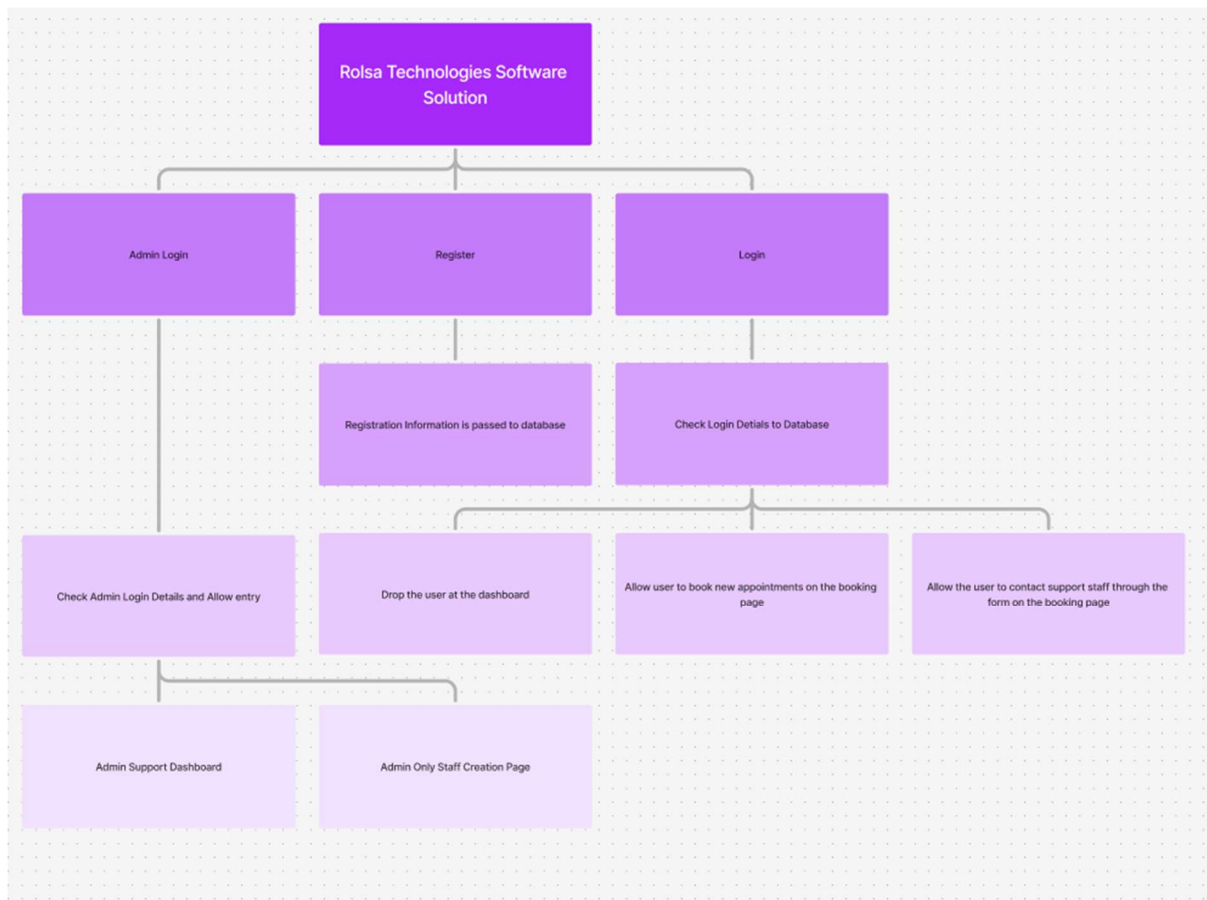
:
catch (Exception £e) :
    // Handle exceptions with error message
    £_SESSION['ERROR'] -> "SUPER REG ERROR: " . £e->getMessage()
    // Redirect back to registration page
    header("Location: one_time_super ")
    exit

:

:

```

Top level Decomposition Diagram



Data Flow Breakdown

User Data Flow

User Registration Flow

- User submits registration data (email, password, first name, second name) via registration form
- System validates registration data
- System stores user data in Users table
- System creates user session and redirects to Dashboard

User Authentication Flow

- User submits login credentials (email, password) via login form
- System validates credentials against Users table
- System creates user session and redirects to Dashboard

- Alternative flow: System returns error and redirects back to login page

Booking Creation Flow

- Authenticated user submits booking details (type, address, date, product, S/N)
- System validates booking data
- System checks booking_types table for required specialty
- System assigns available staff based on specialty_required
- System creates booking record linking user_id and staff_id
- System confirms booking and updates user Dashboard

Dashboard Data Retrieval Flow

- User requests Dashboard view
- System queries Bookings table for user's bookings
- System queries booking_types table for booking details
- System retrieves staff information for each booking
- System presents booking information, with options to rearrange, add notes, or cancel

Contact Form Flow

- User submits contact form (email, preferred method, issue description)
- System validates contact data
- System stores contact request
- System confirms submission to user

Admin Data Flow

User Registration Flow

- User submits registration data (email, password, first name, second name) via registration form
- System validates registration data
- System stores user data in Users table
- System creates user session and redirects to Dashboard

User Authentication Flow

- User submits login credentials (email, password) via login form
- System validates credentials against Users table
- System creates user session and redirects to Dashboard
- Alternative flow: System returns error and redirects back to login page

Booking Creation Flow

- Authenticated user submits booking details (type, date)
- System validates booking data
- System checks booking_types table for required specialty
- System assigns available staff based on specialty_required
- System creates booking record linking user_id and staff_id
- System confirms booking and updates user Dashboard

Dashboard Data Retrieval Flow

- User requests Dashboard view
- System queries Bookings table for user's bookings
- System queries booking_types table for booking details
- System retrieves staff information for each booking
- System presents booking information, with options to manage bookings

Contact Form Flow

- User submits contact form (email, preferred method, issue description)
- System validates contact data
- System stores contact request
- System confirms submission to user

Admin Management Flow

- Admin logs in through admin portal
- System verifies admin credentials and admin type
- System provides dashboard with client lookup, booking management, and profile management
- Admin can create/modify bookings, update client information, add profile notes, and escalate issues

Test Strategy

Please see “Task1_DesignDocs_TestStrategy_000018212_Witney_J”