

Development Log for Rolsa Technologies Solution



Contents

Functional and Non-Functional Requirements	4
Functional Requirements	4
1. Account Systems	4
2. Metric Gathering	4
3. Ability to book Appointments	4
4. Admin Panel	4
Non-Functional Requirements	5
1. Performance and Speed	5
2. Security and Protection of Sensitive Data.....	5
3. Usability	5
Versioning	5
Version 1: Foundation & Authentication.....	5
Version 2: Core Functionality.....	6
Version 3: Visual and Final Touches.....	6
Version 1 During Dev Writeup.....	6
DB Credentials:	6
Insert	7
Update	7
Delete	7
Select	7
Importing SQL Schema	9
Creating DB_Connect	10
Database Change	11
Admin Login	12
Admin Registration / Adding	12
User Login and Registration	12
Funcs and Admin Funcs.....	14
Admin Funcs	14
User Funcs	15
Version 1 Retrospective	17
Version 1 Outline Predevelopment:	17

Developer Thoughts:.....	17
Database Design Issues	18
Parameter Binding Problems.....	18
Security Considerations	18
Version 2 During Dev Writeup.....	19
Version 2 plan	20
Add Staff Page	20
Booking Appointments	21
Version 2 Retrospective	24
Version 3 During Dev Writeup.....	26
CSS and Styling	26
Carbon Calculator	27
Static Pages	27
Version 3 Retrospective	29
Version 3 Outline:	29
Developer Thoughts.....	29
Evaluation of Built Solution.....	30
Functional Requirements	30
5. Account Systems	30
6. Metric Gathering	30
7. Ability to book Appointments	31
8. Admin Panel	31
Non-Functional Requirements	31
4. Performance and Speed	31
5. Security and Protection of Sensitive Data.....	31
6. Usability	31
Targets (FR/NFR).....	32
Database Design Issues.....	33
V4 Plan	33

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 Rolsa 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

Versioning

Version 1: Foundation & Authentication

Version 1 sets up my database structure and secure connections, plus all the login systems. I'll implement one-time superuser setup and complete registration/login flows for both admins and regular users (FR1.a, FR1.b). This version is all about creating secure entry points with proper validation (NFR2.b) before adding any fancy features.

Version 2: Core Functionality

Version 2 brings the application to life with the actual booking system and staff management tools (FR3.a, FR4.c). I'll develop smart appointment scheduling with time-conflict prevention (FR3.c) and add features to manage staff members. Looking at adding a calculator feature, though need to be careful about regulatory compliance. This phase turns the login shell into a working Minimally Viable Product (MVP).

Version 3: Visual and Final Touches

Version 3 is where everything gets pretty! I'll focus on attractive CSS styling, responsive design, and creating a cohesive look across the platform (NFR3.a, NFR3.b). There won't be much new PHP code, mainly static information pages and design refinements. This final update transforms the functional product into something that looks professional and most importantly contains all the company colouring and branding as close to the design pack images as possible.

Version 1 During Dev Writeup

Priority for version 1 is to create the database in my development environment. Using the design created during the design period, I am going to use the built-in export function from DrawSQL to get the needed Data Definition Language (DDL) to build my database in an efficient manner (NFR1.b). I am aware that the export from DrawSQL can at times be problematic and not always entirely correct. My approach to correcting this is going to be to head to Claude.ai and utilise it to alter the SQL into efficient and complete data rather than the convoluted mess that DrawSQL makes.

DB Credentials:

I have decided that I am going to create 4 different users for my Database Connections to further secure my system beyond having the one specialist user. The four users will be:

Insert

Username: rolsa_insert

Password: R0ls4123@#

Update

Username: rolsa_update

Password: R0ls4123!!

Delete

Username: rolsa_delete

Password: R0ls4123\$*

Select

Username: rolsa_select

Password: R0ls4123%”

Before I can create the four users however I need to create my database, the easiest way to do this is to create a new user and tick the box to create a new database with the same name. I have chosen to name my new user “rolsa_technologies” with password “R0ls4123££” to ensure the database has a conventional name. This user will not have access to anything in the database nor will it be referenced in any codebases.

Add user account

Login Information

User name:

Use text field

rolsa_technologies

⚠ An account already exists with the same username but possibly a different hostname.

Host name:

Local

localhost

?

Password:

Use text field

Strength: Good

Re-type:

Authentication plugin

Native MySQL authentication

Generate password:

Generate

Database for user account

☒ Create database with same name and grant all privileges.
 ☐ Grant all privileges on wildcard name (username_%).

Next, I need to go the Privileges header once I have clicked into the “rolsa_technologies” database I created and begin adding in my secure user accounts.

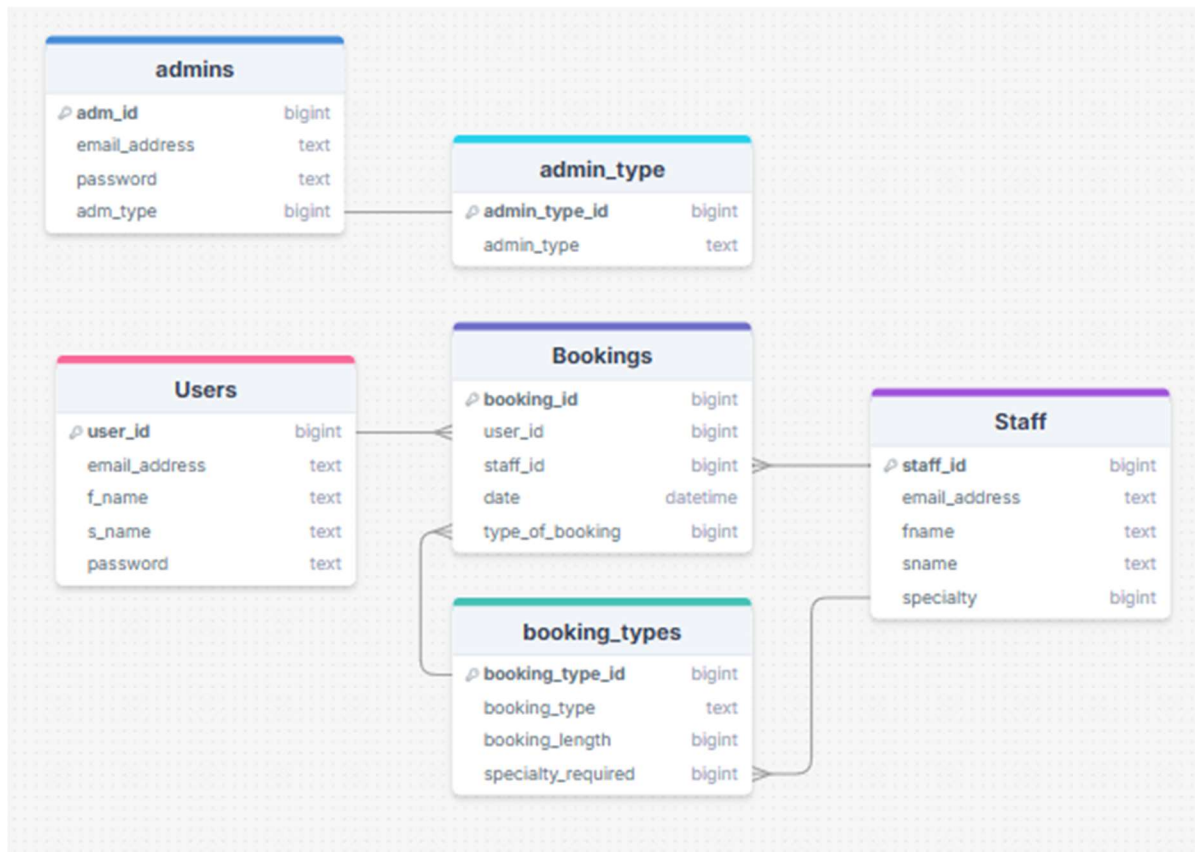
Users having access to "rolsa_technologies"

	User name	Host name	Type	Privileges	Grant	Action
<input type="checkbox"/>	mysql.infoschema	localhost	global	SELECT	No	Edit privileges Export
<input type="checkbox"/>	rolsa_delete	localhost	database-specific	DELETE	No	Edit privileges Export
<input type="checkbox"/>	rolsa_insert	localhost	database-specific	INSERT	No	Edit privileges Export
<input type="checkbox"/>	rolsa_select	localhost	database-specific	SELECT	No	Edit privileges Export
<input type="checkbox"/>	rolsa_technologies	localhost	database-specific	ALL PRIVILEGES	No	Edit privileges Export
<input type="checkbox"/>	rolsa_update	localhost	database-specific	UPDATE	No	Edit privileges Export
<input type="checkbox"/>	root	localhost	global	ALL PRIVILEGES	Yes	Edit privileges Export

Now I have created the users on my local database. Each user has ONLY Its corresponding name in permissions. This is to protect the integrity of the data and avoid a scenario where an end user, malicious or accidental, manages to get a hold of a root-esc privilege access account (priv acc).

Importing SQL Schema

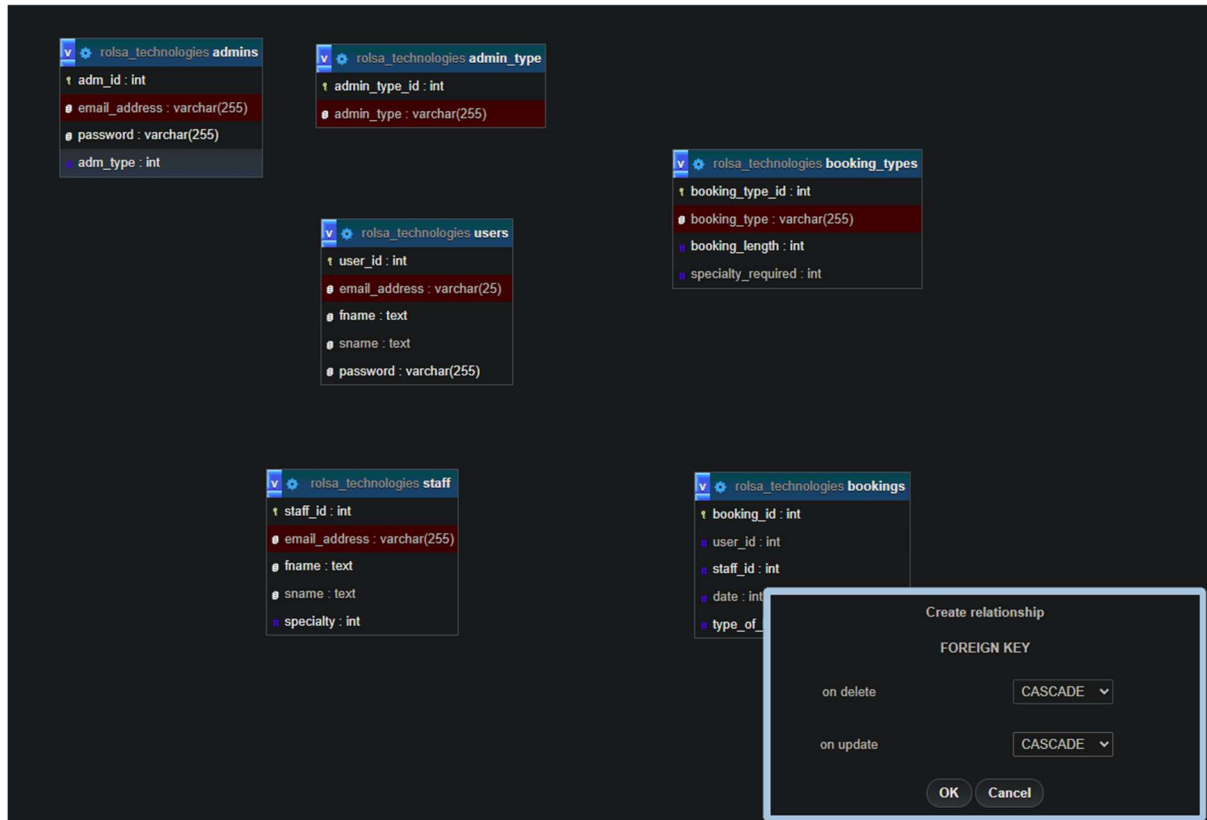
Now its about time to populate the database with the tables from my schema created during the design period.



With the predesigned schema I have created all the tables designed in the previous period.

Table	Action
<input type="checkbox"/> admins	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> admin_type	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> bookings	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> booking_types	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> staff	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop
6 tables	Sum

Now to ensure referential integrity and data sensitivity I am going to set the tables up properly using the foreign key system and using “cascade, cascade” in my integrated designer to employ checking to ensure an appointment cannot be made for a user that does not actually exist and any appointments for a user and deleted when the user is deleted / removed from the database.

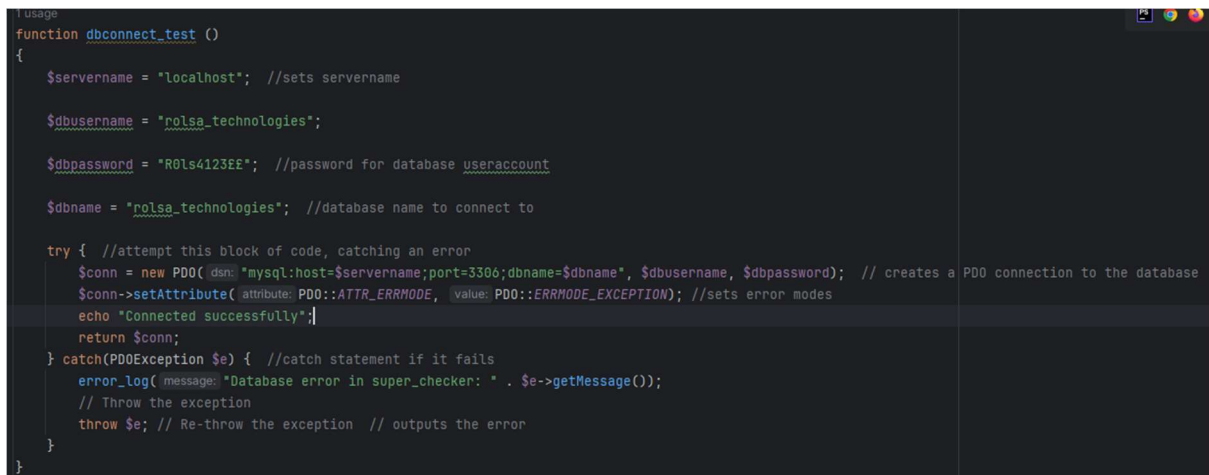


The relationship constraints have now been set and foreign key links created between the necessary tables, its time to move towards creating a database connection in my PHP interpreter.

Creating DB_Connect

Using my earlier described approach to database connection and querying I have created a “db_connect_master.php” file that contains functions to Insert, Update, Select and Delete data as and when the relevant function is called(NFR1.b). I have also created a function called “dbconnect_test” for the purpose of testing the database

connection, as pictured I can see that it works successfully.



```
usage
function dbconnect_test ()
{
    $servername = "localhost"; //sets servername

    $dbusername = "rolsa_technologies";

    $dbpassword = "R0ls4123Ef"; //password for database useraccount

    $dbname = "rolsa_technologies"; //database name to connect to

    try { //attempt this block of code, catching an error
        $conn = new PDO( dsn: "mysql:host=$servername;port=3306;dbname=$dbname", $dbusername, $dbpassword); // creates a PDO connection to the database
        $conn->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION); //sets error modes
        echo "Connected successfully";
        return $conn;
    } catch(PDOException $e) { //catch statement if it fails
        error_log( message: "Database error in super_checker: " . $e->getMessage());
        // Throw the exception
        throw $e; // Re-throw the exception // outputs the error
    }
}
```

Now that we have a Database connection successfully working I am going to work on the administrator panel section of my version 1 plan. Step one of that is to work on the “adm_funcs.php” to build the super checker function to check for a super user or SUDOer to ensure the code behind the process is robust and reusable.

I have also built the register_adm function and valid_email function again this is for robustness and reusability of code in a wide manner of senses.

Database Change

After my first day of development I have realised that my system for admin types in the database having a separate table is not the most efficient way to do this. I have instead opted to change the data type of "adm_type" to a Text field and directly log the type of administrator to the admins table (NFR1.b). The new types for administrators will be "super", "admin" and "privileged_user". These will be incorporated into the system in the same manner as the previous version while also allowing an easier process without compromising security (NFR2.b).

The 1-time Sudo user registration will force type to be “super” in the database and then a dynamic dropdown for the options within the admin registration page that can only be accessed as a “super” user privilege.

Admin Login

Now that the Super user has successfully been made as a onetime system, I have set the system so that if that page is ever accessed again, it is an instant route straight to the admin login page (NFR2.b). This page queries for the email and password against the admins table and logs the user in if they are correct credentials (FR1.b). If this is not the case, then the user is rerouted back to the admin_login page to try again with their login. A Successful login will send the user to an "admin_dashboard". I have also created a logout script to terminate the login. This has been built ahead of time to allow me to log out of the super user accounts as it was being stored in the session and not terminated between iterations of code.

Admin Registration / Adding

Next on the list is the admin registration page (FR1.a). This page is going to be designed to be only accessible to the "super" adm_type to ensure that no other admins can create a new admin at their user level (NFR2.b). Like the 1-time super user page this page will redirect to admin_dashboard if an inappropriate access level is able to get onto the page.

I have written this page myself and I am unable to figure out what is wrong with it and why I am constantly sent to "index" despite no reference to index at all in the codebase. I am going to turn to AI to see if I can query the problem and fix it from there.

After AI turned out to be unable to assist in the issue I further investigated and realised I had created a paradox for myself where I had a session variable referenced by a different name in the admin_add page to what it was referred to as in the actual admin registration function leading to a loop of errors.

		adm_id	email_address	password	adm_type
<input type="checkbox"/>	 Edit  Copy  Delete	3	steve@rtech.co.uk	\$2y\$10\$gEsv/Zq1V/cFyatqlmQ8HuEhjRvegknQY9237uGPef8...	super
<input type="checkbox"/>	 Edit  Copy  Delete	4	sbeve@rtech.co.uk	\$2y\$10\$NyqoA6CCGT9mficwUi40s.994QZnTHnKals/A36a1nW...	admin

User Login and Registration

Now that the Administrator login and registration is created, I am going to replicate the processes to allow for the creation of user accounts (FR1.a, FR1.b). During this time, I have realised that I have not at all allowed for a way to store an address in my database. This is an issue I am going to now have to rectify on my database engine before continuing.

Before:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 user_id	int			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 email_address	varchar(25)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	3 fname	text	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	4 sname	text	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	5 password	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More

After:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 user_id	int			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 email_address	varchar(25)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	3 password	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	4 fname	text	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	5 sname	text	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	6 addressln1	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	7 addressln2	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	8 city	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	9 postcode	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	10 phone	text	utf8mb4_0900_ai_ci		No	None			Change Drop More

I have added the relevant fields required for an official post office / gov style address form gathering (NFR2.b). It seems that I discussed this in my design pack but did not reflect it in my database design. Unfortunately, the design pack is already submitted and cannot be tweaked but as this was a deliverable I will be including it regardless of the database design submitted as part of the design packs.

As part of the user registration process (FR1.a), I had of course set parameters to be inserted into the database. In this process I had made the mistake of binding my password parameter to field 4 when in fact it is in field 2 of my database. This issue led

to a catastrophic error when attempting to log in. My testing browser had offered to save my password field on entry and of course provide it auto filled for later use.

Unfortunately, this was not even remotely correct because of the database discrepancy. Having noticed this error, it is now corrected, and the registration and login system is a smooth process (NFR3.a). This error was partially because of adding the address fields into my database late as this caused me to rewrite my database connections and other functions that inevitably boils down to where the parameter binds became misconstrued.

Functs and Admin Functs

To build a readable, reuseable and maintainable codebase I have opted to build a lot of my processing code into functions that can be called at any time for use. These functions have been referred to earlier in my development log but not in detail. These functions are split between Administrator functions and User Functions. The only large difference in the two is that the admin functions connect to the admin table and have higher privilege, where the user functions of course connect to the user table.

A few of these functions are not yet in use however given that I have the knowledge of how to build them and what I want them to do as well as the data I know they will be processing I chose to build my entire function set in version 1 to allow for ease of further development in version 2 and beyond.

Admin Functs

`admin_error(&$session)`: I created this function to handle admin error and success messages through session variables, similar to `usr_error()` but with a key difference - it uses pass by reference instead of by value. This allows me to directly modify the session variable data properly within the function. It checks for error or success messages in the session, returns them with appropriate prefixes, and cleans up by unsetting them.

`sudo_check($conn)`: I initially struggled with this function which checks if a superuser admin exists in the database. My original version (commented out) had issues I couldn't identify quickly, so I revamped it with some AI assistance. The new version correctly queries for admins with type 'super' and properly fetches and checks the results. This was a good learning experience for me on proper PDO result handling.

`valid_email($email)`: I implemented this to validate admin email addresses by ensuring they contain the specific domain "@rtech.co.uk". It's a simple check using `strpos()` to

verify email format, though I noticed it might have a logic issue since strpos returns the position (which could be 0) or false.

register_admin(\$conn, \$post): I built this to handle admin registration by inserting new admin details into the database. It first validates that email and password fields are provided, then checks if the email has the correct domain using valid_email(). I used prepared statements with proper parameter binding for security and made sure to hash passwords. The function includes comprehensive error handling and closes the database connection when finished.

User Functns

pwd_checker(\$password, \$cpass): I created this function to validate password inputs by checking two conditions - if they match and meet the minimum length of 8 characters. It returns true only when both conditions are met. It's simple but effective for our basic password validation, though I might enhance it with complexity requirements later.

usr_error(): I implemented this to manage user notifications through session variables. It checks if error or success messages exist in the session, returns them with appropriate prefixes, and cleans up by unsetting the variables. This gives me a clean way to display transient messages across page loads.

only_user(\$conn, \$username): I designed this to verify if an email address already exists in our users table. I used prepared statements for SQL injection protection and proper error handling with PDO exceptions. It returns true if the email exists, false otherwise, which helps me prevent duplicate registrations.

reg_user(\$conn, \$post): I built this to handle user registration by inserting new user details into the database. I made sure to validate required fields, use prepared statements for security, and properly hash passwords. I included comprehensive error handling with specific exceptions for different failure scenarios and close the database connection when finished.

`validlogin($conn, $email_address)`: I wrote this function to check if a user exists in the database during login. It's a simple function that returns true if a matching email is found, false otherwise. I've fixed the column name to properly use "email_address" in the SQL query for consistency.

`pswdcheck($conn, $email_address)`: I created this to verify a user's password by comparing the hashed password in the database with the submitted one. I'm using `password_verify()` for secure comparison and proper error handling. Returns true if passwords match, false otherwise.

`get_userid($conn, $post)`: I implemented this to retrieve a user's ID from the database based on email address. I use it after registration to store the user ID in session data. I made sure to use prepared statements and proper error handling, cleanly closing the database connection when done.

`get_time($choice)`: I developed this to generate formatted date-time strings for appointment scheduling. It can either use current time or add one week, depending on the parameter. It returns a datetime string formatted for HTML datetime-local input fields.

`get_appt_staff($conn)`: I wrote this to retrieve all staff members from the database with their IDs, names, and roles. I use it to populate staff selection dropdowns for our appointment booking. It uses prepared statements and returns the complete result set.

`valid_staff($conn, $post)`: I created this to validate that the selected staff member has the correct role for the requested appointment type. It checks the staff's role against the appointment type from the form submission, returning true if they match, false otherwise.

`valid_appointment($conn, $post)`: I built this to check if a requested appointment time is valid based on staff availability. It verifies the time is within working hours and doesn't conflict with existing bookings. I still have some debugging code in there ("echo last check") that I need to remove before going to production.

`in_working_week($datetimeLocalValue)`: I implemented this to validate if a given appointment time falls within our working hours (8 AM to 4 PM, Monday to Friday). This ensures appointments are only scheduled during business hours on weekdays.

`booking_time_check($existingBookings, $selectedtime)`: I created this to prevent booking conflicts by ensuring appointments are at least 90 minutes apart. It compares the selected time against existing bookings and returns false if any conflict is found.

`commit_booking($conn, $post)`: I developed this to insert new bookings into the database. It converts the datetime-local input to epoch time, uses the current user ID from session data, and I've left a placeholder for the product field that I still need to implement.

`appointment_timings($datetimeLocalValue, $epochTimes)`: I started working on this for version 2 of our booking system. It will check if appointments are at least 60 minutes apart on the same day, but it's still incomplete. I need to add a proper return statement for all cases.

Version 1 Retrospective

Version 1 Outline Predevelopment:

Version 1 sets up my database structure and secure connections, plus all the login systems. I'll implement one-time superuser setup and complete registration/login flows for both admins and regular users. This version is all about creating secure entry points with proper validation before adding any fancy features.

Developer Thoughts:

Version 1 of the system designed for Rolsa Technologies has ticked all the things I had outlined to do. I have created all the functionality designed with good effect and in good time. Version 1 also contains a few functions not yet called, this was a deliberate decision to build functions I know I will need all at the same time as they follow similar syntax and were quite easy to manipulate for each use case. They also help me in advance know what my post variables need to be named for the most efficient development going ahead. The development was not without hardships though, the ignoring of the address entering to my database caused by a poor design of the users table in the design pack lead to having to alter the entire user table and registration form after creation. This change also affected the database insert as when I added the new fields, I did not correctly enumerate them accordingly to their corresponding database location. This led to posting the hashed password to the surname column and an extended period of confusion and issues before version 1 could be wrapped up.

Database Design Issues

The database I had originally designed in phase 1 of the solution creation had 6 tables, after a brief review during the administrative section development I found that my proposed methodology to identifying administrative privilege levels through a separate “admin_type” table linked via an “adm_type_id” key was an impossible manner of doing so with my current understanding of databases and PHP. As a result, I changed my database to reflect this page, dropping the “admin_type” table in favour of changing the “adm_type” field in the “admin” table to be reflective of a text type rather than an integer and directly inserting the adm_type through a dynamic dropdown menu in the admin creation page of my solution.

After this issue I further reviewed my database and realised that I had also left out the entire address storage from my users table. Upon discovering this during my user signup testing I acted swiftly to update the database accordingly and further to that I ensured my user registration function was taking the relevant fields. This did however lead to an issue around my parameter binding and further database insertion.

Parameter Binding Problems

At the time of updating my database to reflect the previous mistake surrounding address logging inside my users table I also went to add new parameter binds and insertion to my user registration function. While doing this I made the mistake of assigning the wrong field binds to the wrong packets of information. I assigned my hashed password to the “sname” field rather than to the password field due to location of each in my database. This led to a large issue when attempting to log in. This came as the browser I was using to live develop and test was storing the correct password entered but due to the insertion of data being incorrect the login system was comparing the password entered to the password stored which in the relevant field turned out to be the “sname” variable and as a result the password was only correct on entering something silly like “Wozniak” if the first name was “Steve” and taking the “sname” as being an outrageous PHP hash. This issue was a costly oversight in terms of development time because of one single number being incorrect.

Security Considerations

The decision to have 4 separate user accounts in the database each of which can only do their named operation was the best decision made in the solution so far. None of

them allow root access and the account with root access is not named anywhere in the system which means the system is reinforced from root attacks.

Insert

Username: rolsa_insert

Password: R0ls4123@#

Update

Username: rolsa_update

Password: R0ls4123!!

Delete

Username: rolsa_delete

Password: R0ls4123\$*

Select

Username: rolsa_select

Password: R0ls4123%”

This decision proved to be incredibly easy to implement as it something I have built time and time again, the preface of the code never changes, and it is always only different in the credentials themselves. Having these 4 operational accounts has provided tight security from the Get-Go and will continue to provide the best possible security within my development environment all the way through version 3 and beyond.

If I had decided to go with this 4 account approach without building a function it may have been that it was too time consuming given needing frequent reference back to this document where the credentials are documented however due to the intuitiveness of the functions it has sped my development up significantly thus far.

Version 2 During Dev Writeup

Version 2 plan


Version 2 brings the application to life with the actual booking system and staff management tools. I'll develop smart appointment scheduling with time-conflict prevention and add features to manage staff members. Looking at adding a calculator feature, though need to be careful about regulatory compliance. This phase should turn the login/registration shell into a functional Minimally Viable Product (MVP).

Add Staff Page







My first port of call for version 2 is to build my staff adding page into the admin side of things. I will be using the add_admin page and tweaking the page to fit my new use case as it is essentially identical. I am going to make use of my predefined functions from version 1 regarding adding staff members.

I have realised that my database has a design flaw in my staff table. I have not queried for a password anywhere in my sign up process and as a result if I was to complete the process as designed in my function I would be unable to log any correct data or log in. In order to correct this I am going to update my staff table and insert a new password field directly after the email column. After my issue in version 1 with parameter binding I will also be going back through my function and updating the order of binding to ensure no other issues arise.

Before:

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	staff_id 	int			No	None		AUTO_INCREMENT	 Change  Drop  More
<input type="checkbox"/>	2	email_address	varchar(255)	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	3	fname	text	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	4	sname	text	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	5	specialty	int			No	None			 Change  Drop  More

After:

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	staff_id 	int			No	None		AUTO_INCREMENT	 Change  Drop  More
<input type="checkbox"/>	2	email_address	varchar(255)	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	3	password	varchar(255)	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	4	fname	text	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	5	sname	text	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	6	specialty	int			No	None			 Change  Drop  More

The staff adding will need to leverage the “valid_email” function to ensure that a staff member being added has the correct domain address (@rtech.co.uk). This is a security increase as it ensures that no staff can be added if they do not have the relevant email address. While this does in theory make life difficult for external contractors in the future they will have to be provided with contractorXY@rtech.co.uk emails to work with the valid_email checking rather than removing and redeveloping it.




















The Add staff page provided some hardship in its development. Namely correctly referencing the email validation and only allowing “@rtech.co.uk” emails to be passed into the database. Through 3 different iterations of test this failed to work. I had assigned the posted email address variable from the form into a new \$email_address variable after running the valid email function on the posted variable. Unknowingly I had just addressed the new, non posted variable with a boolean value as I had incorrectly used my own function. This led to the code I had written having no effect on the data or the database it was inserted to. I also directly threw the valid email function onto the email address variable at the time of parameter binding which led to the email field in the database just being blank, which admittedly I did not know was ever remotely possible. Finally I realised that it was going to be eaier and more effective (not hard though) than my previous two solutions. I created an If statement to do the comparison which has now led to the email adres being checked appropriately and only sending data through when it is in line with the valid email strpos.

The code does now work, and I will be creating staff relevant to the next stage of booking appointments.


Booking Appointments

First course of action for the booking system is to review my database and enquire if any more changes may be needed. From looking at it I can see that, while not needed, the lack of a products header in the bookings table is an omission that cannot be ignored. I will be updating the table accordingly and inserting a product header between data and type_of_booking.

Before:

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	booking_id 	int			No	None		AUTO_INCREMENT	 Change  Drop  More
<input type="checkbox"/>	2	user_id 	int			No	None			 Change  Drop  More
<input type="checkbox"/>	3	staff_id 	int			No	None			 Change  Drop  More
<input type="checkbox"/>	4	date	int			No	None			 Change  Drop  More
<input type="checkbox"/>	5	type_of_booking 	int			No	None			 Change  Drop  More

After:

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	booking_id 	int			No	None		AUTO_INCREMENT	 Change  Drop  More
<input type="checkbox"/>	2	user_id 	int			No	None			 Change  Drop  More
<input type="checkbox"/>	3	staff_id 	int			No	None			 Change  Drop  More
<input type="checkbox"/>	4	date	int			No	None			 Change  Drop  More
<input type="checkbox"/>	5	product	varchar(255)	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	6	type_of_booking 	int			No	None			 Change  Drop  More

Now that the database has been changed to include the product field, I will be able to create a more complete booking system with all the necessary information to go forward into version 3.

The booking system has proven to be quite a tricky system to build so far, I am finding that the PHP required is not PHP I have done much of before and as such proving to be a difficult page to wrap my head around and even begin to build a working version.

While editing my database I realised that my functions would be wrong as a result. I have gone through and systematically updated and upgraded what was needed in each of them. However, the development is still not a smooth process, and I expect it will take a lot of iteration and attempts to achieve a successful version. I may have to extend

the rest of my PHP / Processing code into version 3 to keep on track for version production timelines.

After some iterative nonsense trying to pull together a codebase from my previously made pages, I scrapped that idea off and decided to start working from scratch.

Initially, rereading my functions for a third time I was finally happy with how id built them. They made logical sense in what they did, how they did it and why. Building on these I decided to create the core booking page.

To make the development of the page easier, I am going to insert some records into the staff database to make testing easier.

After a bit of time away from the process, over the weekend, I have come back to the solution with fresh eyes. Reviewing my codebase again, specifically my predefined functions from version 1, I had understood it. I'd tried to run before I could walk and created a whole tangled mess of logic.

Taking a current look at trying to create a booking this error pops up: (also referenced as FIG12 in test log)

Fatal error: Uncaught Exception: Booking error: SQLSTATE[42000]: Syntax error or access violation: 1142 SELECT command denied to user 'rolas_insert'@'localhost' for table 'booking_types' in H:\Task 2 Occ Spec\MY work\Version 2 - REG STAFF, BOOKING AND UNBOOKING\funcs.php:302 Stack trace: #0 H:\Task 2 Occ Spec\MY work\Version 2 - REG STAFF, BOOKING AND UNBOOKING\book_appt.php(17): commit_booking(Object(PDO), Array) #1 {main} thrown in H:\Task 2 Occ Spec\MY work\Version 2 - REG STAFF, BOOKING AND UNBOOKING\funcs.php on line 302

I can see from re-reading my function that it refers to multiple “dead” variables because the table they were in has no real function after the booking table refactor to include types and products in it. I am going to rewrite the function to only refer to the bookings table and correctly assign variables to fields in the database.

After a quick revamp and posting of one more variable I have fixed what I believe to be the last issue surrounding the booking system and will now move back to testing.

Immediately I have found that the user_id is now not correctly being passed into the function to commit the booking which is leading to a failed data commit as it cannot have a null user_id. This is a result of using a Boolean operator to check if my user is logged in and not storing the user_id in its intended format. To combat this, I have redesigned the login script to store the user_ID as well as a true ssn variable.

Jack Witney LL-000018212

Looking at the refined booking system I was able to leverage my newfound fresh eyes to string together the complex multi-function using query to verify the staff member, verify the appointment was within the constraints of the working week with my valid appointment function before finally using my commit booking function to insert the whole ordeal into the database.

The system correctly identifies If a user wants to book an incompatible member of staff for their appointment type and vice versa. Stopping garbage data entering the database.

I plan to investigate the use of some JavaScript to influence the type of installation field to become dynamic based on the type of staff member chosen. However, I will not be doing this until version 3 as there is no real functionality to it and it is more of a Quality-of-Life feature.

Version 2 Retrospective

Version 2 of the system designed for Rolsa Technologies has accomplished most of the goals I had outlined. I have successfully implemented the staff management functionality (FR4.a) and laid the groundwork for the booking system (FR3.a, FR3.b), turning the login shell created in Version 1 into a working Minimally Viable Product (MVP). Like Version 1, this development phase wasn't without its challenges, particularly around the booking system which proved more complex than initially anticipated. Some implementation aspects had to be pushed to Version 3 to maintain production timelines, but the core functionality is now in place.

Staff Management Implementation

The staff addition page was my first focus for Version 2, adapting the existing add_admin page to fit this new use case (FR4.a). During implementation, I discovered a critical design flaw in my staff table - I had omitted the password field in my database design, which would have prevented proper data storage and login capabilities. This necessitated updating the staff table structure by inserting a password field directly after the email column. I also revisited my parameter binding order to prevent issues like those encountered in Version 1.

Security was enhanced by leveraging the "valid_email" function to ensure that staff members being added have the correct domain address (@rtech.co.uk) (NFR2.c, NFR2.d). This proved challenging through three different iterations, as I incorrectly implemented the email validation logic. Initially, I assigned a boolean value to a variable instead of the actual email address, resulting in the code having no effect on the data.

In another attempt, I applied the validation function directly at the time of parameter binding, which unexpectedly resulted in blank email fields in the database. Finally, I implemented a proper comparison using an If statement that successfully validates email addresses before database insertion.

Booking System Challenges

The booking system development (FR3.a, FR3.b) required a review of my database structure, revealing the need to add a products header in the bookings table between the date and type_of_booking fields. This database modification necessitated systematic updates to all related functions to maintain compatibility.

The booking system proved particularly challenging, requiring PHP implementations I had limited experience with. After multiple unsuccessful attempts at adapting existing code, I decided to start from scratch. Following a break from development over the weekend, I returned with fresh perspective and was able to better understand my predefined functions from Version 1. I discovered I had created unnecessarily complex logic and needed to simplify my approach.

During testing, I encountered an error related to "dead" variables that existed because of the booking table refactoring. I rewrote the function to correctly reference only the bookings table and properly assign variables to database fields. Additionally, I found that the user_id wasn't being correctly passed to the function, causing failed data commits due to null values. This was resolved by redesigning the login script to properly store both the user_ID and session variable.

With these issues addressed, I successfully implemented a booking system that leverages multiple functions to verify staff members, validate appointment times within working week constraints, and commit valid bookings to the database (FR3.a). The system now correctly identifies incompatible staff-appointment type combinations, preventing invalid data entry, which contributes to the overall system security (NFR2.d) by ensuring data integrity.

Appointment Scheduling System

A significant achievement in Version 2 is the implementation of smart appointment scheduling with time-conflict prevention (FR3.a, FR3.j). The system now validates if requested appointment times fall within working hours (8 AM to 4 PM, Monday to Friday) using the in_working_week() function. It also prevents booking conflicts by ensuring appointments are at least 90 minutes apart using the booking_time_check() function.

The commit_booking() function successfully handles the insertion of new bookings into the database, converting datetime inputs appropriately and linking them to user accounts. This allows for personalizing the user experience by associating appointments with specific user accounts (FR1.a).

The appointment system's validation logic contributes to both data protection (NFR2.d) by preventing invalid entries and system usability (NFR3.e) by providing clear feedback about appointment availability.

Future Enhancements

I plan to explore using JavaScript to create a dynamic installation field based on the type of staff member chosen (NFR3.e). However, this will be implemented in Version 3 as it's primarily a quality-of-life feature rather than core functionality. I also considered adding a carbon calculator feature but postponed it due to concerns about regulatory compliance.

Conclusion

Version 2 successfully transformed the secure login system from Version 1 into a functioning booking application with staff management capabilities (FR3.a, FR4.a). Despite numerous challenges with database design modifications and complex PHP implementations, the core functionality is now operational. The booking system now effectively prevents time conflicts and validates staff availability, meeting key functional requirements while maintaining data security (NFR2.c, NFR2.d).

Some features have been deferred to Version 3 to maintain the development timeline, but the foundation for these enhancements is now firmly in place. Performance considerations (NFR1.a, NFR1.b) will be further addressed in Version 3 as we optimize database queries and enhance the user interface.

Version 3 During Dev Writeup

CSS and Styling

I have chosen to start with my styling in version 3, I am aware that I still have processing code to do regarding the carbon footprint calculator BUT I also am aware that the calculator will be a very difficult and long-winded development task. Therefore, my logic is that if I complete my styling prior to the calculator then I can spend the remaining time of the development period solely focused on the calculator without concern for the styling of the page to meet WCAG and W3C.

Attempting to stick as close as possible to my design pack, I have encountered a few issues surrounding layouts not being 100% possible. I have accordingly adjusted them to maintain the feel of the Rolsa Technologies Website while also being feasible to

create visually. The look and feel are consistent throughout the entire system with a functional “nav bar”, styled fields etc. There isn’t much to say about CSS since it’s only a styling language and not anything related to processing code like PHP.

Carbon Calculator

Initially looking at this page, I am unsure about regulations surrounding this type of product. I am going to take some time to review co2 calculators and investigate the potential existence of a language compliant library to assist in the development.

After a lot of browsing and reading different regulations such as DEFRA and its EU counterpart I have understood that the calculations required for a 100% accurate calculation according to the aforementioned legislations is not a possibility in its entirety within my development environment. In order to still return some form of product I have chosen to omit some of the fields / values usually requested when a full calculation is done and focus on Electricity Consumption , Gas Consumption (both measured in their recognised unit of (KWH/Y), Water Consumption (L/Y), Transportation using motor vehicles (car, bus, train etc) for which I took the average multiplier used in the real calculation where each of those is a separate heading and calculated the mean average of the three to use as my multiplier in this simpler solution.

My decision to simplify the carbon footprint calculator comes from constraints both within the development environment and lack of proper understanding of how to calculate some of the required variables.

I have included in the calculator page a “Disclaimer” section that highlight that this calculator does not make use of every factor of determining carbon footprint and instead focusing on data that the everyday user can gain easy access to using Rolsa Technologies smart home management products.

The system successfully calculates a carbon tonnage total based on the factors entered and provides the user with insight into their carbon footprint compared to the average household I.E. lower, average or higher. Alongside this it also gives generic information on how to lower your carbon footprint in future.

Static Pages

I have created static versions of:

The contact us page, while currently just a static mock up it is entirely possible to transform this page into a fully functional contact page.

Rolsa Technologies

Dashboard

Book Appointment

Carbon Calculator

Logout

Contact Us

We'd love to hear from you! If you have any questions about our renewable energy solutions or would like to schedule a consultation, please fill out the form below or use our direct contact information.

Your Name

Enter your name

Email Address

Enter your email

Subject

What's this regarding?

Message

Type your message here

Send Message

Our Information

Address:

123 Green Energy Way
Sustainable City, SC1 2RE
United Kingdom

Phone:

+44 (0)123 456 7890

Email:

info@rolsatech.co.uk

Business Hours:

Monday-Friday: 9am-5pm
Saturday: 10am-2pm
Sunday: Closed

The dashboard page has also been created statically presenting the options to rearrange, leave a note or cancel an appointment. I have created this page statically as there is no confirmation that they want the customers to be able to rearrange or cancel the bookings only that they should be able to schedule them. I have taken the choice to not build these pages to perform any processing. This is in case the client decides they want the booking and rearranging to be done through the phone or via email. Whichever method the client decides it is easy to either delete the page or make it entirely processing capable through PHP.

Rolsa Technologies Dashboard, [User]

Book Appointment

Carbon Calculator

Contact Us

Logout

Date: 15/03/2025

Type: Consultation

Notes: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc id ullamcorper lectus. Vivamus.

Rearrange

Add Notes

Cancel

Date: 19/04/2025

Type: Installation

Notes: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc id ullamcorper lectus. Vivamus.

Rearrange

Add Notes

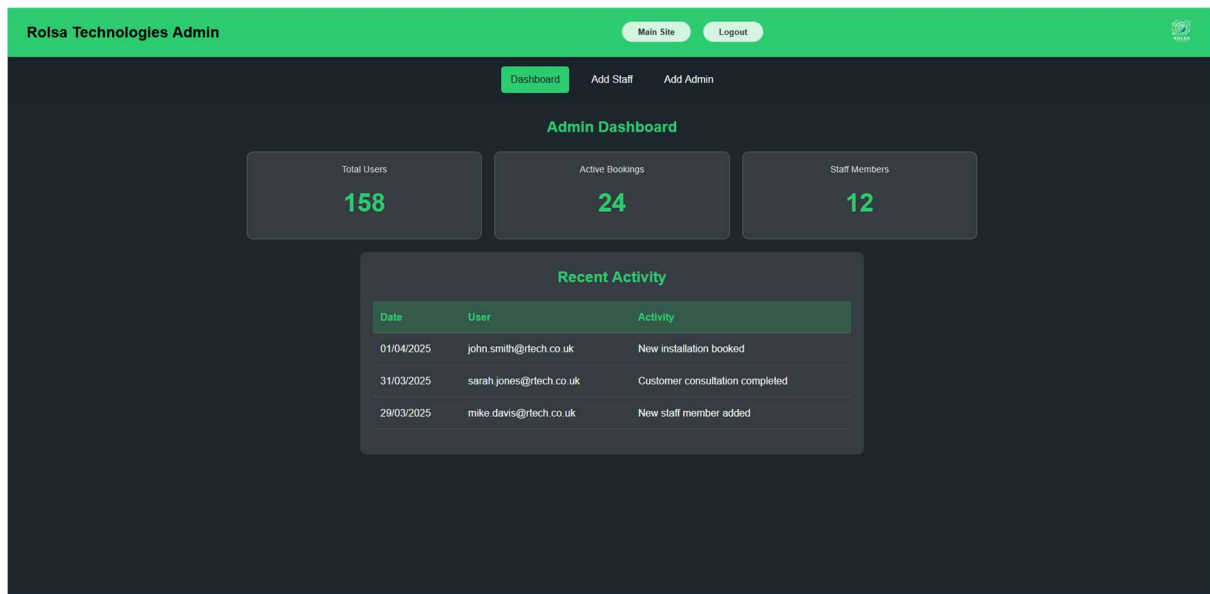
Cancel

Your Usage Rates this Year

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc id ullamcorper lectus. Vivamus consequat mauris sed diam porta pellentesque. Aenean eu urna laoreet, tempor risus ac, euismod lectus. Maecenas fermentum feugiat dolor, non finibus augue efficitur ut. Proin congue sem orci. Donec porttitor metus et elit lobortis, in eleifend massa mattis. Maecenas scelerisque fringilla massa ut viverra. Fusce ac mattis elit, nec congue erat. Nullam sit amet sollicitudin urna. Maecenas tristique faucibus lectus id aliquet. Aliquam rhoncus at justo sit amet luctus. Quisque imperdiet eleifend risus. Nulla eget sollicitudin velit. Suspendisse ultrices metus nibh, ut pulvinar purus dapibus ut.

Jack Witney LL-000018212

Another page I have built statically is the admin dashboard. Again, this page is something that the user may only want available to the super user type and not every type of admin. To combat this, I have created the static page with fabricated data on it. That way there is no sensitive data provided to people who are not privy to it. This is an easy conversion to dynamic data or a revamp if the client wants different metrics portraying.



Version 3 Retrospective

Version 3 Outline:

Version 3 is where everything gets pretty! I'll focus on attractive CSS styling, responsive design, and creating a cohesive look across the platform. There won't be much new PHP code, mainly static information pages and design refinements. This final update transforms the functional product into something that looks professional and most importantly contains all the company colouring and branding as close to the design pack images as possible.

Developer Thoughts

After successfully wrapping up version 3 I can say that I have achieved everything planned for version 3... and a little more. As earlier mentioned in my version 2 retrospective, I delayed the development of my carbon calculator into version 3 to ensure I could wrap up version 2 in good time as it still had a large amount of new functionality that still made for a great writeup and retro.

Jack Witney LL-000018212

While version 3 has not added much in the way of functionality, only adding the calculator it has been leaps and bounds in the way of styling and legislative compliance. Ensuring to comply with WCAG and W3C rules and other sub-legislations.

The styling I have scripted is consistent with my designs for the most part. However, the places in which it differs it differs because in some cases the design was not fully achievable and in others it did not allow for easy and intuitive understanding of what to do on the website. (NFR 3.A)

Evaluation of Built Solution

- review what targets were hit, why some might not have been (FR/NFR)

Functional Requirements

5. 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 Rolsa Technologies market.

6. 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.

7. 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.

8. 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

4. 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.

5. 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.

6. 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

Targets (FR/NFR)

Overall, I have met the FR and NFR to an acceptable level. I have confidently met FR1.a and FR1.b to allow the user to register an account and then log into said account. I have not met my outlined FR1.c and FR1.d to update passwords and deleting accounts. These two FR have not been met through choice as the client has not directly asked for the user to be able to do these tasks themselves. Therefore, to avoid creating scripts the client may not need I have omitted them from my final solution, however, it would be quite a quick job to implement these features and the functions relative to updating and deleting data in the database are already created as a part of my security coverage of having a different user account for each SQL action.

I have met FR 2.a in some sense that there is a data position on the dashboard.php file but it is currently static, I have created a static page as I am unsure how the client would want the data providing and visualising and what data they would specifically want to be readily available to a customer. This is something that would need to be reviewed in detail and moved forward to with version 4 development. The other requirements of FR 2 I have been unable to meet through time constraints, database complexity and development environment issues. This boils down to again another conversation with the client about the next steps and what they want to have the customer able to do themselves.

FR3 has been an interesting one to consider. FR3.a has been met, appointments can be booked with the ability to select the type of staff member and the appointment type. Prior to developing the solution, I outlined the want to “set notes on bookings”(FR3.b) and “cancellation and rescheduling”(FR3.c) However, due to no communication between the client and myself prior to development starting I have decided to implement these pages statically as an example of what is entirely possible to build in version 4 after a conversation with the clients to properly outline their wants and needs as this will also require a database overhaul.

FR4 has been almost entirely postponed, the Admin dashboard does exist it just doesn't allow for fetching user data ETC. The page instead has been statically built to show usage statistics for the site. I prioritised the ability to add administrators and staff to the database and allow them to log in correctly. I would like to see a conversation

surrounding the exact needs and wants for the administrator panels and pages as it is possible I have completely misconstrued the clients needs.

In terms of NFR all around I have completely met NFR 1 and NFR 3, the site is efficient in loading times and database querying as well as being entirely user friendly in terms of flow of what the user can do, the dyslexia friendly font choice and an acceptable level of contrast between text colour and background colour. NFR 2.a I was unable to comply with due to development environment concerns, I have instead opted for default PHP password hashing system so that the user's data IS encrypted but not in the most ideal format.

Database Design Issues

The database I had originally designed in phase 1 of the solution creation had 6 tables, after a brief review during the administrative section development I found that my proposed methodology to identifying administrative privilege levels through a separate "admin_type" table linked via an "adm_type_id" key was an impossible manner of doing so with my current understanding of databases and PHP. As a result, I changed my database to reflect this page, dropping the "admin_type" table in favour of changing the "adm_type" field in the "admin" table to be reflective of a text type rather than an integer and directly inserting the adm_type through a dynamic dropdown menu in the admin creation page of my solution.

After this issue I further reviewed my database and realised that I had also left out the entire address storage from my users table. Upon discovering this during my user signup testing I acted swiftly to update the database accordingly and further to that I ensured my user registration function was taking the relevant fields. This did however lead to an issue around my parameter binding and further database insertion.

V4 Plan

The carbon calculator implemented in Version 3 provides a functional but simplified approach to carbon footprint estimation. For Version 4, I would significantly expand it to ensure full legislative compliance with DEFRA and EU regulations. This would include breaking down transportation emissions by vehicle type rather than using a combined factor, incorporating food waste calculations, adding consumption factors for clothing and electronics, and providing more detailed energy usage analysis with seasonal patterns. The calculator would move beyond the current basic inputs to capture the full

spectrum of carbon-generating activities, storing historical data to track progress over time and offering more personalized reduction recommendations based on specific usage patterns.

While the current system allows users to book appointments, only a static mockup exists for appointment management. Version 4 would implement full functionality behind this mockup, allowing users to view all upcoming bookings on their dashboard, add notes to existing appointments, reschedule or cancel appointments with appropriate time restrictions, and receive automated notifications about their bookings. This functionality could either be built directly into the platform or leverage third-party services for notifications, depending on client preferences. The booking system enhancement would complete the user journey by enabling them to manage their entire relationship with Rolsa Technologies through a single interface, rather than requiring separate channels for booking versus managing appointments.