# Summer 24: Task 1 – Design

--- SOME IMPROVEMENTS FOR THIS DOCUMENT

A screenshot of a computer program

AI-generated content may be incorrect.

This is the design portion of the documents for the product that the client () would like to be produced.

This document contains numerous diagrams and visual representations of different algorithms and other information related to the project. Starting from the top, it begins with an overview of the site: a diagram of the sitemap, which shows all the pages which will be accessible. Additionally, there’s a Use Case Diagram.

It then moves on to the visual design of the web application. This includes styling decisions (fonts, colours, spacing, etc) and page mock-ups (wireframes and full-coloured).

After that, the document goes over data requirements, such as the database structure and included tables, as well as regulations we are following as part of data normalisation. Additionally, it shows an ERD (Entity Relationship Diagram) which visually shows the relationship between all the entities in storage. Lastly, it shows example SQL statements that will be used.

The penultimate section is about algorithms.

## Sitemap

(When I refer to “routes” in this section, I mean destinations that users can access).

I have drawn up a hierarchical map of all the routes that the site will have available.

Additionally, I have colour coded some of the routes which show the main routes and sub-sections vs individual routes. For example, the “Visit Us” rectangle is a sub-section – like a folder in that it contains the routes under it.

--- INSERT SITE MAP

## Use Case Diagram (UCD)

The use case diagram (UCD) shows exactly what users will be able to do whilst on the site.

--- INSERT UCD

## Visual/Interface designs

In this section, I will be going over the designs for this project and what I decided what the site should look like and why. A lot of the user interfaces have been influenced by the research I completed as part of Task 1 Activity A. You can reference to those documents to see the exact sites I took inspiration from.

### Styles

I have chosen to go for a specific style for this web application.

--- EXPLAIN CHOSEN STYLE

#### Font

In the initial designs, I have opted for the “” font family.

--- EXPLAIN CHOSEN FONT

Another consideration was font sizes. I decided in the end to have the following styles:

--- FONT SIZE SIZES

Finally, the last consideration to do with the font is the colouring. Generally, I have decided the following colours will be used, primarily:

* White (#FFFFFF)
  + Used whenever is most fitting. Darker background requires white font.
* Black (#000000)
  + Used whenever is most fitting. Lighter background requires black font in important areas such as the navbar.

#### Spacing

Purposefully, throughout all the designs, I make sure to keep spacing and leading consistent. This is to improve readability for our users. The spacing shouldn’t be too small, as this causes the text to become hard to read, but also it shouldn’t be too large as this causes users to lose where they are reading and become lost.

### Page Mock-ups

For all the page mock-ups, I’ve decided to make the default screen size 1920x1080, as this is the most common resolution for desktop devices.

--- EXPLAIN COMMON THEMING AND CHOICES MADE

#### Landing Page

I first constructed a design with bare bones placeholders instead of actual content.

--- INSERT IMAGES

## Data requirements

### Data Dictionary

|  |  |
| --- | --- |
| Term | Definition |
| PK (Primary Key) | The column in the team that should be used to distinguish each row from every other row (a unique identifier). |
| INTEGER | A whole number, positive or negative, without fractions. (1) |
| STRING | A collection of characters, used for text. (“hello”) |
| BOOLEAN | A data type with two possible values (true, false) |
| LONG | A very large whole number, which exceeds the size of an integer. (9223372036854775807) |

### Database Structure

I had planned the design for the database structure. This covers all the data that will need to be stored within the project.

Table columns that have “PK” are marked as the primary key.

All tables must have a primary key as they need to be uniquely identified and linked together in some way. This is why each table has an “Id” column.

#### Users

|  |  |  |
| --- | --- | --- |
|  | Description | Data type |
| Id (PK) | A unique identifier for each user | INTEGER |
| Email | The email address of the account. Will be used for password resets and extra verification. | STRING |
| Username | Chosen by the user per their discretion. Usernames are unique. | STRING |
| Password | Password used to gain access into the account. Will be stored hashed and salted. Doesn’t have to be unique. | STRING |

Users must be stored in the database so their accounts can be saved and be logged into later. Additionally, any user-related data must be attached to a user so that we know who owns what data, etc. This can be further developed and expanded into more complex systems.

Storing an email and username is arguably not required, but I believe that storing the email is an important attribute for each user. Storing an email brings several positives to the system. For example, users can reset their passwords. If a user forgets their password, they can request an email to their address and use a one-time link to reset it.

Also, emails are unique. Storing multiple accounts under the same email would cause many conflicts and problems within the system. Essentially, emails act as an additional unique identifier for each account.

Usernames are a way to display personal information on the users screen without a risk of unintentionally showing private information to others. A warning will be shown to users when they sign up, stating that the username shouldn’t include details they wouldn’t want to be shared.

Lastly, the password. The password is stored so that the user can login again when they want to. It is stored “hashed” and “salted” which means that even if someone looks at the database, the passwords will be unreadable to the human eye.

All these attributes (apart from the ID) are stored as STRINGs. This is because they are a collection of characters and none of them will be just numbers.

--- EXPLAIN EACH DATA TABLE OTHER THAN USERS

### Data Normalisation

Whilst designing and developing these SQL table structures; I have kept in mind the rules of data normalisation.

#### First Normal Form (1NF)

Every table is designed to contain atomic values, and each row can be uniquely identified. As you can see, every table contains an ID field, which automatically increments for each new entry, therefore making it unique. Additionally, they should all contain atomic values only.

#### Second Normal Form (2NF)

Second normal form doesn’t really apply to any of our tables as there’s nothing that needs to be directly defined, like for example a product of some sort.

The only requirement that we do fulfil as part of 2NF is being in line with 1NF.

#### Third Normal Form (3NF)

Again, third normal form doesn’t really relate in our case. We do fulfil being in line with 2NF, however, so we do count as being compliant with 3NF as well.

### Entity Relationship Diagram (ERD)

I have designed an ERD (Entity Relationship Diagram) to show the relationship between the entities that will be used in the product.

--- ERD IMAGE

--- EXPLAIN ERD

### Sample SQL Statements

In preparation for the development of the product, I have devised some sample SQL statements which will be used in the actual end-product for the client.

#### Adding a new user

INSERT INTO Users (Email, Username, Password) VALUES (“test@gmail.com”, “NewUser1”, “TestingPass1!”);

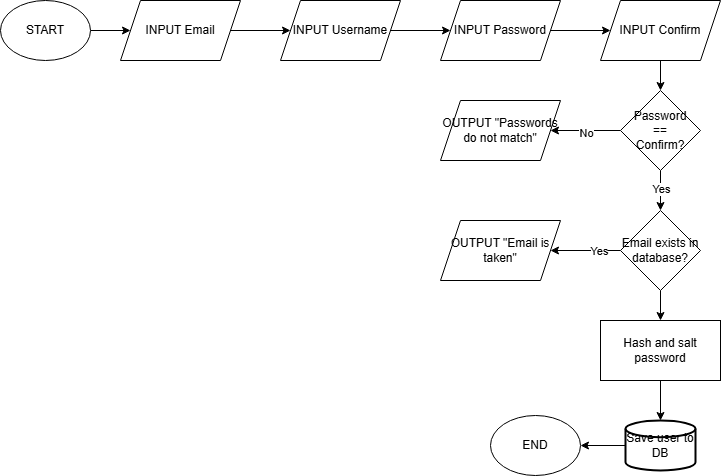
#### Getting an existing user’s password

SELECT Password FROM Users WHERE Id=0;

--- MORE EXAMPLES

## Algorithm designs

### Registration

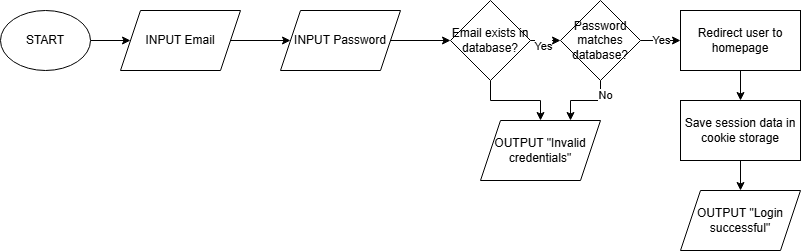


This is the flow chart that represents the process for registering new users. It starts off by asking the user for details. These details are:

* Email
* Username
* Password
* Password confirmation

We ask for password confirmation to ensure that the user knows what their password is once they’ve signed up. The user would have to misspell it the same way twice in a row for a mistake to be made. After this, we validate that the passwords match. If they don’t, they will get an error message. If they do match, we next check if the email entered is already in the database. This would mean that there is already an account that is using that email. An error message is sent if this is the case. If not, the process finishes by hashing and salting the password and saving their details in the database.

### Login



This is the flowchart that represents the steps that are undertaken during the login process. It begins by asking the user to input the following details:

* Email
* Password

After those details are submitted, we check if the email is present in the database. If it isn’t, that means an account hasn’t been made with that email, so it’s not possible for them to sign in. Therefore, we present an error message. If the email does exist, we check the password that is tied to that email for a match with the one that was entered by the user. If they don’t match, once again an error message is presented. If they do match, the user is redirected to the homepage and the session data is stored for the user. They will get a success message telling them the result.

--- PROJECT SPECIFIC FLOW, SUCH AS NEW BOOKING

## Test strategies

I will use a table to display the individual components that will be tested throughout development. I won’t include every single specific test, as there are a lot which fall under each component.

The index column indicates the order in which components should be tested.

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Component | Type of test | Description |
| 1 | Database initialisation | Unit Test | Tests that the server connects to the database correctly and creates all the required tables. |
| 2 | User registration | Unit Test | Tests that the registration form correctly handles the data that is sent from the front-end. |
| 3 | User registration (database) | Integration test | Tests that the data from the front-end is successfully sent to the database and inserted. |
| 4 | User login | Unit Test | Tests that the login form correctly handles the data that is sent from the front-end. |
| 5 | User login (database) | Integration test | Tests that the data from the front-end is checked against what is stored in the database successfully. |
| --- ADD |  |  |  |

Whilst testing these components, a combination of white-box and black-box testing should be used. I have written below the definitions of each.

### Strategy Dictionary

|  |  |
| --- | --- |
| Strategy | Definition |
| White-box testing | Approach that uses the internal logic, structure, and implementation. Testers use the context of the code and algorithms. |
| Black-box testing | Primarily focuses on the functionality without knowledge of the internal workings and structure. Testers only access the inputs and outputs of the system. |

Now I will go over what components should be tested with what strategy:

|  |  |  |
| --- | --- | --- |
| Component | Test strategy | Description |
| Database Init | White-box | Test logic for initializing database and structure. |
| User registration | White-box | Test user input validation and registration logic. |
| User registration (DB) | White-box | Test user data insertion and database interaction. |
| User login | White-box | Test login logic (password validation, session handling). |
| User login (DB) | White-box | |  | | --- | |  |  |  | | --- | | Test login with database (credential check, session). | |
| User login | Black-box | Test end-to-end login functionality without code knowledge. |