# UG Group 64 – Data plan

## What is a data plan?

A data plan is a comprehensive blueprint that outlines the interactions between the client and server in a web application.

It specifies the types of data exchanged, the method of storage, and the necessary processing steps.

A well-structured data plan is crucial for the efficient development and operation of a web application, ensuring data integrity, security and seamless functionality.

## Overview

The project involves designing and developing a web application for ‘Hungervault’, a volunteer organisation dedicated to addressing hunger-related issues. The application will facilitate Hungervault in promoting their services, recruiting new members, and keeping existing members informed about updates and events.

# Events

1. User Sign Up and Log In: Enable users to sign up and log in
2. Manage User Information: Allow users to manage their personal information
3. Join Hungervault: Facilitate users joining and interacting with volunteer organisations
4. View Updates and Events: Provide a platform for Hungervault managers to manage members and events
5. Post Updates and Manage Events (Organisation Managers): Integrate email notifications, social media links, or mobile app functionality for enhanced user engagement / experience

## Event 1 – User Sign Up and Log In

### Client-server interaction – Data sent (client)

* Sign Up: The client sends user details to the server. These include the username, email and password of the client.

{

"username": "string", 🡪 userID (String): Unique identifier for the user

"email": "string",

"password": "string"

}

* Log In: The client sends login credentials to the server. These credentials include the username OR email and password of the client.

{

"username/email": "string",

"password": "string"

}

### Client-server interaction – Data storage (server)

* A users table will be created and used as data storage to manage users that sign up

Users table

CREATE TABLE Users (

user\_id INT PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

* Processing Data
  + Sign Up: Validate inputs, hash the password before storing
  + Log In: Verify credentials, generate and return a session token

## Event 2 – Manage User Information

### Client-server interaction – Data sent (client)

* Update Information: The client sends updated user information to the server

{

"username": "string",

"email": "string",

"password": "string" (optional)

}

* Get Information: The server sends the current user information to the client

{

"user\_id": "int"

}

### Client-server interaction – Data storage (server)

* The users table as described above in event 1
* Processing Data
  + Update information: Validate and update the user record, hash the new password if provided
  + Get information: Retrieve and send user data to the client

## Event 3 – View Update and Events

### Client-server interaction – Data sent (client)

* Request Updates / Events: The client requests updates and events for organisations they are a member of

{

"user\_id": "int"

}

### Client-server interaction – Data storage (server)

* Updates / Events Table: The server sends the requested updates and events data

Updates Table

CREATE TABLE Updates (

update\_id INT PRIMARY KEY AUTO\_INCREMENT,

organization\_id INT,

content TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

visibility ENUM('public', 'private'),

FOREIGN KEY (organization\_id) REFERENCES Organizations(organization\_id)

);

Events Table

CREATE TABLE Events (

event\_id INT PRIMARY KEY AUTO\_INCREMENT,

organization\_id INT,

event\_name VARCHAR(100),

event\_date DATE,

event\_description TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (organization\_id) REFERENCES Organizations(organization\_id)

);

* Processing Data
  + Fetch Updates / Events: Query updates and events to the user and return them

## Event 4 – Post Updates and Manage Events (Organisation Managers)

### Client-server interaction – Data sent (client)

* Post Update: Managers send new updates or event details to the server

{

"organization\_id": "int",

"content": "string",

"visibility": "enum('public', 'private')"

}

* Create / Update Event: Managers request to view or update existing events

{

"organization\_id": "int",

"event\_name": "string",

"event\_date": "date",

"event\_description": "string"

}

### Client-server interaction – Data storage (server)

* Updates / Events Table as described above in event 4
* Processing Data
  + Post Update: Validate and insert a new record in ‘Updates’ table
  + Create / Update Event: Validate inputs and insert / update a record in the ‘Events’ table

## Event 5 – Event RSVP

### Client-server interaction – Data sent (client)

* Event RSVP: Users can RSVP to indicate attendance to an event.

{

"user\_id": "int",

"event\_id": "int",

"response": "string" (e.g., 'attending', 'not attending', 'maybe')

}

### Client-server interaction – Data Storage(server)

* RSVP table: Server stores the RSVP information submitted by users for events

CREATE TABLE RSVPs (

rsvp\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT,

event\_id INT,

response ENUM ('attending', 'not\_attending', 'maybe'),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id),

FOREIGN KEY (event\_id) REFERENCES Events(event\_id)

);

## Event 6 – Branch Management (System Admins)

### Client-server interaction – Data sent (client)

* Create Branch: Admins send requests to create a new branch.

{

"organization\_id": "int",

"branch\_name": "string",

"location": "string"

}

* Edit Branch: Admins send requests to edit branch details.

{

"branch\_id": "int",

"branch\_name": "string",

"location": "string"

}

* Deactivate Branch: Admins send requests to deactivate branch.

{

"branch\_id": "int",

}

### Client-server interaction – Data storage (server)

* Branches Table: The server stores information about HungerVault branches.

CREATE TABLE Branches (

branch\_id INT PRIMARY KEY AUTO\_INCREMENT,

organization\_id INT,

branch\_name VARCHAR(100),

location VARCHAR(255),

is\_active BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (organization\_id) REFERENCES Organizations(organization\_id)

);

## Event 7 – Email Notifications

### Client-server interaction – Data sent (client)

* Subscribe to Email Notifications: Users send a request to subscribe to email notifications for updates and events.

{

"user\_id": "int",

"email\_address": "string",

"notification\_types": ["string"] (e.g., ["updates", "events"])

}

* Unsubscribe from Email Notifications: Users send a request to unsubscribe from email notifications for updates and events.

{

"user\_id": "int",

"email\_address": "string"

}

### Client-server interaction – Data storage (server)

* Email Notifications Table: The server stores email notification information.

CREATE TABLE EmailNotifications (

email\_notification\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT,

email\_address VARCHAR(255),

notification\_type ENUM('updates', 'events'),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

## Event 8 – Link to Social Media

### Client-server interaction – Data sent (client)

* Link Social Media: Managers link HungerVault to social media

{

"organization\_id": "int",

"social\_media\_account": "string",

"auth\_token": "string"

}

* Post Update to Social Media: Updates are cross-posted to social media

{

"update\_content": "string",

"social\_media\_account": "string",

"auth\_token": "string"

}

### Client-server interaction – Data storage (server)

* Server stores the links details between various ‘Social Media’ websites and HungerVault – of which are submitted by managers

SocialMediaLinks Table:

CREATE TABLE SocialMediaLinks (

link\_id INT PRIMARY KEY AUTO\_INCREMENT,

organization\_id INT,

social\_media\_account VARCHAR(50),

auth\_token VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (organization\_id) REFERENCES Organizations(organization\_id)

);

* Processing Data
  + Store social media link details, use APIs to post updates to social media

## Event 9 – Donation Management

### Client-server interaction – Data sent (client)

* Make Donation: Users can make donations

{

"user\_id": "int",

"amount": "decimal",

"payment\_method": "string",

"transaction\_id": "string"

}

* Track Donations: Managers can track and manage donations

{

"manager\_id": "int"

}

### Client-server interaction – Data storage (server)

* Donations initiated by users can be tracked using the donations table below as a form of data storage.

Donations Table:

CREATE TABLE Donations (

donation\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT,

amount DECIMAL(10, 2),

payment\_method VARCHAR(50),

transaction\_id VARCHAR(50),

donated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

* Processing
  + Make Donation: Validate the donation details, process payment via a payment gateway, and insert a record in the ‘Donations’ table
  + Track Donation: Query the ‘Donations’ table and return donation records for reporting and tracking purposes

# Security and Privacy Considerations

## Data Minimisation

Hungervault collects only the data necessary for the intended analysis to ensure that user privacy is respected and data storage is optimised

## Encryption

Passwords are stored using strong, one-way hashing algorithms (such as ‘bcrypt’). This ensures that even if the database is compromised, the actual passwords are not accessible.

When a user creates an account or updates their password, it is hashed on the client side before being sent to the server. On the server, the password is hashed again before being stored in the database.

## Emails

Email addresses are encrypted using strong encryption algorithms such as AES-256 before storage. This encryption is managed by a secure key management system that controls access to the encryption keys – when emails need to be sent or accessed for communication purposes, they are decrypted in a secure environment.

## Address

User addresses, which are considered sensitive personal data, are encrypted both in transit and at rest using strong algorithms such as AES-256. The addresses are encrypted on the client side before being sent to the server and remain encrypted in the database. Decryption only occurs when necessary and in a secure environment.

## Hashing

All sensitive data, including session tokens will be hashed using strong algorithms to ensure data integrity and security (such as SHA-256). Hashing is employed both on the client side (before data transmission) and on the server side (for data storage and processing). This ensures that even if data is intercepted, it cannot be easily tampered with from malicious third parties.

## Session Tokens

Session tokens are used to authenticate user sessions securely. These tokens are generated using a combination of user-specific data and random elements – making them unique and almost impossible to predict.

Tokens are encrypted and hashed before transmission between the client and server. They are stored securely in server-side session storage, and are validated against the stored hash each time the user makes a request. Expired or invalid tokens will be rejected to prevent authorised access and data breaches.

## Secure Data Transmission

All data transmitted between the client and server will be encrypted with security services such as TLS (Transport Layer Security). This protects data from being intercepted or tampered with during transmission. TLS ensures data integrity and confidentiality are maintained.

## Anonymisation

When possible, data is anonymised to protect user privacy. This involves removing personally identifiable information (PII) such that users cannot be directly identified. Anonymised data is used for analytics and reporting to ensure that insights can be gained without compromising individual privacy.

# Conclusion

This data tracking plan for HungerVault provides a comprehensive framework for collecting and analysing user interaction data while prioritising security and privacy. By systematically tracking key user events, adhering to strict naming conventions, and implementing robust security measures, HungerVault can leverage these insights to enhance user experience.

This structured approach to data tracking and security will support HungerVaults mission to combat hunger-related issues effectively and efficiently.