CSCI P445

Software Architecture Specification

Cheyenne Pierpont

Bailey Forbes

Brandon Maurer.

SAS

1 Proposed Software Architecture

1.1 Overview

The proposed system for Cave Country Canoes is a web-based emergency management platform. It will enable real-time emergency distress tracking and customer feedback management. The system will be built using Python's Flask framework for the backend and MySQL for persistent data management. It will be deployed on Heroku, providing scalability and cloud-based access for both staff and customers.

1.2 Subsystem Decomposition

- Emergency Management Module: Responsible for logging emergency incidents and letting owners know where customers are on the river.
- Customer Feedback Module: Allows customers to provide feedback on trips and the app, stored and analyzed for service improvement.

1.3 Hardware/Software Mapping

- Server: Flask-based backend hosted on Heroku, managing all system logic and database interactions.
- Database: MySQL hosted on a cloud-based service (Heroku's database), ensuring accessibility from multiple locations and devices.
- Client-Side Devices: Mobile devices enabling access to the system for customers and employees.

1.4 Persistent Data Management

- Data to be Persisted:
  - Emergency Incident Logs: Incident reports, timestamps, locations, and response actions.
  - Feedback Data: Customer reviews and ratings after each trip.
- Database Design: See Section 2.

1.5 Access Control and Security

- Actors:
  - Customers: Provide feedback and send distress calls.
  - Staff: Can access emergency reports, allocate resources, and manage customer feedback.
  - System Administrators: Full access to all system functionality, including database management and log auditing.
- Authentication and Security:
  - Role-based Access Control (RBAC): Enforce different permissions for customers, staff, and administrators.
  - User Authentication: Implement secure login mechanisms, such as OAuth for staff and customers.
  - Security Provisions: SSL/TLS for data encryption in transit, hashing of user passwords, and logging user activities for security monitoring.

1.5.1 Shred Cipher

While the Shred Cipher has yet to be integrated, it has been developed:

> Shred Cipher is a cryptographic system that uses simulated qubits to enhance security through quantum-inspired techniques. Each subkey in the system is associated with a pair of simulated qubits: a public indicator and a private indicator. The public indicator functions similarly to a traditional public key, allowing the subkey to interact with other subkeys for encryption, decryption, and verification. The private indicator ensures secure, confidential operations within the key protocol, offering both flexibility and security depending on the indicator's state.
>
> The simulated qubits start in a state of superposition using Hadamard gates, allowing them to represent both binary values simultaneously. This state is then modified with phase gates, introducing cryptographically secure randomness. When these qubits are entangled, any measurement of one qubit directly affects its paired counterpart. Shred Cipher uses this principle to generate secure outputs that are essential for key assembly, ensuring that only the correct combination of subkeys can fully assemble the cryptographic key.
>
> The key assembly process relies on interactions between the simulated qubits of multiple subkeys through a custom Diffie-Hellman technique. Public indicators interact securely, and their measurements contribute to the formation of the master key. The process includes neighbor checks, where public indicators engage with their neighboring subkeys in a circular pattern, and self-assessments, where public indicators verify their state with their private counterpart. These operations generate partial secrets, which combine through a key derivation function (KDF) to form the master key.
>
> Once a key operation is complete, Shred Cipher disassembles the key, clearing sensitive data from memory and resetting the qubits. This ensures that any assembled key is only temporarily accessible and must be reconstructed with the proper subkeys for future use. This secure assembly and disassembly mechanism supports distributed key management and ensures secure, multi-party cryptographic operations within a network environment.

2 Database Structure and Relationships

The `ccc_emergency_map` database is designed to manage emergency responses, customer interactions, trip planning, resource allocation, and activity logging. It organizes data to handle emergencies, track trips, and maintain detailed logs. Below are the details of each table and how they interact with others through primary and foreign key relationships.

2.1 Permissions

Field	Type	Details
PermissionID	TINYINT	Primary Key, Unique, Not Null
PermissionName	VARCHAR(100)	Unique, Not Null

Relationships:

- **One-to-Many**: `Account.PermissionTier` references `Permissions.PermissionID`.

Description: Defines access levels within the system, integrating with the `Account` table to control permissions.

2.2 Account

Field	Type	Details
AccountID	INT	Primary Key, Auto Increment
Username	VARCHAR(100)	Unique, Not Null
PasswordHash	VARBINARY(255)	Not Null
Email	VARCHAR(100)	Unique, Not Null
PhoneNumber	VARCHAR(15)	Optional
Address	TEXT	Optional
RegistrationDate	TIMESTAMP	Defaults to current timestamp
FirstName	VARCHAR(100)	Not Null
LastName	VARCHAR(100)	Not Null
AccountType	ENUM	'Customer' or 'Employee', Not Null
PermissionTier	TINYINT	Foreign Key to Permissions table

Relationships:

- **One-to-One**: Linked to the `Customer` and `Employee` tables.
- **One-to-Many**: Links to `Permissions`.

Description: Stores user credentials and information, forming the base for both customer and employee data.

2.3 Customer

Field	Type	Details
CustomerID	INT	Primary Key, Foreign Key to Account table
EmergencyContactName	VARCHAR(100)	Optional
EmergencyContactPhone	VARCHAR(15)	Optional

Relationships:

- **One-to-One**: Links to `Account`.
- **One-to-Many**: Connects to `CustomerTrip`.

Description: Associates customer-specific data with user accounts, linking to trips and emergency alerts.

2.4 Employee

Field	Type	Details
EmployeeID	INT	Primary Key, Foreign Key to Account table
Position	VARCHAR(100)	Not Null
HireDate	TIMESTAMP	Defaults to current timestamp
Status	VARCHAR(50)	Defaults to 'Available', Not Null
RoleUpdateRequired	TINYINT	Default 0

Relationships:

- **One-to-One**: Links to `Account`.
- **One-to-Many**: Connects to `EmergencyDistressAlerts`.
- **Many-to-Many**: Through `EmergencyResponseTeamMember`.

Description: Holds employee details and integrates with the emergency management system for staff assignments.

2.5 LocationsOfInterest

Field	Type	Details
LocationID	INT	Primary Key, Auto Increment
Name	VARCHAR(100)	Not Null
Description	TEXT	Optional
Latitude	DECIMAL(10,7)	Not Null
Longitude	DECIMAL(10,7)	Not Null
Type	VARCHAR(50)	Optional
Elevation	DECIMAL(8,2)	Optional
TerrainType	VARCHAR(50)	Optional
AccessibilityNotes	TEXT	Optional
LastUpdated	TIMESTAMP	Defaults to current timestamp

Relationships:

- **One-to-Many**: Referenced by `Route` and `EmergencyResource`.

Description: Stores geographical data, used in trip planning and resource allocation. Acts as an object construction table which can be referred to other tables. Some of these "objects" can be houses, rocks, lights, etc. which people should see and pay attention to on a voyage.

2.6 Route

Field	Type	Details
RouteID	INT	Primary Key, Auto Increment
StartLocationID	INT	Foreign Key to LocationsOfInterest
EndLocationID	INT	Foreign Key to LocationsOfInterest
Distance	DECIMAL(5,2)	Not Null
EstimatedTime	DECIMAL(5,2)	Optional
DifficultyLevel	VARCHAR(50)	Optional
AgeRequirement	INT	Optional

Relationships:

- **Many-to-One**: Uses `LocationsOfInterest`.
- **One-to-Many**: Links to `Trip`.

Description: Defines paths and connects trips to locations. Acts as another construction table to create a general case for what the Cave Country Canoes can offer to the public.

2.7 TripType

Field	Type	Details
TripTypeID	INT	Primary Key, Auto Increment
TypeName	VARCHAR(100)	Unique, Not Null
Description	TEXT	Optional

Relationships

- **One-to-Many**: The `TripTypeID` is referenced by the `Trip` table, specifically in the `TripTypeID` column. This relationship allows each trip to be categorized under a specific type.

Description The `TripType` table stores categories for trips, such as "Family", "Adventure", or "Educational". It provides a way to differentiate and organize trips based on their type, making it easier to manage and filter trips in the system. Each trip type is linked to multiple trips in the `Trip` table, ensuring that each trip has a designated category.

2.8 Trip

Field	Type	Details
TripID	INT	Primary Key, Auto Increment
RouteID	INT	Foreign Key to Route(RouteID)
TripTypeID	INT	Foreign Key to TripType(TripTypeID)
TripDate	TIMESTAMP	Defaults to current timestamp
TotalDistance	DECIMAL(5,2)	Optional

Relationships:

- **Many-to-One**: Links to `Route` and `TripType`.
- **One-to-Many**: Referenced by `CustomerTrip`.

Description: Tracks individual trip details, linking trips to routes and types for organization. Trips are specific kinds of routes pertaining to the customer base (at the time).

2.9 CustomerTrip

Field	Type	Details
CustomerTripID	INT	Primary Key, Auto Increment
CustomerID	INT	Foreign Key to Customer(CustomerID)
TripID	INT	Foreign Key to Trip(TripID)

Relationships:

- **Many-to-Many**: Connects `Customer` and `Trip`.

Description: Serves as a junction table between customers and trips, enabling many-to-many associations.

2.10 EmergencyDistressAlerts

Field	Type	Details
AlertID	INT	Primary Key, Auto Increment
CustomerTripID	INT	Foreign Key to CustomerTrip(CustomerTripID)
Latitude	DECIMAL(10,7)	Not Null
Longitude	DECIMAL(10,7)	Not Null
Timestamp	TIMESTAMP	Defaults to current timestamp
Status	VARCHAR(50)	Not Null
SeverityLevel	ENUM	Severity levels
AssignedEmployeeID	INT	Foreign Key to Employee(EmployeeID)

Relationships:

- **Many-to-One**: Links to `CustomerTrip` and `Employee`.

Description: Tracks emergency alerts associated with trips and assigns employees for response.

2.11 EmergencyDistressAssignmentQueue

Field	Type	Details
QueueID	INT	Primary Key, Auto Increment
AlertID	INT	Foreign Key to EmergencyDistressAlerts
Processed	TINYINT	Not Null

Relationships:

- **One-to-One**: Each alert is processed once.

Description: Queues alerts for employee assignment, tracking which have been processed. Allows for better management of the situation since priority can change.

2.12 EmergencyResource

Field	Type	Details
ResourceID	INT	Primary Key, Auto Increment
ResourceName	VARCHAR(100)	Not Null
ResourceType	ENUM	Not Null
CurrentLocationID	INT	Foreign Key to LocationsOfInterest

Relationships:

- **Many-to-One**: Links resources to locations.

Description: Tracks available emergency resources, linking them to locations for deployment.

2.13 EmergencyResponseTeam

Field	Type	Details
TeamID	INT	Primary Key, Auto Increment
TeamName	VARCHAR(100)	Not Null
LeadEmployeeID	INT	Foreign Key to Employee(EmployeeID)

Relationships:

- **One-to-Many**: Links to `Employee`.

Description: Organizes emergency response teams, designating leaders.

2.14 EmergencyResponseTeamMember

Field	Type	Details
TeamID	INT	Composite Primary Key, Foreign Key to EmergencyResponseTeam
EmployeeID	INT	Composite Primary Key, Foreign Key to Employee(EmployeeID)
Role	VARCHAR(50)	Optional

Relationships:

- **Many-to-Many**: Connects `Employee` and `EmergencyResponseTeam`.

Description: Associates employees with teams, defining their roles within emergency teams.

2.15 RouteConditionLog

Field	Type	Details
LogID	INT	Primary Key, Auto Increment
RouteID	INT	Foreign Key to Route(RouteID)

Relationships:

- **Many-to-One**: Logs conditions for specific routes.

Description: Monitors and logs conditions for each route.Used to get a better idea of people in distress and help with geofencing in SOS situations.

2.16 EmergencyIncidentAnalysis

Field	Type	Details
AnalysisID	INT	Primary Key, Auto Increment
AlertID	INT	Foreign Key to EmergencyDistressAlerts

Relationships:

- **Many-to-One**: Associates incident analyses with alerts.

Description: Tracks and analyzes emergency incidents linked to alerts. Present to keep a backlog in the event emergency response personnel needs a copy of data (when warranted).

2.17 CustomerFeedback

Field	Type	Details
FeedbackID	INT	Primary Key, Auto Increment
CustomerTripID	INT	Foreign Key to CustomerTrip(CustomerTripID)
Feedback	TEXT	Optional
Rating	INT	Check between 1 and 5
Timestamp	TIMESTAMP	Defaults to current timestamp

Relationships:

- **Many-to-One**: Links feedback to trips.

Description: Stores customer feedback related to their trips.

2.18 ActivityLog

Field	Type	Details
LogID	INT	Primary Key, Auto Increment
ActivityType	VARCHAR(100)	Not Null
TableName	VARCHAR(100)	Not Null
RecordID	INT	Not Null
OperationType	VARCHAR(10)	Not Null
Description	TEXT	Not Null
Username	VARCHAR(100)	Not Null
Timestamp	TIMESTAMP	Defaults to current timestamp

Description: Logs all significant actions within the system, providing an audit trail for monitoring and review. Helps prevent fraud and exploitation of users (such as administrators) who have higher permissions since all changes to the database are logged here. Helps resort data in the event of faulty or otherwise corrupt backups.

Triggers

Triggers automate actions such as logging changes and cleaning up old records:

- Example: The `trg_log_cleanup_AfterInsert` trigger automatically deletes log entries older than 30 days, ensuring the `ActivityLog` remains manageable.
- Other triggers automatically insert or update entries in the `ActivityLog` when changes occur in critical tables like `Account`, `Customer`, and `Employee`.

3 Global Software Control

- Event-Driven Flow: When a customer sends an emergency alert, it triggers the emergency management system, notifying staff and logging the incident.
- Procedural Control: Routine functions like user login, trip logging, and feedback submission follow a linear flow, involving form submission and database interactions.

4 Boundary Conditions

- Startup and Initialization:
  - System Startup: Deployed on Heroku, with database migrations to initialize MySQL.
  - Error Handling: Flask's error-handling framework will manage exceptions, while automated backup scripts ensure data recovery in the case of a server failure.
  - Daily/Weekly/Monthly Tasks: Regular backups of the MySQL database (e.g., daily using mysqldump). Routine server health checks and updates.
  - Server Migration: Every 3-4 years when new hardware is procured, data migration would be handled through bulk data export/import using tools like mysqldump for MySQL and reconfiguring the Heroku deployment.

5 Individual Contributions

5.1 Cheyenne Pierpont

- Helped in research of Front End.
- Maintained communication with the owner of Cave Country Canoes on a bi-weekly basis for updates.
- Acquired the map needed for the app and is in the process of acquiring their logo.

5.2 Bailey Forbes

- Finalized Shred Cipher for the project - yet to be integrated.
- Continued on Full Stack development.
- Added notes on Shred Cipher and Database to RF-3.
- Recoded RF-3 in LaTeX

5.3 Brandon Maurer

- Filled out RF-3 in Docs
- Helped through project research for the architecture of Front End.