

**Alpine Ascents**

**Website Development**

**Documentation**

|  |  |
| --- | --- |
| Authors | C2312L – Group 2 |
| Van Hieu Nguyen – Leader  Quan Luu Minh  Manh Nguyen Duc  Hau Nguyen Van |
| Date | 25/09/2024 |
| Instructor | Hoang Nguyen Duc |
|  |  |

Hanoi, 09/2024

# 

**INDEX**

[**1.** **Problem Research** **3**](#_Toc178040607)

[1.1. General Introduction 3](#_Toc178040608)

[1.2. Current Website System 3](#_Toc178040609)

[1.3. Proposed System Functions 3](#_Toc178040610)

[1.4. Boundaries of the System 6](#_Toc178040611)

[1.5. Hardware and Software Requirements 7](#_Toc178040612)

[**2.** **Requirements Analysis** **8**](#_Toc178040613)

[2.1. Users of the System 8](#_Toc178040614)

[2.2. System Functions 8](#_Toc178040615)

[**3.** **System Designs** **12**](#_Toc178040616)

[3.1. Entity Relationship Diagram 12](#_Toc178040617)

[3.2. Database Design 13](#_Toc178040618)

[3.3. Sitemap 16](#_Toc178040619)

[3.4. System Functions Design 17](#_Toc178040620)

[**4.** **Validation Checklists** **43**](#_Toc178040621)

[**5.** **Task sheet** **45**](#_Toc178040622)

1. **Problem Research**
   1. **General Introduction**

* Website Origin:

The Alpine Ascents Vietnam website was established with the goal of providing unique adventure travel experiences at renowned destinations in Vietnam. Building on the expertise of Alpine Ascents International and designed to harmonize with the natural beauty and rich cultural heritage of Vietnam, Alpine Ascents Vietnam offers premier exploration experiences, from conquering the majestic Fansipan peak and exploring the magnificent cave systems of Phong Nha – Ke Bang, to enjoying the mystical beauty of Ba Na, and embarking on exciting trekking adventures at Nui Chua.

* Need for Developing This Website:

In the context of the growing adventure tourism industry in Vietnam, there is an increasing demand for a professional and comprehensive online platform for searching, booking, and participating in exploration trips. The Alpine Ascents Vietnam website serves not only as a channel to showcase adventure tours but also as a resource for information, knowledge, and community for adventure travel enthusiasts. Establishing this website will facilitate easier customer access, enhance brand reputation, and create a competitive edge in the market.

* 1. **Current Website System**

Currently, there are several companies in Vietnam providing adventure travel and exploration services, such as Oxalis Adventure, Viettrekking, and Adventure Vietnam.

* Strengths of Competitors:
* Intuitive and user-friendly interface.
* Online tour introduction and booking functionality.
* Detailed information about trips and necessary equipment.
* Weaknesses of Competitors:
* Lack of rich, in-depth blog content about real experiences.
* Interface is not fully user-friendly across all devices.
* Limited customer interaction and no online community.
  1. **Proposed System Functions**
     1. **Adventure Tour Programs**

1. *Description:*

* Display a list of available adventure tours, including tour name, price, images, and a brief description of each tour.
* Each tour detail will include comprehensive information such as location, requirements, itinerary, necessary equipment, and important participation notes.
* Customers can view tour information and proceed to book a tour if interested.

1. *Functions:*

* Tour list
* Tour detail page with detailed information and images.
* "Book Tour" button that redirects to the booking form.
  + 1. **Management of Adventure Tours**

1. *Description:*

* Admin can manage the list of adventure tours, including adding, editing, or deleting tours.
* Admin can update pricing, images, and other detailed information about the tours.
* Admin has the authority to cancel tours.

1. *Functions:*

* CRUD (Create, Read, Update, Delete) operations for adventure tours.
* Update pricing.
* Manage images, descriptions, and detailed information about tours.
  + 1. **Gallery**

1. *Description:*

* Customers can view a library of images and videos related to tours, organized by each adventure tour.
* The library is divided into individual albums for each tour, allowing users to view photos and videos related to specific tours.

1. *Functions:*

* Display a list of albums corresponding to each tour.
* Support slideshow viewing mode.
  + 1. **Gallery Management**

1. *Description:*

* Admin has the authority to manage the image and video library for each tour.
* Admin can add, update, or delete images and videos related to the adventure tours.
* Manage directories or albums by each tour.

1. *Functions:*

* Add, delete, and update photos and videos for each tour.
* Create and manage albums (folders) for each tour.
  + 1. **Branches**

1. *Description:*

* Display a list of the company's branches in various countries or regions.
* Each branch will have information about the address, phone number, and location on a map.
* Integrate a Maps API to allow users to view the exact location of each branch.

1. *Functions:*

* Display branch information such as name, address, and phone number.
* View the branch location on the map.
  + 1. **Branch Management**

1. *Description:*

* Admin has the authority to add, delete, and edit branch information.
* Manage detailed information for each branch, including name, address, phone number, and geographical location.

1. *Functions:*

* CRUD (Create, Read, Update, Delete) operations for branch information.
* Update location coordinates (latitude, longitude) for the map.
  + 1. **Information**

1. *Description:*

* Provide general information about mountaineering: history, classification, techniques, etc.
* Information may include an introduction to the company, mission, core values, and contact details.

1. *Functions:*

* Display static information about the company.
* Serve as a resource for researching and accessing supportive information about mountaineering.
  + 1. **User Account Management**

1. *Description:*

* Users can register, log in, and manage their accounts.
* Users can edit personal information such as name, phone number, email, and password.

1. *Functions:*

* Register, log in, and log out of accounts.
* Manage personal information and update details.
  + 1. **Admin Account Management**

1. *Description:*

* Admin has the authority to manage account information for all users in the system.
* Manage permissions between admin accounts and regular users.
* Admin can add or delete user accounts, update information, and reset passwords.

1. *Functions:*

* CRUD operations for user accounts (add, delete, edit).
* User role management (admin or user).
* Manage account status (active or inactive).
  + 1. **Order Placement**

1. *Description:*

* Users can place tour bookings through an order form.
* The form includes customer personal information (name, phone number, email) and tour details (tour name, number of participants, total order value).
* After placing an order, users receive a notification regarding the order result (success or error).

1. *Functions:*

* Order form with fields for personal information and tour details.
* Calculate the total order value based on quantity and tour price.
* Store the order in the database and display the result.
  + 1. **Order Management**

1. *Description:*

* Admin can view a list of all orders placed by customers.
* Admin can view details of each order, including customer information, booked tour, number of participants, and total value.

1. *Functions:*

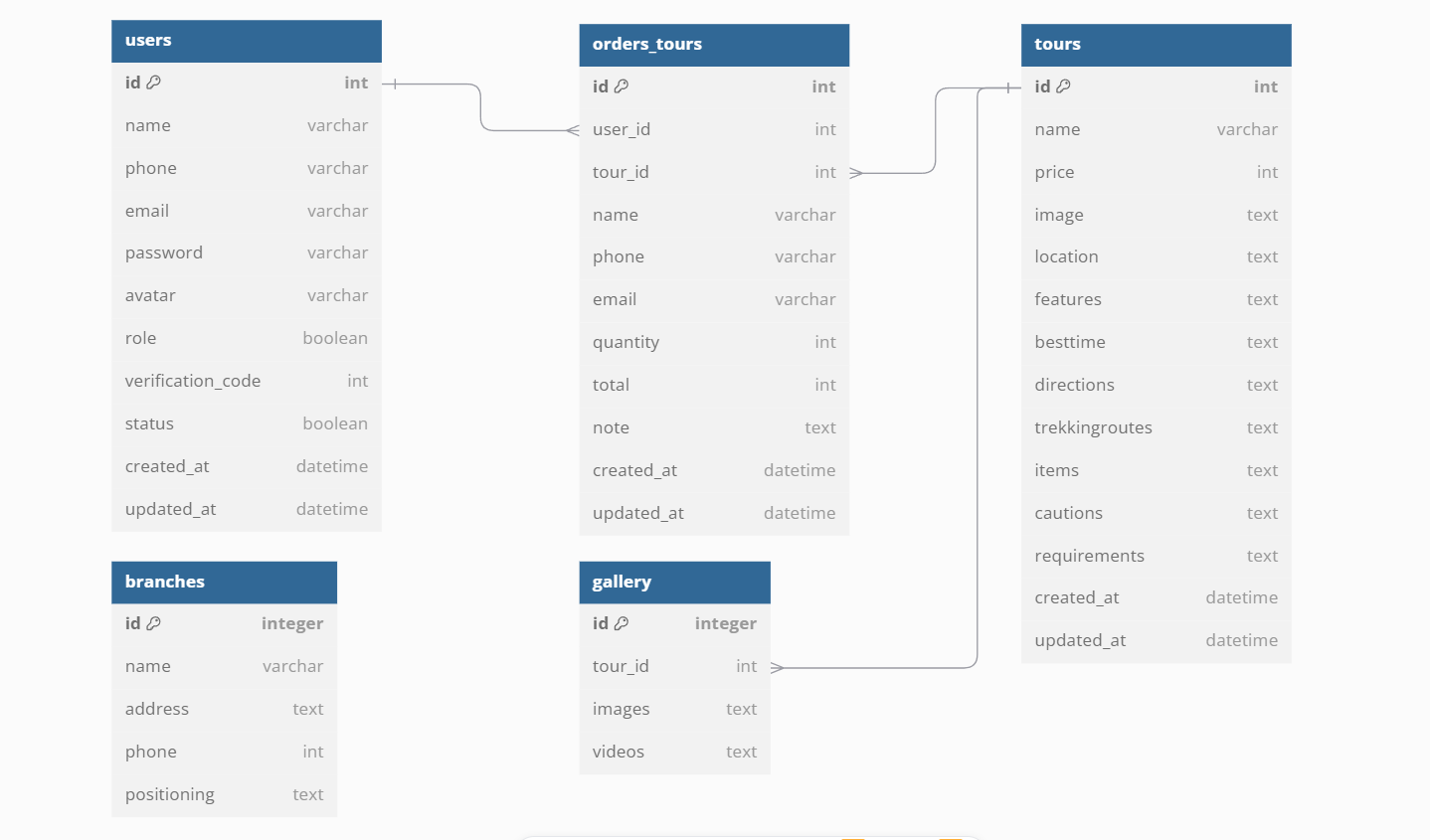
* Display a list of orders with detailed information.
* Update and modify order information.
* View details of each order, including user information and booked tour.
  1. **Boundaries of the System**
* No Support for User Avatar Updates: Currently, the system does not have a feature that allows users to upload or change their avatars.
* No Order History Tracking: The system does not provide functionality for tracking users' order history and bookings.
* No Online Payment Integration: The system currently lacks online payment capabilities.
* Limited User Search Functionality: The system may not support searching for users or administrators based on specific criteria such as name, email, or account status.
* No Shopping Cart Feature: The system does not provide a shopping cart feature to allow users to add multiple tours and process payment for all at once.
* No Order Filtering by Criteria: The system lacks the ability to filter orders by status, order date, or other criteria.
* No Handling of Spam Orders: The system does not have mechanisms to detect and manage spam or invalid orders.
* No Order Status Management: The system does not support updating or changing the status of orders (e.g., pending confirmation, paid, completed).
* No Tour Filtering by Criteria: The system does not allow users to filter tours based on criteria such as location, price, duration, or tour type.
* No Branch Filtering by Geographical Locations: The system does not provide functionality to filter branches by geographical location.
* No Reporting and Analytics Features: The current system does not offer reports or analytics on sales, number of tours sold, or user activity.
* No Detailed Role-Based Permissions: The system does not support detailed permissions for admins or specific roles (e.g., an admin managing only the gallery or orders).
* Not Tested with Over One Million Records: The system has not been performance tested with a large volume of records.
* No Multilingual Support: The system currently only supports a single language, lacking the ability to switch between different languages.
* No Advanced Security Features (2FA, Data Encryption): The system does not integrate advanced security methods such as two-factor authentication (2FA) or encryption of personal data and transactions.
* No Notification System: The system does not support sending notifications via email or push notifications for important events such as successful orders, order cancellations, or status updates.
* No Feedback and Rating System: The system does not provide functionality for users to give feedback or rate the tours they have participated in.
* No Promotion and Discount Management System: The current system does not have management features for promotions or discount codes for users when booking tours.
* No Error Logging and Analysis System: The system lacks the ability to store and analyze error logs for quick detection and resolution of issues.
  1. **Hardware and Software Requirements**
* **Hardware:**
* Intel Core i3/i5 Processor or higher
* 8 GB RAM or above
* Color SVGA
* 500 GB Hard Disk space
* Mouse
* Keyboard
* **Software:**
* Operating System: Windows
* Browsers: Edge, Chrome, Mozilla Firefox, Safari

1. **Requirements Analysis**
   1. **Users of the System**

* **Customers**: Individuals interested in adventure trips, trekking, and mountaineering. They use the website to search for information, book tours, view the photo gallery, check the list of branches worldwide, and research essential information related to mountaineering. They can also manage their personal information when registering and logging into the system.
* **Administrators**: Individuals who manage the website, add and update tour information, process bookings, manage the photo gallery, oversee partner lists, and handle customer information.
  1. **System Functions**

| **Function** | **Input Information** | **Output Information** | **Processing Method** | **Data to be Stored** |
| --- | --- | --- | --- | --- |
| **Tour** | - List of tour programs, including tour name, description, price, itinerary, and other special information. | - List of available tours with detailed descriptions and images.  - Options for online booking. | - Retrieve data from the database about tours. - Display detailed information about each tour.  - Provide options for booking tours. | - Information about tour programs (name, description, price, itinerary).  - Images and videos related to the tours. |
| **Tour Management** | - Tour information: tour name, price, representative image, location, highlights, best time to visit, directions, trekking routes, required items, warnings, participation requirements for customers. | - List of all tours, displaying details of each tour (name, price, location, etc.). - Details of a specific tour when requested by the user. - Result of adding, editing, or deleting a tour (success or error). | - User (admin) submits a request to add, edit, or delete tour information.  - The system checks input data: all required fields (name, price, location, etc.) must be fully filled.  - If the data is valid, the system performs operations with the database (add, edit, delete).  - The system returns results to the user interface. | - Tours table: stores detailed information about each tour.  - Data fields: id, name, price, image, location, features, best time, directions, trekking routes, items, cautions, requirements, created\_at, updated\_at. |
| **Gallery** | - Images, videos, and links (e.g., YouTube) related to each tour. | - Page displaying images and videos for each tour.  - Display images in a slideshow format for each tour. | - When a user accesses the gallery page for a tour, the system retrieves the list of images and videos for that tour from the galleries table. | - Images (stores the image paths).  - Videos (stores links to YouTube videos).  - tours\_id (links to the specific tour). |
| **Gallery Management** | - List of images and videos for each tour.  - Tour ID linked to the gallery.  - URL or YouTube link for videos. | - List of galleries corresponding to each tour.  - List of images and videos belonging to each tour.  - Result of adding or removing images/ videos in the gallery. | - When a user accesses the gallery page for a tour, the system retrieves the list of images and videos for that tour from the galleries table.  - If the admin needs to add or delete images or videos, they will submit a request through the admin interface.  - The system checks the validity of the data (images must be in a valid format, videos must have valid links).  - Update the galleries table and store the new information. | - Galleries table: stores links to images and videos for each tour.  - Data fields: id, tours\_id, images (stored as JSON containing a list of image links), videos (stored as JSON containing a list of video links), created\_at, updated\_at. |
| **Branch** | Branch name, address, phone number, geographical location. | List of branches along with a map displaying the location of each branch. | - Retrieve branch information from the database.  - Use an API to display the locations of branches on the map. | name, address, phone, positioning |
| **Branch Management** | - Branch information: name, address, phone number, positioning information.  - Admin requests to add, edit, or delete branches. | - List of the company's branches worldwide.  - Details of a specific branch.  - Result of adding, editing, or deleting a branch (success or error). | - User (admin) submits requests to add, edit, or delete branch information.  - The system checks required fields (name, address, phone number).  - The system performs operations with the database (add, edit, or delete). - Returns results to the admin. | - Branches table: stores branch information.  - Data fields: id, name, address, phone, positioning, created\_at, updated\_at. |
| **Information** | Information related to history, types, techniques, accommodation, and dangers of mountaineering and trekking. | Display articles and information categorized accordingly. | - Store related information as articles (blogs).  - Classify and display by specific categories. | - Data fields: title, content, category, status. |
| **Account Management** | - Account information upon registration: name, phone number, email, password.   * - When updating information: avatar, account status (status), verification code (verification\_code). * - Login information: email, password. * - Information required for password recovery, if any. | * - User account information (name, email, avatar, role). * - Login status: success or failure. * - Result of registration, account update, or password change. * - Verification or account recovery notifications, if any. | * - When a user registers an account, the system checks if the email and phone number already exist. * - If valid, create a new account and store the information in the database. * - The system will send a verification code via email; if verified correctly, the account status will change to active. * - When a user logs in, the system checks the email, password, and returns the login result. * - When a user updates information, the system verifies access rights and saves changes to the database. * - If a user requests password recovery, the system will send a verification code via email. | - Users table: stores user account information.  - Data fields: id, name, phone, email, password, avatar, role, verification\_code, status, created\_at, updated\_at. |
| **User Management** | * User information such as name, email, phone number, avatar, password, role. | * Display the list of users, details of each user. | * - Create, update, delete user information. * - Manage access rights (admin, regular user). | name, email, phone, password, avatar, role |
| **Order** | * - Customer information when placing an order: name, phone number, email, notes (if any). * - Information about the booked tour: tour ID, quantity, total order value. | * - List of orders placed. * - Detailed information about a specific order (customer name, booked tour, quantity, order value). * - Result of the order placement (success or error). | * - When a user places an order, the system retrieves tour information, quantity, and calculates the total order value. * - The user enters personal information. * - The system confirms the order and stores it in the database. * - The system returns the order placement result. | - Orders table: stores information about the order.  - Data fields: id, user\_id (user ID), tour\_id (tour ID), name (customer name), phone, email, quantity (number of tours), total (total value), note, created\_at, updated\_at. |
| **Order Management** | * Information about the order placer, booked tour, quantity, notes. | * List of orders placed, details of each order. | * - Create an order when a customer books a tour. * - Calculate the total amount, manage order information. | user\_id, tour\_id, quantity, total, note. |

1. **System Designs**
   1. **Entity Relationship Diagram**



* 1. **Database Design**
     1. **List of Tables:**
        1. ***Users Table***
* *Description*: This table stores user information, including verification information and user roles.
* *Fields:*
* id: Primary Key, type BIGINT.
* name: User name, type VARCHAR.
* phone: User's phone number, type VARCHAR, must be unique (UNIQUE).
* email: User's email address, type VARCHAR, must be unique (UNIQUE).
* password: User password, type VARCHAR.
* avatar: Path to the user's avatar, type VARCHAR, can be null (nullable).
* role: User's role, type BOOLEAN, default is false (regular user).
* verification\_code: Account verification code, type INTEGER.
* status: Account verification status, type BOOLEAN, default is false.
* timestamps: Includes created\_at and updated\_at.
* *3NF Normalization Description:*
* Primary Key: id
* The remaining attributes (name, phone, email, password, avatar, role, verification\_code, status) all directly depend on the primary key (id).
* This table is in 3NF as there are no transitive dependencies or attributes that do not depend on the primary key.
  + - 1. ***Tours Table***
* *Description*: Stores detailed information about travel tours, including descriptions, features, requirements, and instructions.
* *Fields*:
* id: Primary Key, type BIGINT.
* name: Tour name, type VARCHAR.
* price: Tour price, type INTEGER.
* image: Representative image of the tour, type TEXT, can be null.
* location: Location of the tour, type TEXT.
* features: Highlights of the tour, type TEXT.
* besttime: Ideal time to take the tour, type TEXT.
* directions: Directions to the tour, type TEXT.
* trekkingroutes: Trekking routes, type TEXT.
* items: Items to prepare, type TEXT, can be null.
* cautions: Special notes when participating in the tour, type TEXT, can be null.
* requirements: Necessary requirements to participate in the tour, type TEXT.
* timestamps: Includes created\_at and updated\_at.
* *3NF Normalization Description:*
* Primary Key: id
* The attributes (name, price, image, location, features, besttime, directions, trekkingroutes, items, cautions, requirements) all directly depend on the primary key (id).
* This table is also in 3NF as all attributes have a direct relationship with the primary key without any transitive dependencies.
  + - 1. ***Orders\_Tours Table***
* *Description*: Stores information about customer orders related to tour bookings.
* *Fields*:
* id: Primary Key, type BIGINT.
* user\_id: Foreign Key referencing the users table, type BIGINT, can be null.
* tour\_id: Foreign Key referencing the tours table, type BIGINT.
* name: Name of the tour booker, type VARCHAR.
* phone: Phone number of the booker, type VARCHAR, default is 000-000-0000.
* email: Email of the tour booker, type VARCHAR, can be null.
* quantity: Number of participants in the tour, type INTEGER.
* total: Total amount for the order, type INTEGER.
* note: Additional notes from the customer, type TEXT, can be null.
* timestamps: Includes created\_at and updated\_at.
* *3NF Normalization Description:*
* Primary Key: id
* Foreign Keys:
* user\_id: References the users table to identify the tour booker.
* tour\_id: References the tours table to identify the tour booked by the user.
* The other attributes (name, phone, email, quantity, total, note) all directly depend on the primary key (id).
* This table has transitive relationships with users and tours through foreign keys but still complies with 3NF.
  + - 1. ***Branches Table***
* *Description*: Stores information about the company's branches worldwide.
* *Fields*:
* id: Primary Key, type BIGINT.
* name: Branch name, type VARCHAR.
* address: Branch address, type TEXT.
* phone: Contact phone number, type VARCHAR.
* positioning: Location information (could be GPS coordinates) of the branch, type TEXT.
* timestamps: Includes created\_at and updated\_at.
* *3NF Normalization Description:*
* Primary Key: id
* The attributes (name, address, phone, positioning) all directly depend on the primary key (id).
* This table is in 3NF.
  + - 1. ***Galleries Table***
* *Description*: Stores information about the library of images and videos related to each tour.
* *Fields*:
* id: Primary Key, type BIGINT.
* tours\_id: Foreign Key referencing the tours table, type BIGINT. If the tour is deleted, all related galleries will be deleted as well (onDelete('cascade')).
* images: Stores a list of images in JSON format, type JSON, can be null.
* videos: Stores a list of videos in JSON format (containing YouTube links), type JSON, can be null.
* timestamps: Includes created\_at and updated\_at.
* *3NF Normalization Description:*
* Primary Key: id
* Foreign Key: tours\_id: References the tours table to identify which tour contains this gallery.
* The other attributes (images, videos) all directly depend on the primary key (id).
* This table has a relationship with the tours table, but because it directly depends on both the primary key and the foreign key, it still complies with 3NF.
  + 1. **Relationship Diagram Between Tables**
* users ↔ orders\_tours: Relationship between users and tour orders. Each order can belong to one user (1:N).
* tours ↔ orders\_tours: Relationship between tours and orders. Each order will belong to one tour (1:N).
* tours ↔ galleries: Each tour can have one or more images and videos in the gallery (1:N).

Below is the ERD (Entity-Relationship Diagram) for this relationship:

Users

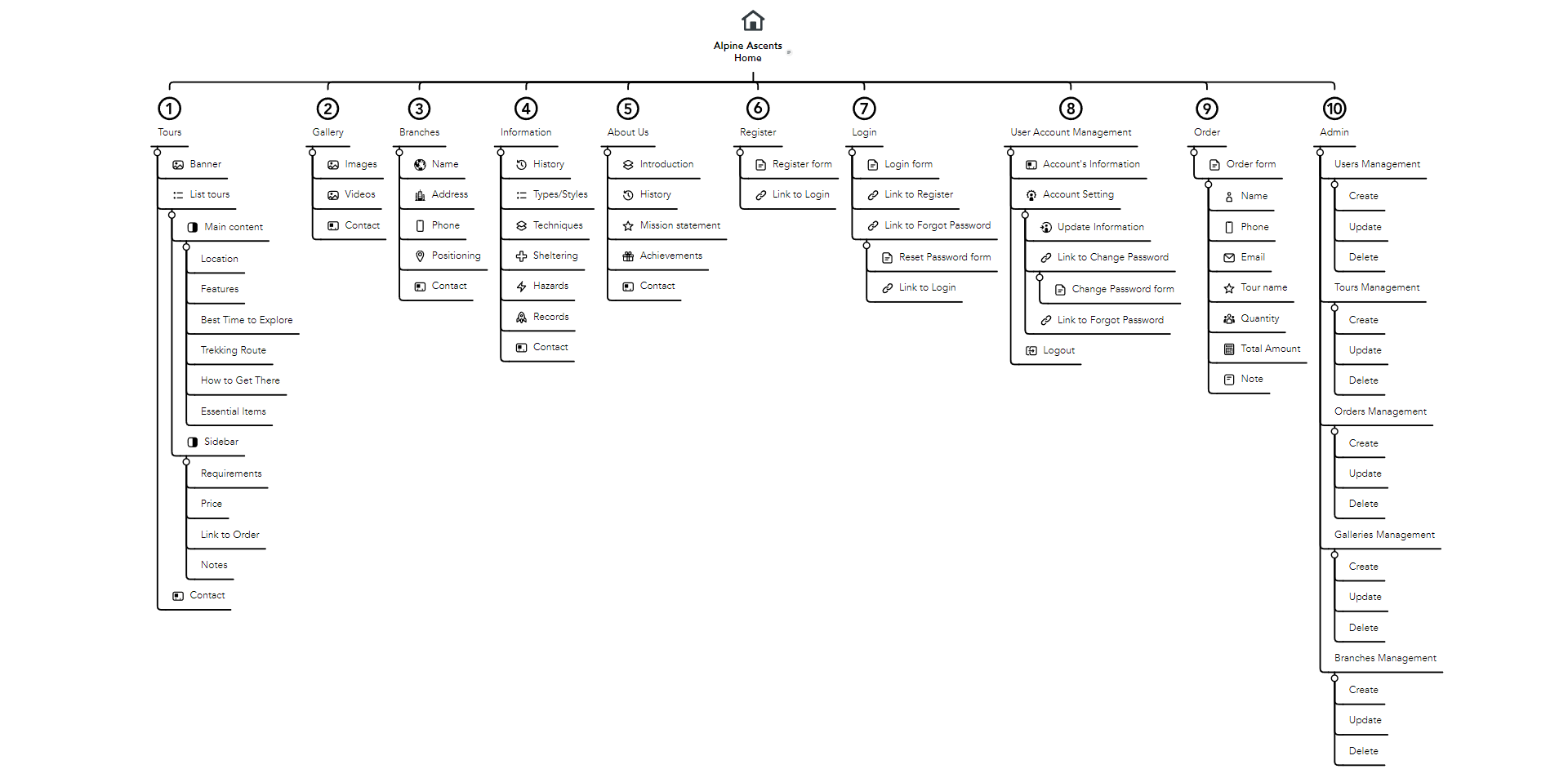
└──(1:N)── Orders\_Tours

Tours

└──(1:N)── Orders\_Tours

└──(1:N)── Galleries

* 1. **Sitemap**



* 1. **System Functions Design**
     1. **Tours Functionality**
        1. ***Detailed Functionality***

1. *Input Information:*

* List of tour programs:
* Tour Name: Name of each travel program (e.g., "Fansipan Tour", "Phong Nha – Ke Bang Tour").
* Tour Description: Detailed information about the program, including destinations, highlights, ideal exploration times, transportation methods, conquest routes, necessary items, etc.
* Tour Price: Cost for each tour (currency can be VND or USD).
* Notes: Information about weather, personal preparation requirements, conditions for participating in the tour, other important notes.
* Illustrative Images: Images representing each destination.

1. *Output Information:*

* List of bookable tours:
* A list of available tours displaying name, price, images, and a brief description of each tour.
* Detailed information when clicking on each tour, including representative images, long descriptions, highlights, ideal exploration times, transportation methods, conquest routes, necessary items, participation conditions, and tour notes.
* Online tour booking options:
* A booking form allowing customers to select a tour, number of people, and special notes (if any).

1. *Processing Method:*

* Retrieving Data from the Database:

Query the database to get a list of available tours along with descriptions, prices, and detailed tour information.

* Displaying Detailed Information About Each Tour:

When a user selects a specific tour, the detail page will show complete information including representative images, long descriptions, highlights, ideal exploration times, transportation methods, conquest routes, necessary items, participation conditions, and notes.

* Providing Booking Options:

Display a “Book Tour” button for customers to access the booking form.

The system checks input information and saves the booking details in the database when the customer submits the order.

1. *Data to be Stored:*

* Information About Tour Programs:

Tours table: stores the name, description, pricing, and other special information about the tour.

* Images Related to Tours:

Galleries table: stores links to images and videos for each tour.

* Booking Data and Tour Status:

Orders\_tours table: stores information about the booker and the number of guests.

* + - 1. ***UI/UX Design:***
  1. *Tour List Page:*
* Display a list of tours with representative images, tour names, prices, and brief descriptions.
* Each tour has a "See More" button to navigate to the detail page.
  1. *Tour Detail Page:*
* Detailed description of the tour program, displaying related images.
* A "Book Tour" button to access the booking form.
  1. *Booking Form:*
* Allows customers to select a tour, choose the number of participants, and add special notes (if any).
* A "Confirm" button to submit the order.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Tours Functionality:*

User Accesses Tour List

Display Tour Details

User selects a tour to view details.

Display detailed information about the tour.

User books the tour.

Display the tour booking form.

Check the input information.

Display booking options for the tour.

Save the tour booking information to the database.

Display notification.

1. *Detailed Steps*

* User Accesses Tour List:

The system retrieves the list of tours from the database and displays it on the page.

* Display Tour Details:

When the user clicks on a tour, the detail page is displayed with complete information about that tour.

* Book Tour:

The user can select a tour and the number of participants. The system checks the input information and saves it into the orders\_tours table.

* Process Tour Booking:

The system displays a success message and saves the order details.

* + - 1. ***Solution Algorithm***
* Retrieve Tour Data from Database:

Use SQL queries to get information from the tours table, including name, description, price, itinerary, and images.

* Save Booking Information:

After checking that the input information is valid, the order will be saved into the orders\_tours table.

* + 1. **Tour Management Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Information about tours:
* Tour Name: Name of the tour program (e.g., "Fansipan Tour").
* Tour Price: Cost for each tour.
* Representative Image: Main illustration of the tour.
* Location: Position of the tour, such as mountains, national parks, historical sites.
* Highlights: Notable features of the tour, e.g., beautiful scenery, special challenges.
* Best Time to Go: The best time of year to participate in the tour (e.g., dry season, autumn).
* Directions: Detailed instructions on how to reach the starting point of the tour.
* Trekking Routes: Detailed description of the trekking route for the tour.
* Items to Bring: List of items that customers need to prepare.
* Cautions: Warnings about difficulty, weather, health.
* Participation Requirements: Mandatory conditions for participating in the tour (e.g., health requirements, age).
* Admin User Requests:
* Request to add, edit, or delete tour information from the management system.

1. *Output Information:*

* List of Tours: Display a list of all tours including names, prices, locations, and basic information.
* Tour Details: Display complete information about a tour when requested by the user.
* Results of Adding, Editing, or Deleting a Tour: Notify the user of the result (success or failure).

1. *Processing Method:*

* Add New Tour:
* Admin accesses the add new tour page.
* Fill in all required information.
* The system checks the data; if valid, the information is saved in the tours table in the database.
* Display a success message for adding the tour.
* Edit Tour:
* Admin selects a tour to edit; the system retrieves current information from the database.
* Admin updates necessary information and submits the request.
* The system checks validity; if valid, updates the information in the tours table.
* Display a success message for editing the tour.
* Delete Tour:
* Admin selects a tour to delete.
* The system checks if the tour has any linked data (e.g., related bookings).
* If there are no links, delete the tour from the tours table.
* Display a success message for deleting the tour.

1. *Data to be Stored:*

Tours Table: Data fields:

* id: Unique ID of the tour.
* name: Tour name.
* price: Tour price.
* image: Representative image.
* location: Location of the tour.
* features: Highlights of the tour.
* besttime: Best time to go.
* directions: Directions to the starting point.
* trekkingroutes: Trekking route description.
* items: Items to bring.
* cautions: Warnings about the tour.
* requirements: Participation requirements.
* created\_at: Creation date.
* updated\_at: Update date.
  + - 1. ***UI/UX Design***

1. *Add New Tour Interface:*

* Form to input fields such as name, price, location, image, highlights, etc.
* "Save Tour" button to add new tour information to the system.

1. *Edit Tour Interface:*

* Display current information of the tour, allowing the admin to update.
* "Update" button to save changes.

1. *Tour List Interface:*

* Display a list of all tours with basic information.
* "Edit" or "Delete" buttons to perform the respective actions.

1. *Tour Detail Interface:*

* Display detailed information about each tour when selected by the user.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Tour Management Functionality:*

Admin accesses the tour management feature.

Retrieve data from the database.

Display the list of tours

Choose an action: Add, Update, Delete tours

Update tour

Add new tour

Delete tour

Retrieve tour data.

Fill information for new tour.

Check relationship and delete.

Update information

Validate data

Delete tour from the database

Update information to database

Save new tour to database

Display notification

1. *Detailed Steps*

* Add New Tour:
* The admin enters tour information into the add new form.
* The system checks the data; if valid, it saves the information into the tours table and displays a success message.
* Edit Tour:
* The admin selects an existing tour; the system loads the current data.
* The admin updates the information, and the system checks its validity before saving the new information.
* Delete Tour:
* The admin selects the tour to delete; the system checks if the tour is linked to any other data.
* If not, the system deletes the tour and displays a success message.
  + - 1. ***Solution Algorithm***

1. *Add New Tour:*

* Step 1: Receive data from the input form.
* Step 2: Check required fields such as name, price, and location.
* Step 3: If valid, save the data into the tours table.

1. *Edit Tour:*

* Step 1: Retrieve the current tour data from the database.
* Step 2: The admin edits the information and submits the update request.
* Step 3: Check the data; if valid, update it in the tours table.

1. *Delete Tour:*

* Step 1: The admin selects the tour to delete.
* Step 2: Check for data links (e.g., whether there are related bookings).
* Step 3: If there are no links, delete the tour from the tours table.
  + 1. **Gallery Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Images: Image files uploaded by the user (admin) or links to stored images.
* Videos: Links to videos on online platforms (e.g., YouTube) or videos uploaded directly.
* Link to Tour: Each image and video must be linked to a specific tour (via the tours\_id field).

1. *Output Information:*

* Gallery Display Page: Displays a list of all images and videos related to each tour in a slideshow or carousel format.

1. *Processing Method:*

Display Gallery:

* When the user accesses the gallery page, the system queries the images and videos linked from the database.
* All images and videos will be displayed in a slideshow or carousel format.
* Videos will be shown as YouTube links, embeddable and playable directly on the page.

1. *Data to be Stored:*

No new data needs to be stored.

* + - 1. ***UI/UX Design***

Gallery Display Interface:

* The detail page of each tour will show the corresponding images and videos.
* Image Slideshow: Allows users to view images in a slideshow format, with navigation between images.
* Video: Videos will be embedded directly from YouTube, with a play button on the page.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Gallery Management Functionality:*

Display gallery

Retrieve gallery’s data from database

Display images and videos

Display slideshow of images

Display embedded YouTube video

1. *Detailed Steps*

Display Gallery:

* Step 1: The user accesses the gallery page.
* Step 2: The system queries the galleries table to retrieve all images and videos linked to the tours\_id of each tour.
* Step 3: Display images in a slideshow format and embed videos from YouTube on the page.
* Step 4: Users can navigate between images and play videos directly on the page.
  + - 1. ***Algorithm for Displaying Gallery:***
* Step 1: Retrieve Tour Information
* Input: Tour ID or slug from the URL.
* Retrieve related information about that tour from the database, including image paths and video links.
* Step 2: Retrieve Images and Videos
* The system reads image files from the directory based on the tour name.
* Videos are fetched from the database or YouTube links.
* Step 3: Display Slideshow
* Create a slideshow from the image files.
* Display a list of videos along with YouTube links.
* Step 4: Handle Display
* Determine the size and layout to ensure images and videos display correctly on the interface.
* Create navigation buttons (left, right) for users to browse through the images in the slideshow.
  + 1. **Gallery Management Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* List of Images and Videos for Each Tour:

Multiple image and video files from the computer or URLs.

* URL or YouTube Links for Videos:

Enter the video URL from YouTube or other platforms.

* Tour ID Linked to the Gallery:

Use the tour ID to identify the corresponding gallery.

1. *Output Information:*

* List of Galleries Corresponding to Each Tour:

Display the tour name and thumbnail of the related image/video.

* List of Images and Videos for Each Tour:

Display details of all images and videos in the gallery.

* Result of Adding or Deleting Images/Videos in the Gallery:

Notification of success or failure in adding/deleting.

1. *Processing Method:*

* Access Gallery Management Page:

When the user accesses the gallery management page for the tour, the system queries data from the galleries table using tours\_id.

* Add/Delete Images/Videos:

Admin submits a request through the form.

* The system checks the validity of the data:

Images: Format (JPEG, PNG, GIF).

Videos: Validate the URL.

* Update galleries Table:

If the data is valid, the system saves the new information into the galleries table.

1. *Data to be Stored:*

Galleries Table: Data fields:

* id: Unique ID of the gallery.
* tours\_id: Foreign key linking to the tours table.
* images: Store the list of image links in JSON format.
* videos: Store the list of video links in JSON format.
* created\_at: Creation date.
* updated\_at: Update date.
  + - 1. ***UI/UX Design***

1. *Gallery List Interface:*

* Display a list of all galleries corresponding to each tour.
* Show detailed information about each tour's images in thumbnail format.
* Show detailed information about videos as YouTube links.
* "Edit" or "Delete" buttons to perform the respective actions.

1. *Add New Gallery Interface:*

* Form to input fields such as:
* Upload images/videos.
* Enter video URL.
* Select linked tour.
* "Save Gallery" button to add information to the system.

1. *Edit Gallery Interface:*

* Display the current list of images/videos with editing options.
* "Update" button to save changes.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Gallery Management Functionality:*

Admin accesses gallery management feature

Retrieve data from database

Display list of gallery

Choose action: Add, Update, Delete gallery

Update gallery

Add new gallery

Delete gallery

Retrieve gallery’s data

Fill new gallery’s information

Update information

Validate data

Delete gallery data from database

Update information to database

Save new gallery data to database

Display notification

1. *Detailed Steps*

* Add New Gallery:
* Admin enters information into the form.
* The system checks the data; if valid, it saves it into the galleries table and displays a success message.
* Edit Gallery:
* Admin selects an existing gallery; the system loads the current data.
* Admin updates the information, and the system checks its validity before saving the new information.
* Delete Gallery:
* Admin selects the gallery to delete.
* The system deletes the gallery and displays a success message.
  + - 1. ***Solution Algorithm***

1. *Add New Gallery:*

* Step 1: Receive data from the input form.
* Step 2: Check required fields (images, videos).
* Step 3: If valid, save the data into the galleries table.

1. *Edit Gallery:*

* Step 1: Retrieve the current gallery data from the database.
* Step 2: Admin edits the information and submits the update request.
* Step 3: Check the data; if valid, update it in the galleries table.

1. *Delete Gallery:*

* Step 1: Admin selects the gallery to delete.
* Step 2: Delete the gallery from the galleries table.
  + 1. **Branches Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Branch Name: The name of each branch (e.g., "US Branch").
* Address: The specific address of each branch.
* Phone Number: Contact phone number for the branch.
* Geographical Location: Latitude and longitude of the branch for display on the map.

1. *Output Information:*

* List of Branches: Display the name, address, and phone number of all branches.
* Map: Show the location of each branch on the map, with markers for each location.

1. *Processing Method:*

* Retrieve Branch Data:
* When the admin or user accesses the branch list page, the system fetches the list from the branches database.
* Information includes branch name, address, phone number, and geographical location (latitude, longitude).
* Display Map:
* The system uses Maps API to display the map.
* Each branch will be marked (marker) on the map with geographical coordinates retrieved from the database.

1. *Data to be Stored:*

No new data needs to be stored.

* + - 1. ***UI/UX Design***
* Branch List Interface:
* Display a list of branches with information such as name, address, and phone number.
* Next to this, there is a map with markers indicating the location of each branch.
* Map:
* Display the map with markers to indicate the locations of the branches.
* Users can zoom in and drag to view detailed locations.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Branches Functionality:*

Display branches

Retrieve branches’ data from database

Display branches’ information

Display map & marker

1. *Detailed Steps*

View Branch List:

* Step 1: The user accesses the branch list page.
* Step 2: The system retrieves the list of branches from the branches table and displays the information on the website.
* Step 3: The locations of the branches are displayed on the map using the Maps API.
  + - 1. ***Solution Algorithm***

Display Branch List and Map:

* Step 1: Retrieve the list of branches from the database.
* Step 2: Display the list of branches along with information such as name, address, and phone number on the page.
* Step 3: Use the Maps API to show the locations of the branches on the map.
  + 1. **Branch Management Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Branch Information:
* Branch Name: The name of the company branch.
* Address: The specific address of the branch.
* Phone Number: Contact number for the branch.
* Positioning Information: Data about geographical coordinates (longitude and latitude).
* Admin Requests:
* Add, edit, or delete branch information.

1. *Output Information:*

* Branch List: Display all company branches worldwide along with address, phone number, and positioning information.
* Add/Edit/Delete Results: Return a message to the admin (success or error).

1. *Processing Method:*

* Add New Branch:
* Admin accesses the add new branch form and enters the required information.
* The system checks the validity of the information (name, address, phone number, positioning).
* If valid, the information is saved into the branches table.
* The system returns a success message for adding the branch.
* Edit Branch:
* Admin selects the branch to edit.
* The current information of the branch is retrieved from the database and displayed on the edit form.
* Admin updates the necessary information and submits the request.
* The system checks the validity of the new information; if valid, it updates the database.
* A success message for editing is displayed.
* Delete Branch:
* Admin selects the branch to delete.
* The system removes the branch from the branches table.
* A success message for deletion is displayed.

1. *Data to be Stored:*

Branches Table: Data fields:

* id: Unique ID of the branch.
* name: Branch name.
* address: Branch address.
* phone: Branch phone number.
* positioning: Geographical coordinate information (contains longitude and latitude).
* created\_at: Creation date.
* updated\_at: Update date.
  + - 1. ***UI/UX Design***
* Branch List Interface:
* Display a list of all branches along with basic information.
* "Edit" or "Delete" buttons for each branch.
* Add New Branch Interface:
* Form containing input fields (name, address, phone number, positioning).
* "Save Branch" button to save information into the system.
* Edit Branch Interface:
* Display the current information of the branch.
* Allow the admin to edit information and click "Update."
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Branch Management Functionality:*

Admin accesses branches management feature

Retrieve data from database

Display list of branches

Choose action: Add, Update, Delete branch

Update branch

Add new branch

Delete branch

Retrieve branch’s data

Fill new branch’s information

Update information

Validate data

Delete branch from database

Update information to database

Save new branch to database

Display notification

1. *Detailed Steps*

* Add New Branch:
* Admin enters information into the form.
* The system checks the data; if valid, it saves it into the branches table and displays a success message.
* Edit Branch:
* Admin selects an existing branch; the system loads the current data.
* Admin updates the information, and the system checks its validity before saving the new information.
* Delete Branch:
* Admin selects the branch to delete.
* The system deletes the branch and displays a success message.
  + - 1. ***Solution Algorithm***
* Add Branch:
* Step 1: Receive data from the input form.
* Step 2: Check required fields (name, address, phone number, positioning).
* Step 3: If valid, save the data into the branches table.
* Edit Branch:
* Step 1: Retrieve the current data from the branches table.
* Step 2: Admin updates the information and submits the request.
* Step 3: Check for validity; if correct, update the data in the table.
* Delete Branch:
* Step 1: Admin selects the branch to delete.
* Step 2: The system removes the branch from the branches table.
  + 1. **Information Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Information about Mountaineering and Trekking:
* History: The development of mountaineering and trekking over time.
* Types: Various types of trekking and mountaineering (exploration, adventure, short treks).
* Techniques: Necessary technical guidelines for mountaineering and trekking.
* Accommodation: Places to stay and rest during the journey.
* Hazards: Risks, weather warnings, health concerns, and terrain issues.
* User Requests (Admin): Add new articles or update information in specific categories (history, types, techniques, etc.).

1. *Output Information:*

* Display Information List: Show a list of articles categorized by (history, types, techniques).
* Article Details: Display the full content of an article when the user requests to view details.
* Category-Specific Display Results: Show specific information by topic.

1. *Processing Method:*

* Classifying and Storing Articles:
* Articles on history, types, and techniques are stored as static HTML. Each article has a clear title, content, and category.
* Articles are organized in a structured folder format.
* Display Articles:

When a user selects a category (e.g., history), the system loads and displays a list of corresponding articles with titles and summaries.

* Manage Articles:

Admin can add, update, or delete content of articles directly in the HTML code.

1. *Data to be Stored:*

HTML Files:

* title: Title of the article (displayed on the homepage or category).
* content: Content of the article (detailed description in each article).
* category: Category of the article (history, types, techniques, accommodation, hazards).
  + - 1. ***UI/UX Design***
* Add and Edit Article Interface:
* Use an HTML editor to add or edit content.
* Organize HTML files by category.
* Category Display Interface:

When clicking on each category, display the content of the corresponding article.

* + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Information Functionality:*

Display categories of information

User chooses a category

Display posts by category

1. *Detailed Steps*

Display Articles:

* Step 1: When the user accesses the information page, the system will load the categories of articles by topic from the HTML files.
* Step 2: The user can select a specific topic (history, techniques, accommodation).
  + - 1. ***Solution Algorithm***
* Step 1: Load the Category Page:

When the user accesses the page, the system displays a list of categories (history, types, techniques).

* Step 2: Display Articles by Category:

Based on the category selected by the user, the system will display the corresponding articles in that category's folder.

* + 1. **User Account Management Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Account Registration Information:
* Name: User's name.
* Phone Number: Contact and account verification.
* Email: Email address for login and verification.
* Password: Encrypted and securely stored.
* Verification Code: Used for verification when the user registers a new account.
* Account Update Information:
* Email: Used to update the user's email information.
* Phone Number: Used to update the user's phone number.
* Password: Used to change the user's current password.
* Verification Code: Used for verification when the user changes email or phone number.
* Login Information:
* Email.
* Password.
* Password Recovery Request (if any):
* Email to send the recovery code.
* Verification Code: Used for verification when the user recovers the password.

1. *Output Information:*

* User Account Information: Name, email, avatar, phone number of the user.

Login Status: Notification of successful or failed login.

* Results of Actions:
* Notification of successful or failed registration.
* Notification of successful or failed account update.
* Notification of successful or failed password change.
* Account Verification Notification: Notice requesting account verification via email.

Account Recovery Notification: Notice of successful or failed password recovery.

1. *Processing Method:*

* Account Registration:
* User fills in registration information.
* The system checks if the email and phone number are already in use.
* If valid, the system saves user information into the users table and sends a verification code via email.
* The user enters the verification code; if correct, the account is activated.
* Login:
* User enters email and password.
* The system checks login information in the users table.
* If valid, the user is logged into the system.
* Account Update:
* User can update information such as name, email, phone number, and password.
* User enters the verification code sent via email and then saves the updated information to the database.
* Password Recovery:
* User requests password recovery.
* The system sends a verification code to the new email.
* User enters the verification code received via email and sets a new password.

1. *Data to be Stored:*

Users Table: Stores user information.

* id: Unique ID of the user.
* name: User's name.
* phone: User's phone number.
* email: Login email.
* password: Password (encrypted).
* avatar: User's avatar.
* role: User's role (admin or user).
* verification\_code: Verification code for registration or password recovery.
* status: Account status (active or unactivated).
* created\_at: Account creation date.
* updated\_at: Account update date.
  + - 1. ***UI/UX Design***

1. *Account Registration Interface:*

* Form includes: Name, phone number, email, password.
* "Register" button to submit information.
* Display error message if email or phone number already exists.

1. *Login Interface:*

* Form to enter email and password.
* "Login" button to check information.
* Display success or failure message.

1. *Account Update Interface:*

* Form to update account information including name, email, phone number, password.
* "Save" button to save updated information.

1. *Password Recovery Interface:*

* Form to enter email to request password recovery.
* After receiving the code, the user will enter the verification code and new password.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of User Account Management Functionality:*

User accesses account management feature

Choose action:Register, Login, Update, Reset password

Login

Register

Reset password

Update

Validate information

Validate information

Email verification

Retrieve data

Confirm login

Save to database

Set new password

Update information

Email verification

Save change to database

Display notification

Email verification

Activate account

1. *Detailed Steps*

* Account Registration:
* Step 1: The user fills in all required registration information.
* Step 2: The system checks if the email and phone number already exist.
* Step 3: If valid, the system saves the information into the users table in the database and sends a verification code via email.
* Step 4: The user enters the verification code; if correct, the account will be activated.
* Login:
* Step 1: The user enters their email and password.
* Step 2: The system checks if the information matches the database.
* Step 3: If correct, the system allows the user to log in.
* Account Update:
* Step 1: The user can update their name, email, phone number, and password.
* Step 2: For phone number updates, the system sends a verification code via email; the user enters the code, and if correct, the system saves the changes to the database.
* Step 3: For email updates, the system sends a verification code to the new email; the user enters the code, and if correct, the system saves the changes to the database.
* Password Recovery:
* Step 1: The user requests password recovery by entering their email.
* Step 2: The system sends a verification code via email, and the user will enter this code before setting a new password.
  + - 1. ***Solution Algorithm***
* Account Registration:
* Step 1: Receive registration information from the form.
* Step 2: Check if the email and phone number already exist.
* Step 3: If valid, save the information into the users table and send a verification code via email.
* Step 4: The user enters the verification code; if correct, activate the account.
* Login:
* Step 1: Receive login information.
* Step 2: Check email and password in the database.
* Step 3: If valid, allow the user to log in.
* Account Update:
* Step 1: Receive update information from the user.
* Step 2: Send a verification code via email.
* Step 3: The user enters the verification code; if correct, save the new information into the users table.
* Password Recovery:
* Step 1: Receive email request for recovery.
* Step 2: Send a verification code via email.
* Step 3: The user enters the verification code and sets a new password.
  + 1. **Admin Account Management Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

User Information:

* Name.
* Email.
* Phone Number.
* Avatar.
* Password (when creating new).
* Role: Admin or Regular User.

1. *Output Information:*

* Display User List:

Name, email, role, avatar, account status (active or not activated).

* User Details:
* Display personal information and permissions.
* Actions: Update, Delete.

1. *Processing Method:*

* Create User:
* Admin enters user information such as name, email, password, phone number, avatar, and role.
* The system checks if the email already exists.
* If valid, the system saves the information into the database and displays the user in the list.
* Update User Information:
* Admin selects a user from the list and updates information such as name, email, phone number, avatar, or role.
* The system checks the admin's access rights before allowing updates.
* Save updated information into the database.
* Delete User:
* Admin selects the user to delete from the list.
* The system checks for any links with other data and deletes the user from the database.
* Manage Access Rights:
* Admin can change the role of a user (from regular user to admin or vice versa).
* The system checks the admin's rights before updating the role.

1. *Data to be Stored:*

Users Table:

* id: Unique ID of the user.
* name: User's name.
* email: User's email.
* phone: User's phone number.
* password: Encrypted password.
* avatar: User's avatar.
* role: User's role (admin or regular user).
* status: Account status (active or not activated).
* created\_at: Account creation date.
* updated\_at: Account update date.
  + - 1. ***UI/UX Design***
* User List Interface:
* Display a list of all users with columns such as name, email, phone number, encrypted password, role, avatar, and account status.
* "Add User" button for the admin to create a new user.
* "Update" or "Delete" buttons next to each user.
* User Detail Interface:
* Display detailed information about the user: Name, email, phone number, role.
* "Update Information" button to change user information, including role.
* Create New User Interface:

Form to create a new user with fields: Name, email, phone number, avatar, role, and password.

* + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Admin Account Management Functionality:*

Admin accesses account management feature

Retrieve data from database

Display list of accounts

Choose action: Add, Update, Delete account

Update account

Add new account

Delete account

Retrieve account’s data

Fill new account’s information

Update information

Validate data

Update information to database

Save new account to database

Display notification

Choose account

Check relationship

Delete account

Delete information from database

1. *Detailed Steps*

* Create New User:
* Step 1: Admin enters the necessary information such as name, email, phone number, role, and avatar into the form.
* Step 2: The system checks if the email already exists.
* Step 3: If valid, the system saves the new user information into the database and displays the updated user list.
* Update User Information:
* Step 1: Admin selects a user from the list to update their information.
* Step 2: After the admin updates, the system checks the validity of the data and saves the changes into the database.
* Delete User:
* Step 1: Admin selects the user to delete.
* Step 2: The system checks for links with other data and requests confirmation before deleting the user from the database.
  + - 1. ***Solution Algorithm***
* Create User:
* Step 1: Receive user information from the form (name, email, phone number, password, avatar, role).
* Step 2: Check if the email exists in the database.
* Step 3: If it does not exist, save the information into the users table and notify that creation was successful.
* Update User Information:
* Step 1: Admin selects the user to update from the list.
* Step 2: Receive the information to update from the admin (name, email, phone number, role).
* Step 3: Check the admin's access rights before saving the new information into the database.
* Delete User:
* Step 1: Admin selects the user to delete.
* Step 2: The system checks for links with other data and requests confirmation for deletion.
* Step 3: After confirmation, delete the user information from the users table.
  + 1. **Order Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Customer Information:
* Customer Name.
* Phone Number.
* Email.
* Notes (if any).
* Tour Booking Information:
* Tour ID.
* Number of tours booked.
* Total order value.

1. *Output Information:*

* Order Details: Customer name, booked tour, quantity, total order value.
* Order Result: Success or error notification.

1. *Processing Method:*

* The user selects the desired tour and quantity.
* The system retrieves tour information and quantity to calculate the total order value.
* The user enters personal information: name, phone number, email, notes (if any).
* The system checks the entered information; if valid, it saves the order information into the database.
* The system returns the order result and displays a success or error message.

1. *Data to be Stored:*

Orders Table: Stores order information.

* id: Order ID.
* user\_id: User ID (if applicable).
* tour\_id: ID of the booked tour.
* name: Customer name.
* phone: Customer phone number.
* email: Customer email address.
* quantity: Number of participants in the tour.
* total: Total value of the order.
* note: Customer notes.
* created\_at: Order creation date.
* updated\_at: Order update date.
  + - 1. ***UI/UX Design***
* Order Interface:
* Form to enter information: name, phone number, email, notes.
* Display tour information and order value.
* "Place Order" button to submit the order information.
* Order Information:

Display detailed information for the customer to review the order, including: name, phone number, email, tour, number of participants, total order value, notes.

* + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Order Functionality:*

Display form order

Fill customer’s information

Choose tour and quantity of participants

Valid information

Invalid information

Validate the order information

Save order to database

Display notification

1. *Detailed Steps*

* Enter Personal Information:

The user enters necessary information such as name, phone number, email, and notes (if any).

* Select Tour and Quantity:

The user selects a tour from the available list and enters the number of participants.

* Validate Information:

The system checks the validity of the input information, such as the quantity of tours, email, and phone number.

* Save Order:

If the information is valid, the system saves the order into the database.

* Notify Result:

The system displays a success message if the order is placed successfully, or an error message if there is an issue.

* + - 1. ***Solution Algorithm***
* Step 1: Receive input information (tour, quantity, personal information).
* Step 2: Validate the fields of information, such as ensuring the name is not empty, the phone number is in the correct format, and the email is valid.
* Step 3: If the information is valid, calculate the total order value based on the number of participants and the tour price.
* Step 4: Save the order information into the orders\_tours table.
* Step 5: Return the order result to the user (success or error).
  + 1. **Admin Order Management Functionality**
       1. ***Detailed Functionality***

1. *Input Information:*

* Order Information:
* user\_id: ID of the user.
* Name.
* Phone Number.
* Email.
* Notes.
* Tour Booking Information:
* tour\_id: ID of the tour.
* Number of participants.
* Customer Notes (if any).

1. *Output Information:*

* List of Orders: Display all orders with basic information (customer name, tour, quantity, total value).
* Order Details: Display complete information about the order, including the orderer, tour, number of participants, total value, and notes.

1. *Processing Method:*

* Admin can view the list of existing orders.
* When a customer places an order, the system automatically calculates the total amount based on the quantity and tour price.
* The order will be saved in the system with customer and tour information.
* Admin can update, cancel, or view details of each order.

1. *Data to be Stored:*

Orders Table:

* id: Order ID.
* user\_id: User ID.
* tour\_id: ID of the booked tour.
* quantity: Number of participants.
* total: Total order value.
* note: Customer notes.
* created\_at: Order creation date.
* updated\_at: Order update date.
  + - 1. ***UI/UX Design***

Order List Interface:

* Display a list of all orders with information: customer name, phone number, email, tour, number of participants, total value.
* Actions: update order information, cancel orders.
  + - 1. ***Activity Diagram (Flowchart)***

1. *Overview Flowchart of Admin Order Management Functionality:*

Customer order tour

Calculate total order cost

Update order’s information

Delete order

Validate input information

Save order’s information to database

Display list of orders

Update database

1. *Detailed Steps*

* Customer Places Order:

When the customer completes the tour booking, the system calculates the total order value, checks the validity of the input data, records the order, and saves it into the database.

* Admin Receives Order:

Admin can access the system and view the list of new orders.

* Update Order:

Admin can change the order information (customer name, phone number, email, tour, number of participants).

* Cancel Order:

If necessary, Admin can cancel the order.

* + - 1. ***Solution Algorithm***
* Step 1: Receive input information from the user (booked tour, number of participants, personal information).
* Step 2: Calculate the total order value based on the quantity and tour price.
* Step 3: Check the validity of the input data and save the order information into the orders table.
* Step 4: Admin views the list of orders from the database.
* Step 5: Admin can change order information or cancel the order if needed.
* Step 6: The system saves the admin's changes and displays the updated list of orders.

1. **Validation Checklists**

|  |  |  |
| --- | --- | --- |
| **Table** | **Column** | **Validation** |
| **users** | name | * Required * String, maximum length of 255 characters. |
| phone | * Required * Unique * String, valid phone number format. |
| email | * Required * Unique * String, valid email format. |
| password | * Required * String, minimum length of 6 characters. |
| avatar | * Nullable * Check if it is an image file. |
| role | * Required * Boolean, accepts only true/false values. |
| verification\_code | * Required * Integer |
| status | * Required * Boolean, accepts only true/false values. |
| **tours** | name | * Required * String, maximum length of 255 characters. |
| price | * Required * Integer, must be a positive value. |
| image | * Nullable * Check if it is an image file. |
| location | * Required * Check string format. |
| features | * Required * Check string format. |
| besttime | * Required * Check string format. |
| directions | * Required * Check string format. |
| trekkingroutes | * Required * Check string format. |
| items | * Required * Check string format. |
| cautions | * Required * Check string format. |
| requirements | * Required * Check string format. |

|  |  |  |
| --- | --- | --- |
| **Table** | **Column** | **Validation** |
| **orders\_tours** | user\_id | * Nullable, check if the user is not logged in. * Foreign key referencing the users table. |
| tour\_id | * Required * Foreign key referencing the tours table |
| name | * Required * String, maximum length of 255 characters. |
| phone | * Required * String, valid phone number format. |
| email | * Required * String, valid email format. |
| quantity | * Required * Integer, must be greater than 0. |
| total | * Required * Integer, positive value, automatically calculated based on quantity and tour price. |
| note | * Nullable * String, maximum length unlimited. |
| **branches** | name | * Required * String, maximum length of 255 characters. |
| address | * Required * String |
| phone | * Required * String, valid phone number format. |
| positioning | * Required * Valid coordinate format (latitude, longitude). |
| **galleries** | tours\_id | * Required * Foreign key referencing the tours table |
| images | * Nullable. * Check for valid JSON format. * JSON content contains a valid image URL. |
| videos | * Nullable. * Check for valid JSON format. * JSON content contains a valid video URL. |

1. **Task sheet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Start time** | **Finish time** | **Implemented by** | **Evaluation (%)** |
| **Feasibility study phase** |  |  |  |  |
| Determine the overall objectives of the project. | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 25% |
| Determine the overall objectives of the project. | 21/08/2024 | 23/08/2024 | Quan Luu Minh | 25% |
| Determine the overall objectives of the project. | 21/08/2024 | 23/08/2024 | Manh Nguyen Duc | 25% |
| Determine the overall objectives of the project. | 21/08/2024 | 23/08/2024 | Hau Nguyen Van | 25% |
| Evaluate human resources. | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 100% |
| Identify technologies and frameworks that can be used. | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 25% |
| Identify technologies and frameworks that can be used. | 21/08/2024 | 23/08/2024 | Quan Luu Minh | 75% |
| Analyze the integration with external technologies. | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 100% |
| **Requirement Analysis phase** |  |  |  |  |
| Clearly define the functions and features needed in the system. | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 50% |
| Clearly define the functions and features needed in the system. | 21/08/2024 | 23/08/2024 | Quan Luu Minh | 50% |
| Analyze database requirements and necessary tables. | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 50% |
| Analyze database requirements and necessary tables. | 21/08/2024 | 23/08/2024 | Quan Luu Minh | 50% |
| **Design phase** |  |  |  |  |
| Design the database architecture. | 24/08/2024 | 30/08/2024 | Hieu Nguyen Van | 50% |
| Design the database architecture. | 24/08/2024 | 30/08/2024 | Quan Luu Minh | 50% |
| Design the system architecture. | 24/08/2024 | 30/08/2024 | Manh Nguyen Duc | 50% |
| Design the system architecture. | 24/08/2024 | 30/08/2024 | Hau Nguyen Van | 50% |
| Create the ERD (Entity-Relationship Diagram). | 21/08/2024 | 23/08/2024 | Hieu Nguyen Van | 50% |
| Create the ERD (Entity-Relationship Diagram). | 21/08/2024 | 23/08/2024 | Quan Luu Minh | 50% |
| Design activity diagrams for the main functions. | 24/08/2024 | 30/08/2024 | Manh Nguyen Duc | 50% |
| Design activity diagrams for the main functions. | 24/08/2024 | 30/08/2024 | Hau Nguyen Van | 50% |
| Design UI/UX mockups for the main pages. | 24/08/2024 | 30/08/2024 | Hieu Nguyen Van | 25% |
| Design UI/UX mockups for the main pages. | 24/08/2024 | 30/08/2024 | Quan Luu Minh | 25% |
| Design UI/UX mockups for the main pages. | 24/08/2024 | 30/08/2024 | Manh Nguyen Duc | 25% |
| Design UI/UX mockups for the main pages. | 24/08/2024 | 30/08/2024 | Hau Nguyen Van | 25% |
| Design the API if integrating with external services. | 24/08/2024 | 30/08/2024 | Hieu Nguyen Van | 100% |
| **Development phase** |  |  |  |  |
| Build the database and tables based on the designed migrations. | 31/08/2024 | 05/09/2024 | Hieu Nguyen Van | 25% |
| Build the database and tables based on the designed migrations. | 31/08/2024 | 05/09/2024 | Quan Luu Minh | 25% |
| Build the database and tables based on the designed migrations. | 31/08/2024 | 05/09/2024 | Manh Nguyen Duc | 25% |
| Build the database and tables based on the designed migrations. | 31/08/2024 | 05/09/2024 | Hau Nguyen Van | 25% |
| Develop the basic functions. | 31/08/2024 | 23/09/2024 | Hieu Nguyen Van | 25% |
| Develop the basic functions. | 31/08/2024 | 23/09/2024 | Quan Luu Minh | 25% |
| Develop the basic functions. | 31/08/2024 | 23/09/2024 | Manh Nguyen Duc | 25% |
| Develop the basic functions. | 31/08/2024 | 23/09/2024 | Hau Nguyen Van | 25% |
| Develop the user authentication system. | 31/08/2024 | 23/09/2024 | Manh Nguyen Duc | 45% |
| Develop the user authentication system. | 31/08/2024 | 23/09/2024 | Hau Nguyen Van | 45% |
| Develop the user authentication system. | 31/08/2024 | 23/09/2024 | Quan Luu Minh | 10% |
| Build the ordering function, calculate the total amount, and save it to the database. | 31/08/2024 | 23/09/2024 | Hau Nguyen Van | 100% |
| Develop the management section for the branches | 31/08/2024 | 15/09/2024 | Hieu Nguyen Van | 100% |
| Develop the management section for the gallery | 31/08/2024 | 15/09/2024 | Hieu Nguyen Van | 100% |
| Integrate Maps API for displaying branches. | 31/08/2024 | 15/09/2024 | Hieu Nguyen Van | 100% |
| Write seeders for the database to add sample data. | 31/08/2024 | 05/09/2024 | Hieu Nguyen Van | 25% |
| Write seeders for the database to add sample data. | 31/08/2024 | 05/09/2024 | Quan Luu Minh | 25% |
| Write seeders for the database to add sample data. | 31/08/2024 | 05/09/2024 | Manh Nguyen Duc | 25% |
| Write seeders for the database to add sample data. | 31/08/2024 | 05/09/2024 | Hau Nguyen Van | 25% |
| Optimize the frontend with Tailwind CSS for the user interface. | 31/08/2024 | 24/09/2024 | Quan Luu Minh | 25% |
| Optimize the frontend with Tailwind CSS for the user interface. | 31/08/2024 | 24/09/2024 | Manh Nguyen Duc | 75% |
| **Testing phase** |  |  |  |  |
| Test the functionality for the main features: managing tours, orders, and users. | 06/09/2024 | 25/09/2024 | Hieu Nguyen Van | 10% |
| Test the functionality for the main features: managing tours, orders, and users. | 06/09/2024 | 25/09/2024 | Quan Luu Minh | 30% |
| Test the functionality for the main features: managing tours, orders, and users. | 06/09/2024 | 25/09/2024 | Manh Nguyen Duc | 30% |
| Test the functionality for the main features: managing tours, orders, and users. | 06/09/2024 | 25/09/2024 | Hau Nguyen Van | 30% |
| Test the ordering process. | 06/09/2024 | 25/09/2024 | Hau Nguyen Van | 100% |
| Test the integrated APIs. | 06/09/2024 | 25/09/2024 | Hieu Nguyen Van | 100% |
| Test the user interface across multiple browsers and devices. | 06/09/2024 | 25/09/2024 | Manh Nguyen Duc | 100% |
| Fix the issues discovered during testing. | 06/09/2024 | 25/09/2024 | Hieu Nguyen Van | 25% |
| Fix the issues discovered during testing. | 06/09/2024 | 25/09/2024 | Quan Luu Minh | 25% |
| Fix the issues discovered during testing. | 06/09/2024 | 25/09/2024 | Manh Nguyen Duc | 25% |
| Fix the issues discovered during testing. | 06/09/2024 | 25/09/2024 | Hau Nguyen Van | 25% |