A Presentation On

Use-Case Diagram, Domain, SSD & ER Model Of

MITHO-MITHO

"A FOOD TRAIL SHARING WEB-APP"

Presented By:

Aastha Paudel [PAS077BCT002]

Aayush Adhikari [PAS077BCT003]

Anup Neupane [PAS077BCT010]

Susmita Aryal [PAS077BCT044]

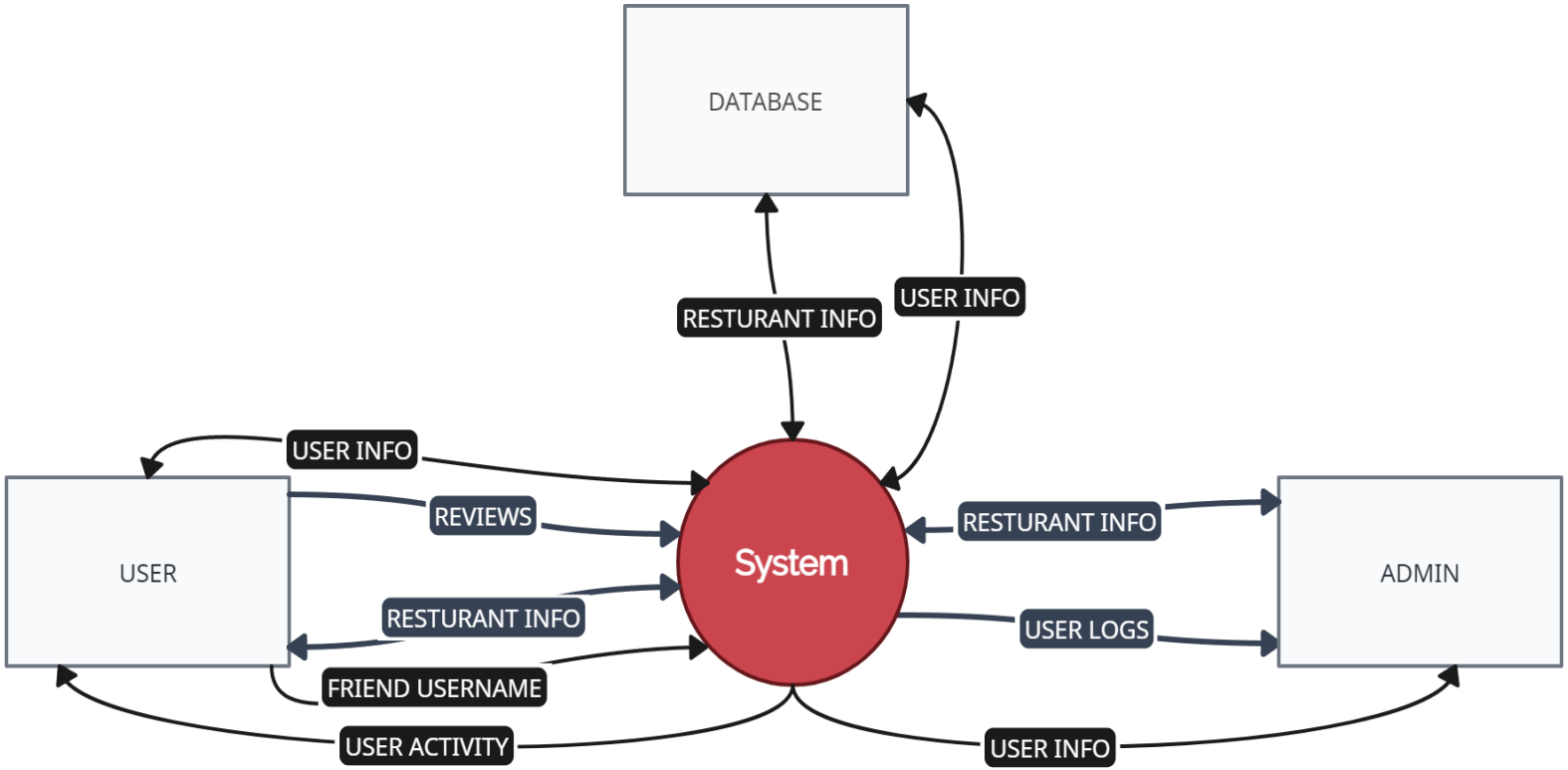
PROBLEM STATEMENT

The absence of a dedicated platform for restaurant discovery, dining history tracking, reviewing, and sharing culinary experiences, as well as connecting with fellow food enthusiasts, poses a significant challenge.

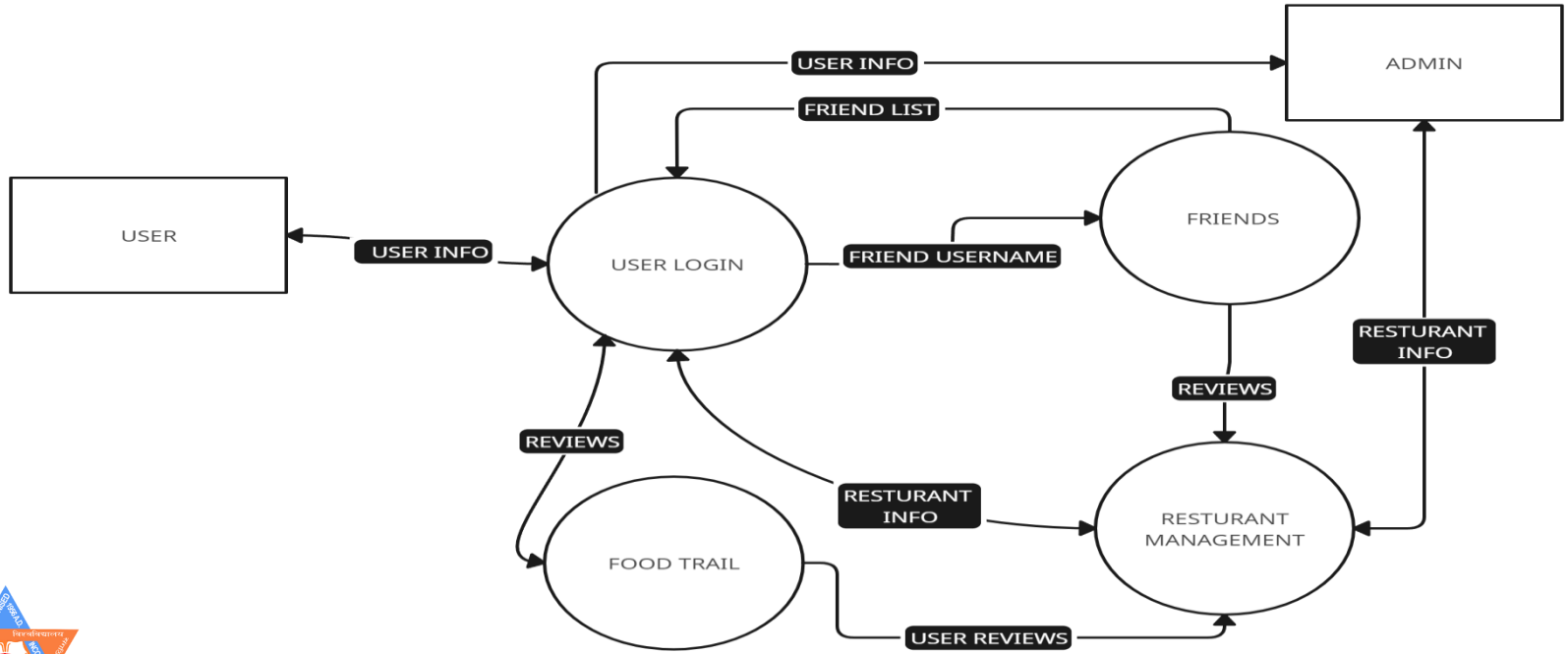
OBJECTIVE

To establish dedicated platform for restaurant discovery, dining history tracking, reviewing, and sharing culinary experiences, as well as connecting with fellow food enthusiasts.

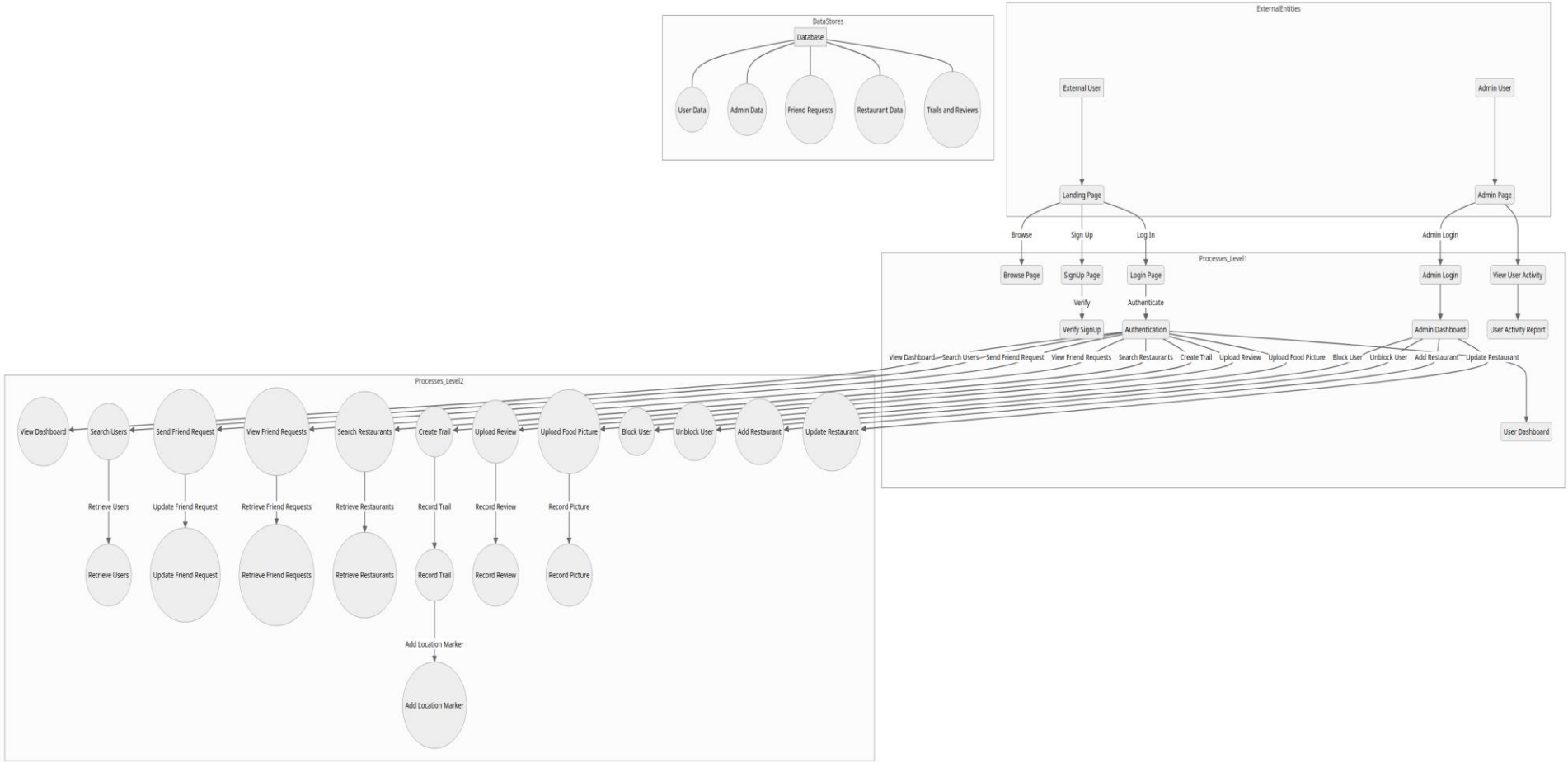
Data-Flow Diagram



Data-Flow Diagram

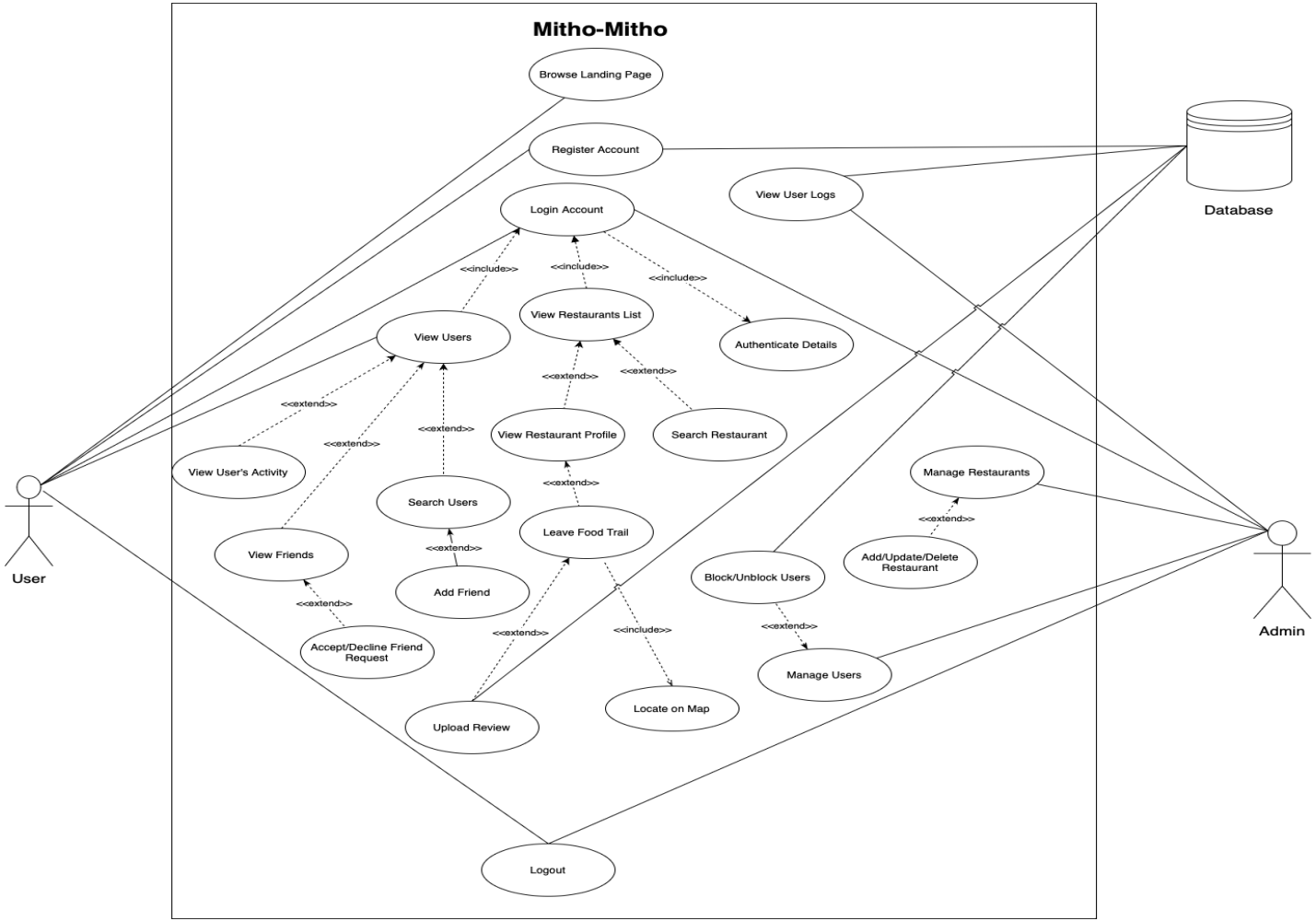


Data-Flow Diagram



Use-Case Diagram

- A behavioural diagram used in Software engineering and system analysis .
- Visualize the interactions between users and a system.
- Provides a graphical representations of the various use cases, actors, and their relationships within the system.
- Provides high-level overview of system's functionality from the user's perspective.


Mitho-Mitho
Browse Landing Page
Register Account
Login Account
View Users
View Restaurants List
Authenticate Details
View User's Activity
View Restaurant Profile
Search Restaurant
Search Users
Leave Food Trail
View Friends
Add Friend
Accept/Decline Friend Request
Upload Review
Locate on Map
Logout
View User Logs
Database
Manage Restaurants
Add/Update/Delete Restaurant
Block/Unblock Users
Manage Users
User
Admin
<<include>>
<<extend>>


Mitho-Mitho
Browse Landing Page
Register Account
Login Account
View User Logs
View Users
View Restaurants List
Authenticate Details
View User's Activity
View Restaurant Profile
Search Restaurant
Search Users
Leave Food Trail
Add/Update/Delete Restaurant
View Friends
Add Friend
Manage Restaurants
Accept/Decline Friend Request
Block/Unblock Users
Upload Review
Locate on Map
Manage Users
Logout
<<include>>
<<extend>>
User
Admin

Fully-Dressed Use Case

1. Browse landing page

Use case ID	UC1
Use Case Name	Browse Landing Page
Primary actor	User
Stake holders and interests	User : browses the landing page
Preconditions	The user has access to the internet.
Post conditions	The user has viewed the landing page content.
Basic flow	User arrives at web page.
Extensions (Alternative flow)	If the landing page undergoes updates or maintenance, the system may display a maintenance message or redirect the user to an alternative page temporarily.
Special requirements	The landing page should load quickly to enhance user experience.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Responsive Web Design frameworks. Data : text content, Images, videos
Frequency of Occurrence	High
Open issues	Performance optimization to ensure fast loading times. monitoring user behavior and feedback to iteratively improve the landing page design and content.

2. Register account

Use case ID	UC2
Use Case Name	Register account
Primary actor	User, Database
Stake holders and interests	User: wants to access the services of the app and login to the site.
Preconditions	User should have a registered gmail account.
Postconditions	An account for the user is registered and can be used for login.
Basic flow	1. User arrives at site. 2. User clicks on register. 3. User inserts personal details. 4. User press on register button.
Extensions	If the user provides invalid information or misses required fields during registration, the system prompts them to correct the errors and resubmit the form.
Special requirements	Password should be securely hashed and stored to protect user accounts from unauthorized access.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : User provided information
Frequency of Occurrence	Moderate

3. Login account

Use case ID	UC3
Use case name	Login account
Primary actor	User
Precondition	User must have registered account
Post condition	User gets logged into his/her account
Basic flow	1.User arrives at site 2.User clicks on login 3.User adds login details 4.User gets logged in
Special requirement	The login process must be secure.
Stakeholders and interests	User: wants to login to an account.
Extensions (Alternative flows)	If the user has forgotten their password, they can request a password reset link through the “Forgot Password” functionality.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : User provided information
Frequency of Occurrence	High
Open issues	Monitoring for suspicious login activity.

4. Authenticate details

Use case ID	UC4
Use case name	Authenticate details
Primary actor	Admin, Database
Precondition	Login details must be accurate
Post condition	User gets logged in
Stakeholders and interests	Admin: retrieves user details and checks/matches with the details stored on database and allows/denies login.
Basic flow	1. User arrives at site. 2.User clicks on login. 3.User adds login details. 4.Login details gets checked. 5.If credentials matches, user access to website services.
Extensions (Alternative flows)	If the username/email provided by the user does not exist in the database, the system denies authentication and prompts the user to verify their details or register for an account.
Special requirements	The authentication process must be secure.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : User provided information
Frequency of Occurrence	High
Open issues	Monitoring for potential security vulnerabilities in the authentication process.

5. See personal profile

Use case ID	UC5
Use case name	See personal profile
Primary actor	User
Precondition	User should get logged in
Post condition	User gets to see his/her profile history
Stakeholders and interests	User : wants to view his/her dining history, trials and previously visited restaurants on map.
Basic flow	1. User arrives at site. 2. User gets logged in. 3. User sees his/her profile.
Extensions (Alternative flows)	If the user has not provided certain profile information, the system may display placeholder text.
Special requirements	The personal profile page should be responsive.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : User provided information
Frequency of Occurrence	Moderate
Open issues	Ensuring data accuracy and integrity when updating profile information.

6. Add personal details

Use case ID	UC5
Use case name	See personal profile
Primary actor	User
Precondition	User should get logged in
Post condition	User gets to see his/her profile history
Stakeholders and interests	User : wants to view his/her dining history, trials and previously visited restaurants on map.
Basic flow	1. User arrives at site. 2. User gets logged in. 3. User sees his/her profile.
Extensions (Alternative flows)	If the user has not provided certain profile information, the system may display placeholder text.
Special requirements	The personal profile page should be responsive.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : User provided information
Frequency of Occurrence	Moderate
Open issues	Ensuring data accuracy and integrity when updating profile information.

7. Search restaurants

Use case ID	UC7
Use case name	Search restaurants
Primary actor	User
Precondition	User must get logged in
Post condition	User gets to see a list of restaurants.
Stakeholders and interests	User : wants to discover restaurants interest.
Basic flow	1. User arrives at site. 2. User gets logged in. 3. User searches for restaurant. 4. A restaurant for the keyword searched (either name or address) if exists will be displayed.
Extensions	If the users search query returns no results, the system informs the user and may suggest alternative search terms or criteria.
Special requirements	The search functionality should be optimized for performance, providing quick and accurate results even with large datasets.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : Restaurant information
Frequency of Occurrence	High
Open issues	Continious improvement of search algorithms and relevance ranking to provide the best possible results to users.

8. View restaurant profile

Use case ID	UC8
Use case name	View restaurant profile
Primary actor	User
Precondition	User must have logged in
Post condition	User gets the whole profile of restaurants
Stakeholders and interests	User : wants to check the restaurants profile and reviews
Basic flow	1. User arrives at site. 2. User gets logged in. 3. User searches for a restaurant. 4. A restaurant with matching details with searched keywords is displayed. 5. User clicks on that restaurant. 6. User gets to see the restaurant profile.
Extensions	If the restaurants profile information is incomplete or missing, the system may display placeholders or notify the user that certain details are not available.
Special requirements	Restaurant profile information should be accurate, up to date, and consistent.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : Restaurant information
Frequency of Occurrence	Moderate
Open issues	Ensuring the accuracy and completeness of restaurant profile information, especially in cases where data may change frequently.

9. Leave a food trail

Use case ID	UC10
Use case name	Leave a food trail.
Primary actor	User
Precondition	User must get logged in.
Post condition	A trail mark is shown on map as well as sidebar.
Stakeholders and interests	User : wants to leave a food trial/review at that restaurant
Basic flow	1. User arrives at site. 2. User gets logged in to his/her account. 3. User searches for a restaurant. 4. User leaves a trial.
Extensions	If the user decides not to share the food trail immediately after creating it, they can save it as a draft and return to it later to share or edit.
Special requirements	The food trail feature should support multimedia content such as photos to enhance the user experience and provide visual context.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : Food trail title, List of restaurants included in the trail
Frequency of Occurrence	Moderate
Open issues	Continuous improvement of the food trail creation interface to enhance usability.

10. Mark location on map

Use case ID	UC11
Use case name	Mark location on map
Primary actor	User, admin
Precondition	User must log in and user leaves a trial
Post condition	A mark is created on map where trial is left by API
Stakeholders and interests	User : wants to leave/mark a trail, review on a restaurant Admin: marks the location on map where the trail is left.
Basic flow	1. User arrives at site. 2. User gets logged in to his/her account. 3. User leaves a trial. 4. A mark is left at corresponding location on map.
Extensions	If the user encounters errors or technical issues while making the location on map, the system notifies the user and may provide steps to retry later.
Special requirements	The mapping feature should provide a user friendly interface for interfacing with the map and marking locations.
Technology and data variation list	Technology :mapping API(e.g. Google Maps), HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : marked location coordinates or address
Frequency of Occurrence	Moderate
Open issues	Continuous improvement of the mapping interface to enhance user experience.

11. Search users

Use case ID	UC12
Use case name	Search users
Primary actor	User
Precondition	User must have logged in his/her account.
Post condition	The search user of exists on database will be displayed.
Stakeholders and interests	User : want to search a user on a site.
Basic flow	1. User arrives at site. 2. User gets logged in to his/her account. 3. User searches for other users 4. The user searched (is exists) will be displayed on the page.
Extensions	If the users search query returns no results, the system informs the user and may suggest alternative search terms or criteria.
Special requirements	The search functionality should be optimized for performance, providing quick and accurate results even with large datasets.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : User provided information, user generated content
Frequency of Occurrence	Moderate to high
Open issues	Continuous improvement of search algorithms and relevance ranking to provide the best possible results to users.

12. Send friend request

Use case ID	UC13
Use case name	Send friend request
Primary actor	User
Precondition	User must get logged in to his/her account.
Post condition	A request will be sent and will be indicated bt pending status on page.
Stakeholders and interests	User : wants to send a friend request to another request.
Basic flow	1. User arrives at site. 2. User logins to the account. 3. User navigates to friends tab and goes to search for friend tab. 4. User clicks on add friend button on the profile of the user to be added. 5. Sent friend request will be indicated by pending status.
Extensions	If the user encounters errors or technical issues while sending the friend request, the system notifies the user and may provide troubleshooting steps.
Special requirements	The friend request feature should be intuitive and user friendly, guiding users through the process of sending and responding to friend request.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : friend request metadata, user relationships, notification settings
Frequency of Occurrence	Moderate
Open issues	Continuous improvement of the friend request feature to enhance usability and user experience.

13. Accept friend request, decline friend request

Use case ID	UC14
Use case name	Accept friend request, decline friend request
Primary actor	User
Precondition	1. User needs to get logged in. 2. To send a friend request, the user whom the request is to be sent must exist within the database, to decline/accept, the user should have received the request.
Post condition	Is a request is sent, declined or accepted, corresponding stars label changes would be reflected.
Stakeholders and interests	User : wants to accept or decline the receive request.
Basic flow	1. User arrives at site. 2. User logins to his/her account. 3. User navigates to friends tab and performs the intended actions and changes are reflected accordingly.
Extensions	If the user encounters errors or technical issues while accepting or declining friend requests, the system notifies the user and may provide troubleshooting steps or prompt them to retry later.
Special requirements	The friend request management feature should be intuitive and user friendly, allowing users to easily accept or decline incoming requests.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : friend request metadata, user relationships, notification settings
Frequency of Occurrence	Variable frequency
Open issues	Continuous improvement of the friend request management feature to enhance usability and user experience.

14. See users activity

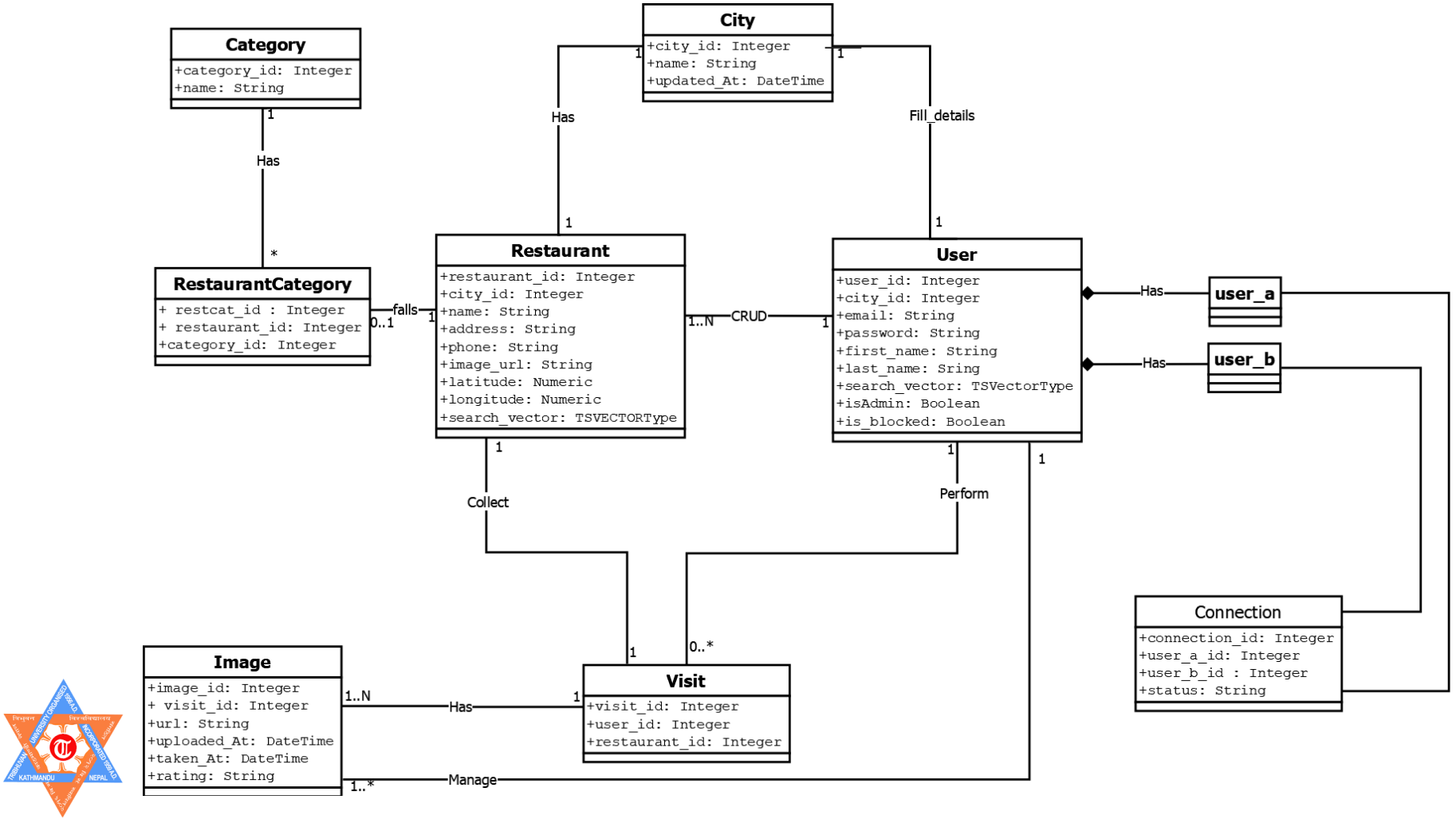
Use case ID	UC15
Use case name	See users activity.
Primary actor	User
Precondition	User needs to have logged into his/her account.
Post condition	Profile of other users is displayed which included food reviews and recently visited locations tracked in map.
Stakeholders and interests	User : wants to see the activity of friends and interests or other users.
Basic flow	1. User arrives at site. 2. User logins to his/her account. 3. User navigates to my friends tab or search friends tab. 4. User searches for an account (if not friend), otherwise clicks on the name of user. 5. User profile of the user is displayed.
Extensions	If the specified users activity feed is empty or contains no recent activity, the system informs the user that there is no activity to display.
Special requirements	The activity viewing feature should provide a comprehensive and up to date overview of the specified users recent actions on the platform.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : user activity data, user privacy settings
Frequency of Occurrence	Variable frequency
Open issues	Continuous improvement of the activity feed feature to enhance usability and user experience.

15. Logout

Use case ID	UC16
Use case name	Logout.
Primary actor	User
Precondition	User must have logged into the site.
Post condition	User gets logged out of the amount and “you have been logged out of the account” message is displayed.
Stakeholders and interests	User : wants to get logged out of the account.
Basic flow	1. User arrives at the site. 2. User logins to his/her account. 3. User clicks on drop bar menu and clicks on logout 4. User gets logged out of the account.
Extensions	If the users session is inactive for a prolonged period, the system may automatically log out the user to ensure security.
Special requirements	The logout functionality should securely terminate the users session to prevent unauthorized access to their account.
Technology and data variation list	Technology : HTML, CSS, JavaScript, Backend (PHP, Python) Database (MySQL, MongoDB) Data : user session data, user preferences or settings that may need to be saved before logout.
Frequency of Occurrence	Low to moderate
Open issues	Addressing any user feedback or complaints regarding issues with logging out, such as difficulty finding the logout option or slow logout process.

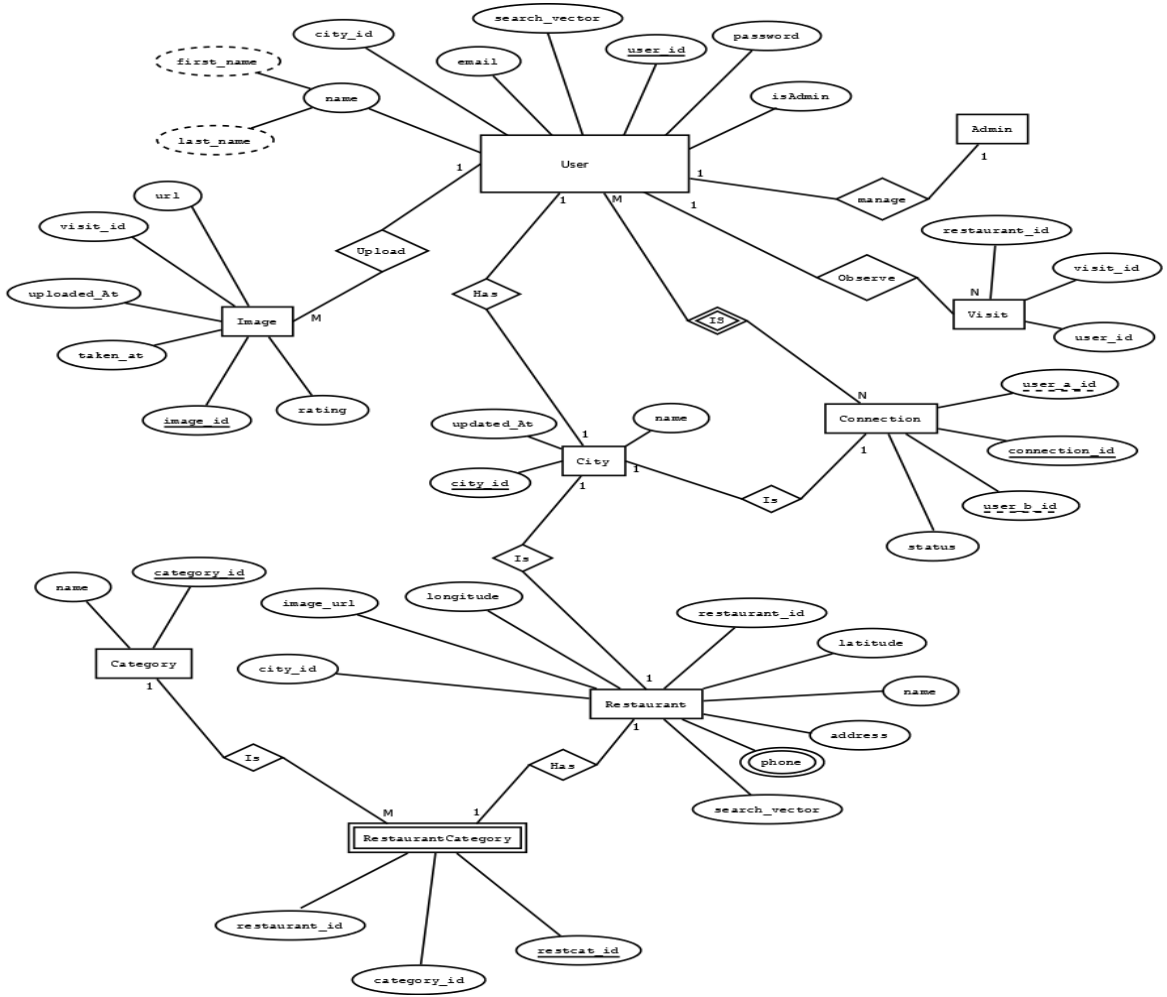
Domain Model

- A fundamental component of software engineering that helps in visualizing and organizing the conceptual structure of a system.
- Serves as a blueprint for understanding the key entities, their relationships, and the behavior of a system.
- Visual representation of conceptual classes or real-situation objects in a domain

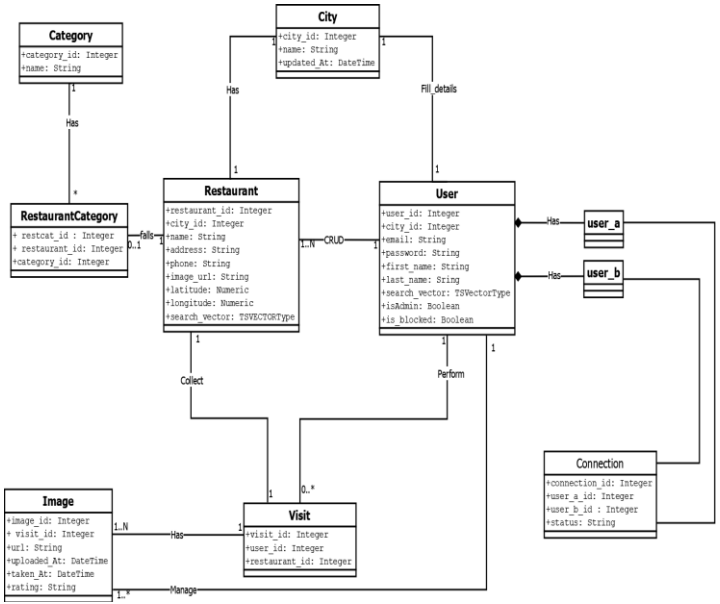

Category
+category_id: Integer
+name: String
City
+city_id: Integer
+name: String
+updated_At: DateTime
Has
Fill_details
Has
RestaurantCategory
+ restcat_id : Integer
+ restaurant_id: Integer
+category_id: Integer
falls
Restaurant
+restaurant_id: Integer
+city_id: Integer
+name: String
+address: String
+phone: String
+image_url: String
+latitude: Numeric
+longitude: Numeric
+search_vector: TSVECTORType
CRUD
User
+user_id: Integer
+city_id: Integer
+email: String
+password: String
+first_name: String
+last_name: Sring
+search_vector: TSVectorType
+isAdmin: Boolean
+is_blocked: Boolean
Has
user_a
Has
user_b
Collect
Perform
Connection
+connection_id: Integer
+user_a_id: Integer
+user_b_id : Integer
+status: String
Image
+image_id: Integer
+ visit_id: Integer
+url: String
+uploaded_At: DateTime
+taken_At: DateTime
+rating: String
Has
Visit
+visit_id: Integer
+user_id: Integer
+restaurant_id: Integer
Manage

ER Diagram

- Visual representation of different entities within a system and how they related
- Tool used to design & model relational database, shows logical structure of data
- Use symbol to represent entities, attributes& relationship

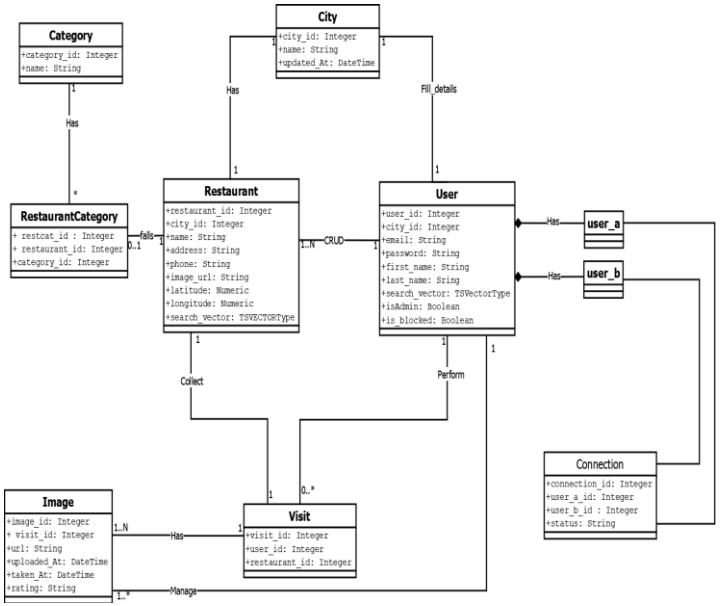

search_vector
city_id
email
user_id
password
first_name
name
isAdmin
last_name
User
Admin
manage
url
visit_id
Upload
restaurant_id
Observe
visit_id
uploaded_At
Has
Image
M
N
Visit
taken_at
IS
user_id
user_a_id
image_id
rating
updated_At
name
Connection
connection_id
City
city_id
Is
user_b_id
status
Is
name
category_id
image_url
longitude
restaurant_id
latitude
Category
city_id
Restaurant
name
address
Is
Has
phone
search_vector
RestaurantCategory
restaurant_id
restcat_id
category_id

OOD to Object-Oriented Code



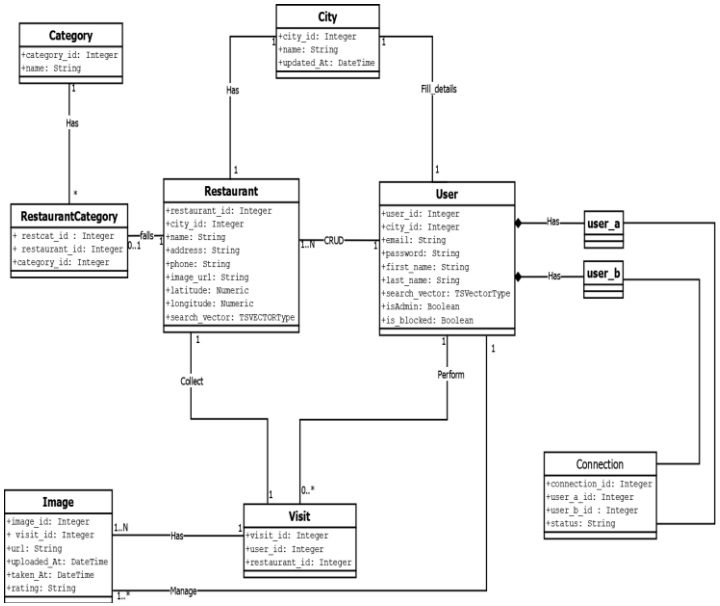

```python
class User(db.Model):
    """User of Mitho-Mitho website."""

    __tablename__ = "users"

    user_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    city_id = db.Column(db.Integer, db.ForeignKey('cities.city_id'), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    # Put name inside TSVectorType definition for it to be fulltext-indexed (searchable)
    search_vector = db.Column(TSVectorType('first_name', 'last_name'))
    isAdmin = db.Column(db.Boolean, default=False)
    is_blocked = db.Column(db.Boolean, default=False)

    city = db.relationship("City", backref=db.backref("users"))

    # profile_picture_url = db.Column(db.String(200), nullable=True)


    # def update_profile_picture_url(self):
    #     self.profile_picture = f"user_{self.user_id}_profile_picture.jpg"
    #     db.session.commit()

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<User user_id=%s email=%s>" % (self.user_id, self.email)

    def get_blocked_status(self):
        return self.is_blocked
```

OOD to Object-Oriented Code



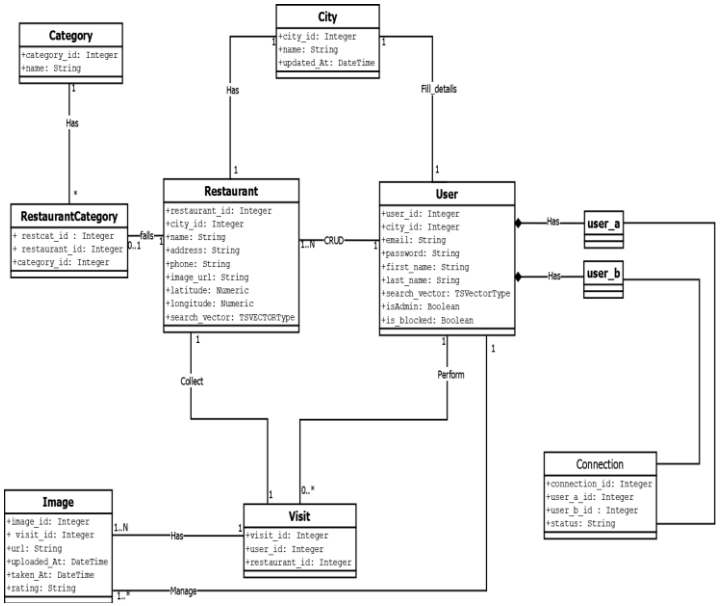

```python
class Image(db.Model):
    """Image uploaded by user for each restaurant visit."""

    __tablename__ = "images"

    image_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    visit_id = db.Column(db.Integer, db.ForeignKey('visits.visit_id'), nullable=False)
    url = db.Column(db.String(200), nullable=False)
    uploaded_At = db.Column(db.DateTime, default=datetime.datetime.utcnow)
    taken_At = db.Column(db.DateTime, nullable=True)
    rating = db.Column(db.String(100), nullable=True)

    visit = db.relationship("Visit", backref=db.backref("images"))

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<Image image_id=%s visit_id=%s>" % (self.image_id, self.visit_id)
```

OOD to Object-Oriented Code



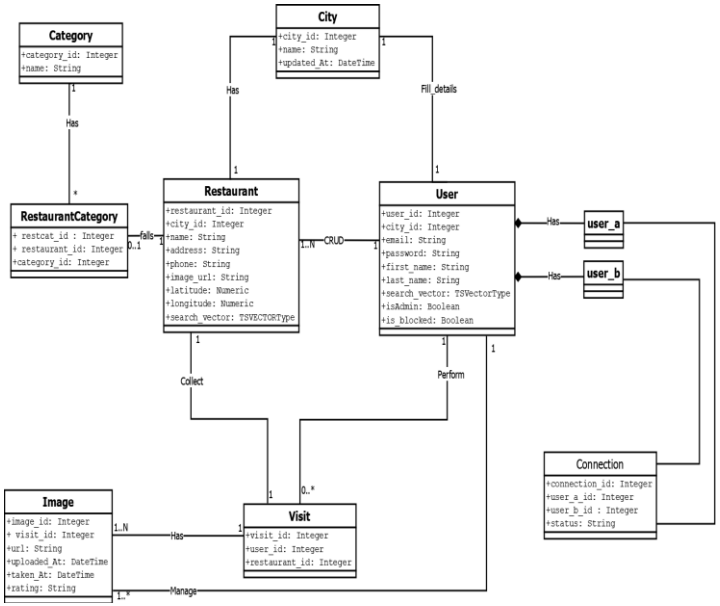

```python
class RestaurantCategory(db.Model):
    """Association table linking Restaurant and Category to manage the M2M relationship."""

    __tablename__ = "restaurantcategories"

    restcat_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    restaurant_id = db.Column(db.Integer, db.ForeignKey('restaurants.restaurant_id'), nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('categories.category_id'), nullable=False)

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<RestaurantCategory restcat_id=%s restaurant_id=%s category_id=%s>" % (
            self.restcat_id, self.restaurant_id, self.category_id)
```

OOD to Object-Oriented Code



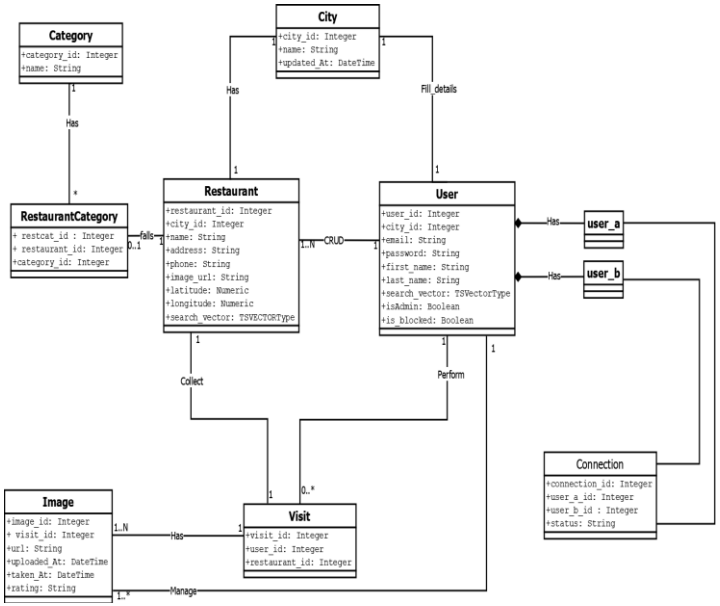

```python
class Category(db.Model):
    """Category of the restaurant."""

    __tablename__ = "categories"

    category_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    name = db.Column(db.String(100), unique=True, nullable=False)

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<Category category_id=%s name=%s>" % (self.category_id, self.name)
```

OOD to Object-Oriented Code



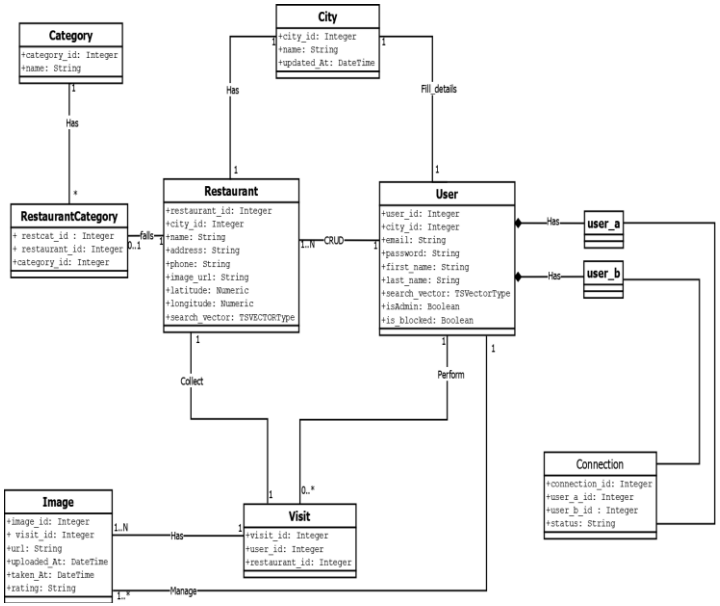

```python
class City(db.Model):
    """City where the restaurant is in."""

    __tablename__ = "cities"

    city_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    # Set default for timestamp of current time at UTC time zone
    updated_At = db.Column(db.DateTime, default=datetime.datetime.utcnow)

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<City city_id=%s name=%s>" % (self.city_id, self.name)
```

OOD to Object-Oriented Code



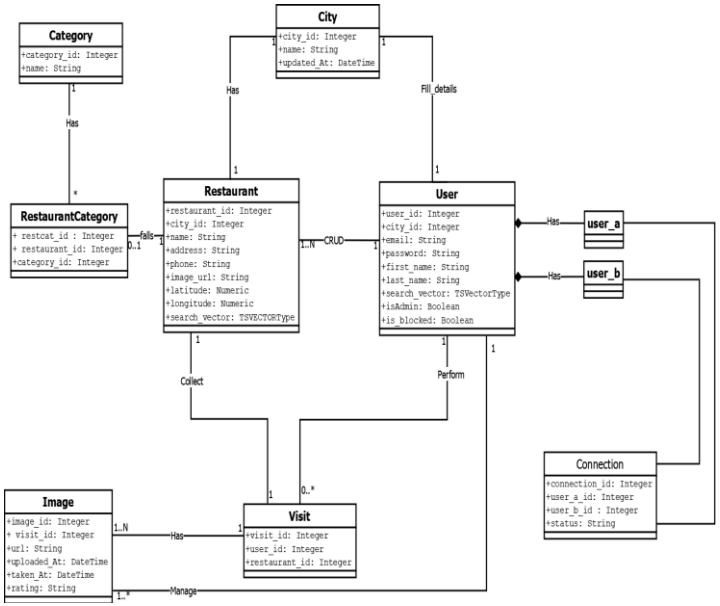

```python
class Visit(db.Model):
    """User's visited/saved restaurant on Breadcrumbs website.
    Association table between User and Restaurant.
    """

    __tablename__ = "visits"

    visit_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.user_id'), nullable=False)
    restaurant_id = db.Column(db.Integer, db.ForeignKey('restaurants.restaurant_id'), nullable=False)

    user = db.relationship("User", backref=db.backref("visits"))
    restaurant = db.relationship("Restaurant", backref=db.backref("visits"))

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<Visit visit_id=%s restaurant_id=%s>" % (self.visit_id, self.restaurant_id)
```

OOD to Object-Oriented Code




```python
class Restaurant(db.Model):
    """Restaurant on Breadcrumbs website."""

    __tablename__ = "restaurants"

    restaurant_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    city_id = db.Column(db.Integer, db.ForeignKey('cities.city_id'), nullable=False)
    name = db.Column(db.String(150), nullable=False)
    address = db.Column(db.String(150), nullable=False)
    phone = db.Column(db.String(20), nullable=True)
    image_url = db.Column(db.String(200), nullable=True)
    # Latitude and Longitude need to be Numeric, not Integer to have decimal places
    latitude = db.Column(db.Numeric, nullable=False)
    longitude = db.Column(db.Numeric, nullable=False)  # Fix the typo here
    # Put restaurant name and address inside definition of TSVectorType to be fulltext-indexed (searchable
    search_vector = db.Column(TSVectorType('name', 'address'))

    city = db.relationship("City", backref=db.backref("restaurants"))
    categories = db.relationship("Category", secondary="restaurantcategories", backref="restaurants")
    users = db.relationship("User", secondary="visits", backref="restaurants")

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<Restaurant restaurant_id=%s name=%s>" % (self.restaurant_id, self.name)
```

OOD to Object-Oriented Code

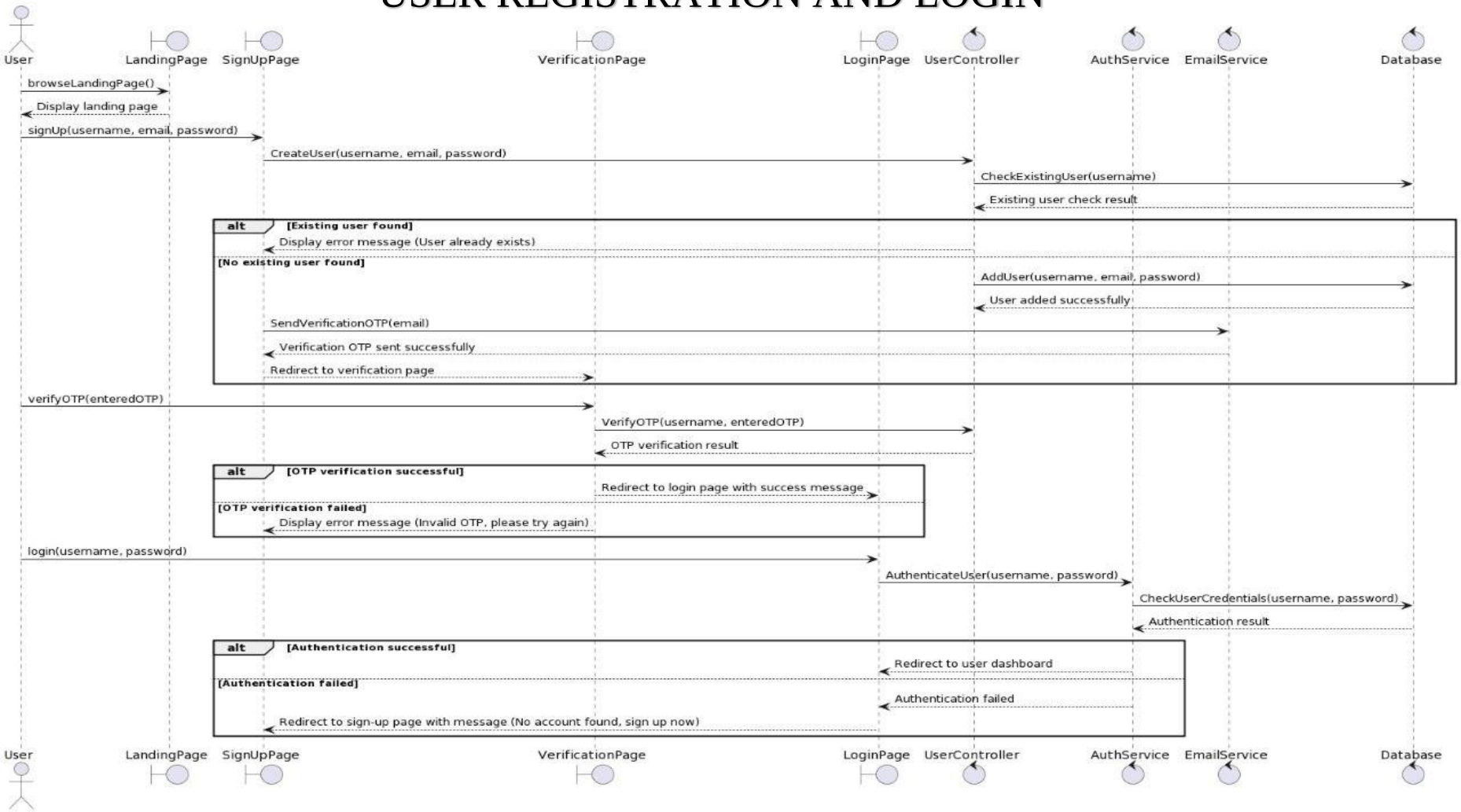



```python
class Connection(db.Model):
    """Connection between two users to establish a friendship and can see each other's info."""

    __tablename__ = "connections"

    connection_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    user_a_id = db.Column(db.Integer, db.ForeignKey('users.user_id'), nullable=False)
    user_b_id = db.Column(db.Integer, db.ForeignKey('users.user_id'), nullable=False)
    status = db.Column(db.String(100), nullable=False)

    # When both columns have a relationship with the same table, need to specify how
    # to handle multiple join paths in the square brackets of foreign_keys per below
    user_a = db.relationship("User", foreign_keys=[user_a_id], backref=db.backref("sent_connections"))
    user_b = db.relationship("User", foreign_keys=[user_b_id], backref=db.backref("received_connections"))

    def __repr__(self):
        """Provide helpful representation when printed."""
        return "<Connection connection_id=%s user_a_id=%s user_b_id=%s status=%s>" % (
            self.connection_id, self.user_a_id, self.user_b_id, self.status)
```

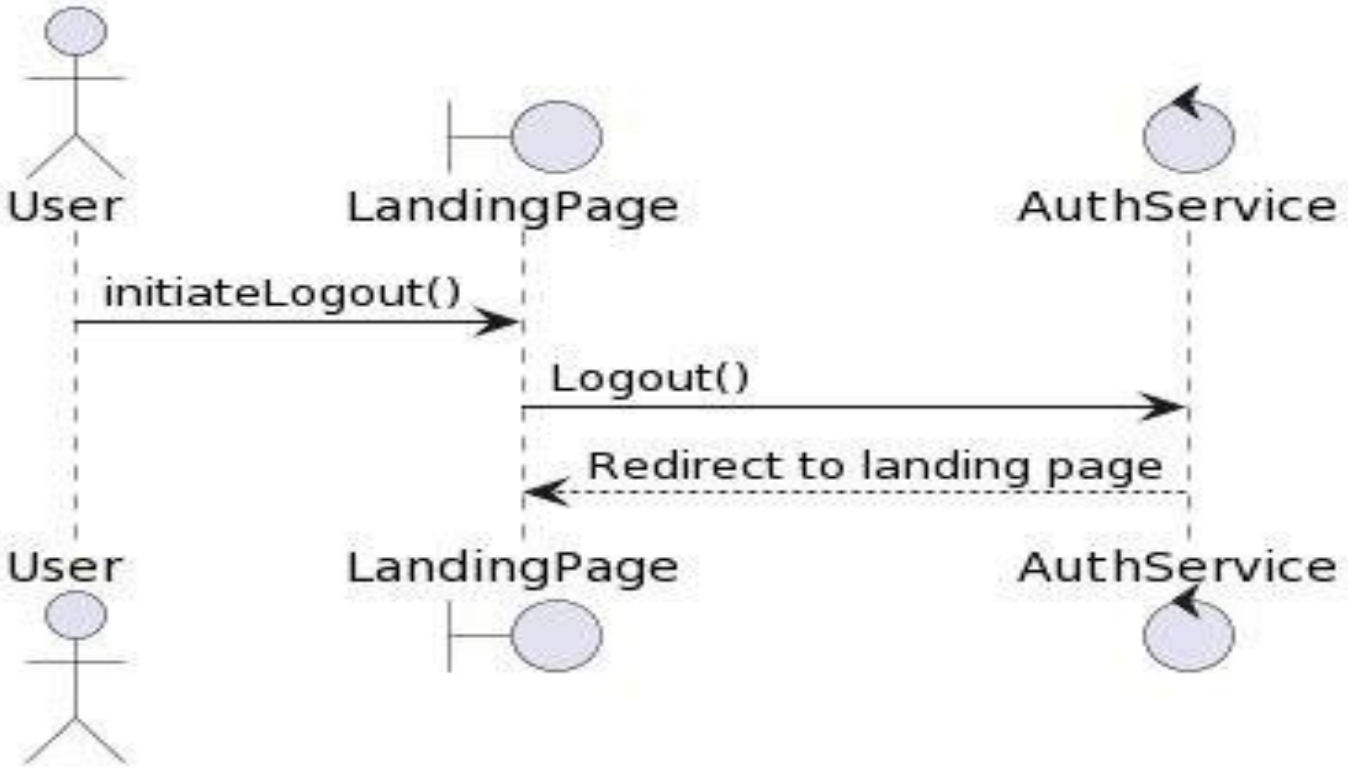
System Sequence Diagram

- Shows system events for one particular scenario of a use case
- Serves as a powerful communication aid between different stakeholders
- Clear and visual representation of the interactions between various components or actors within a system

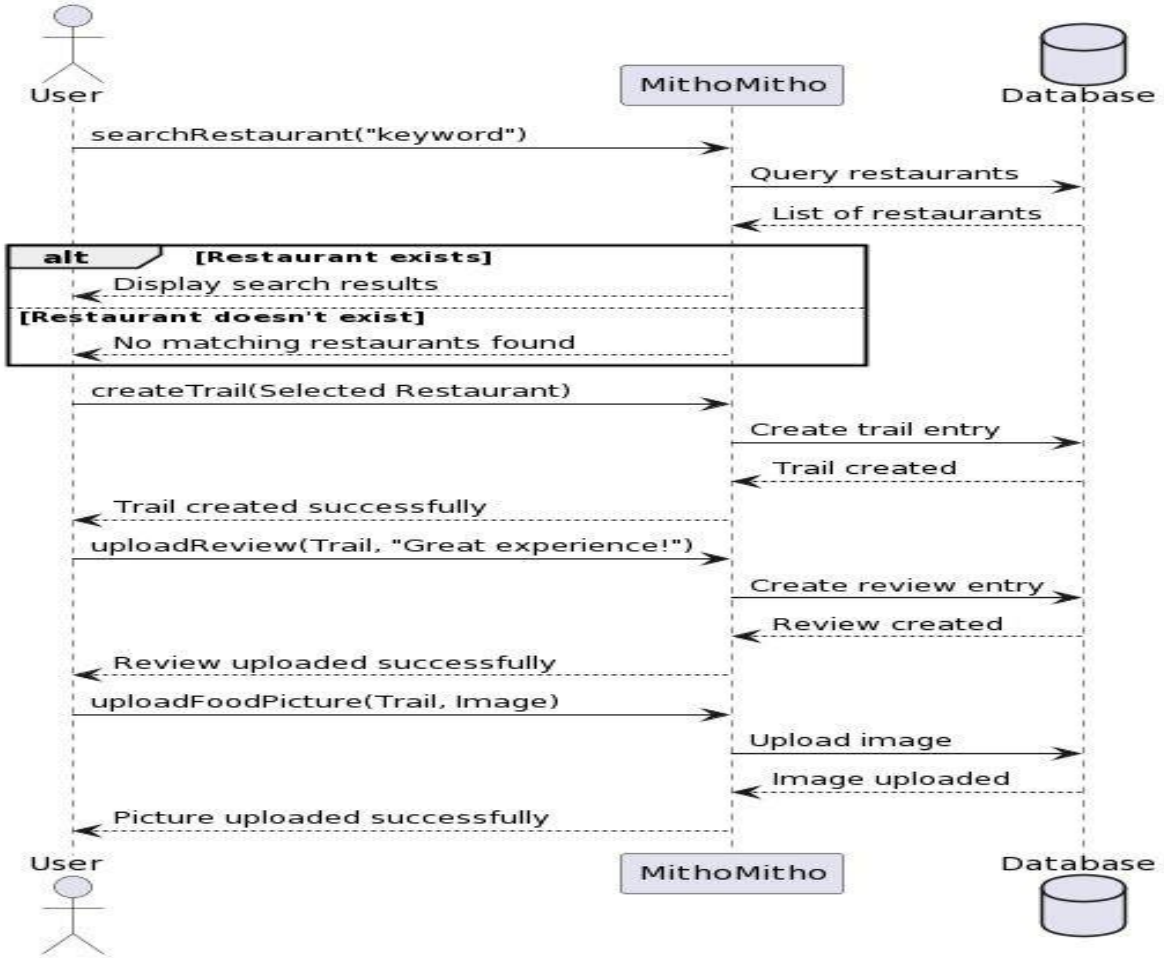
USER REGISTRATION AND LOGIN



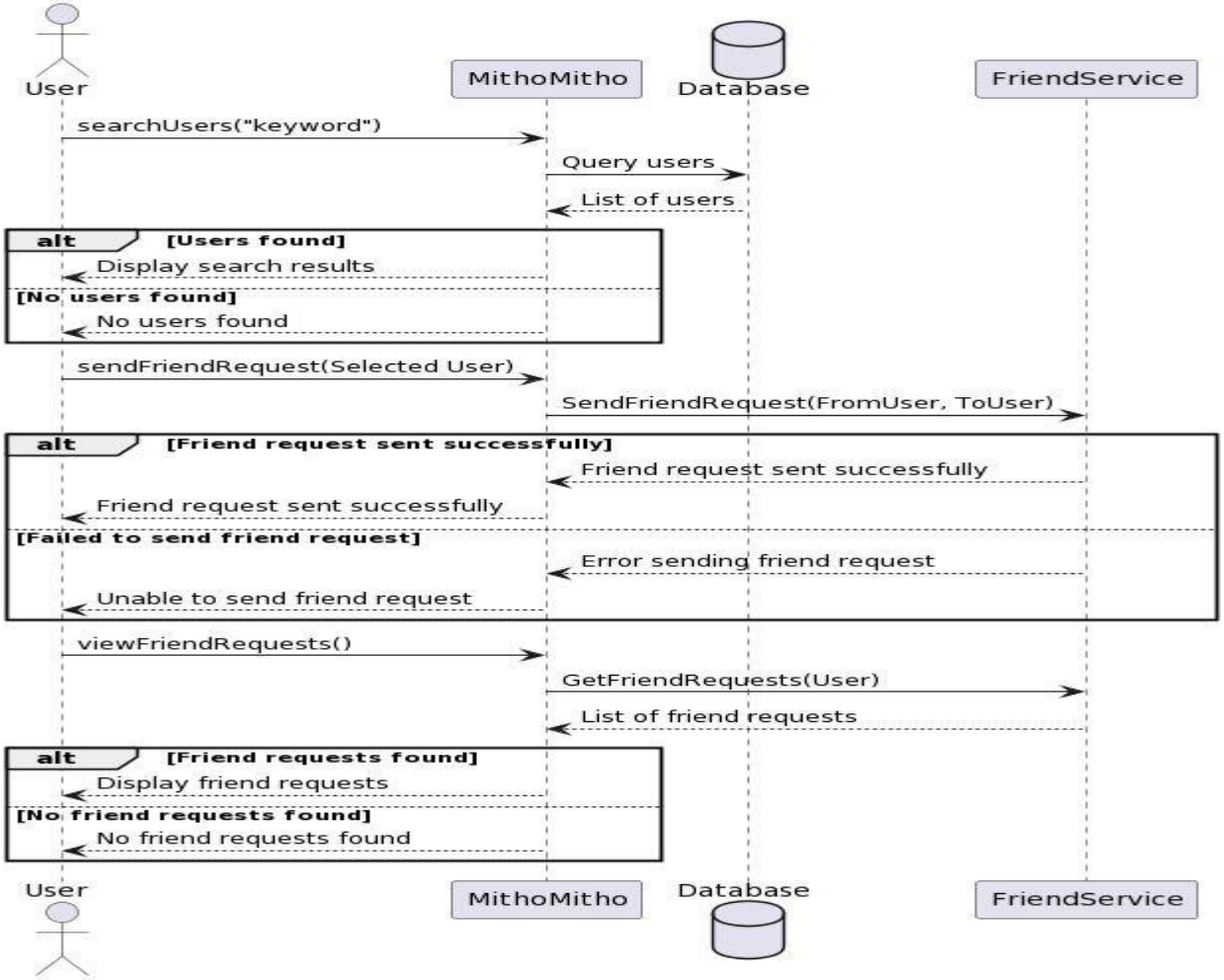
USER LOGOUT


User
LandingPage
AuthService
initiateLogout()
Logout()
Redirect to landing page
User
LandingPage
AuthService

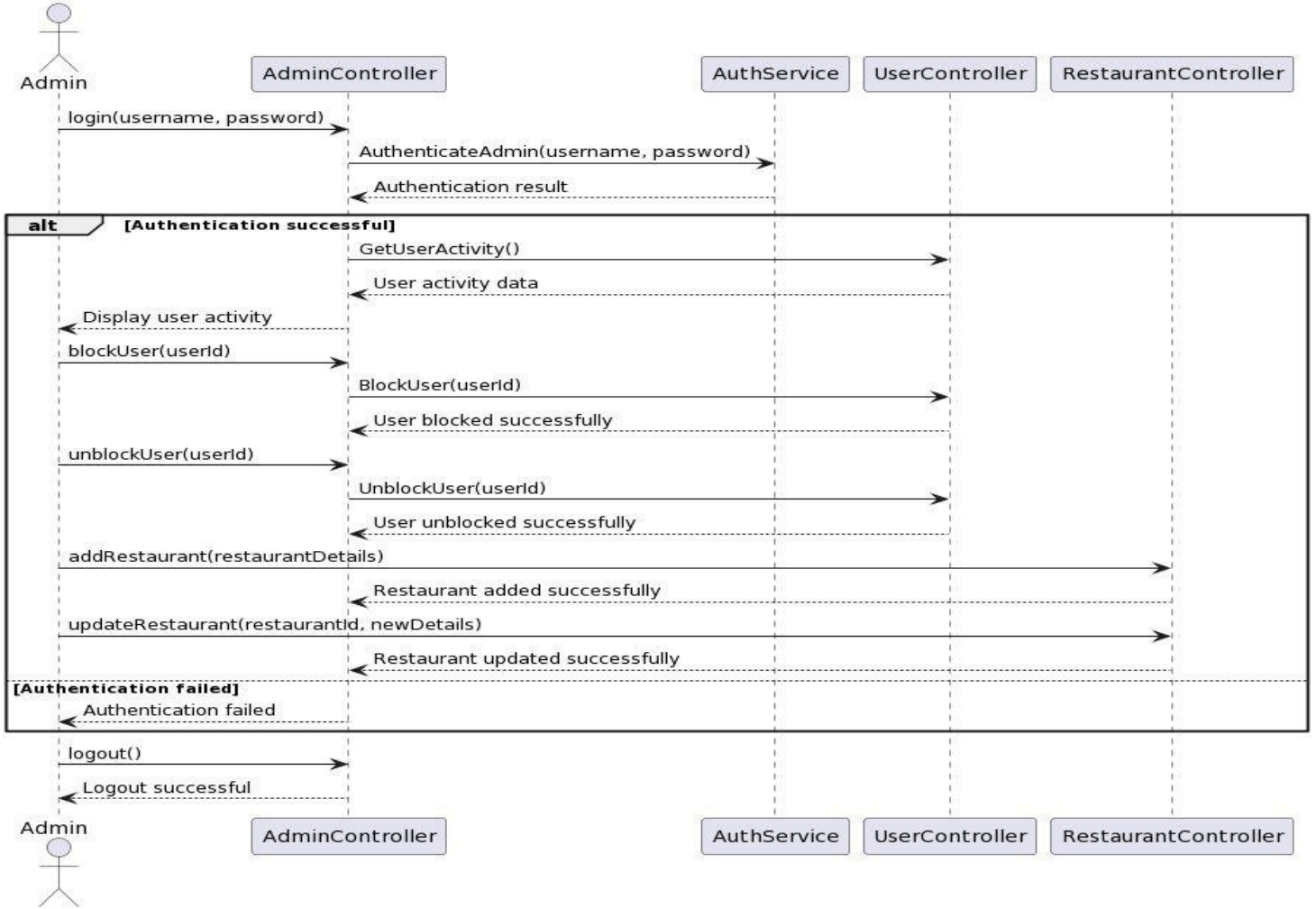
USER ACTIONS: RESTAURANTS AND REVIEW



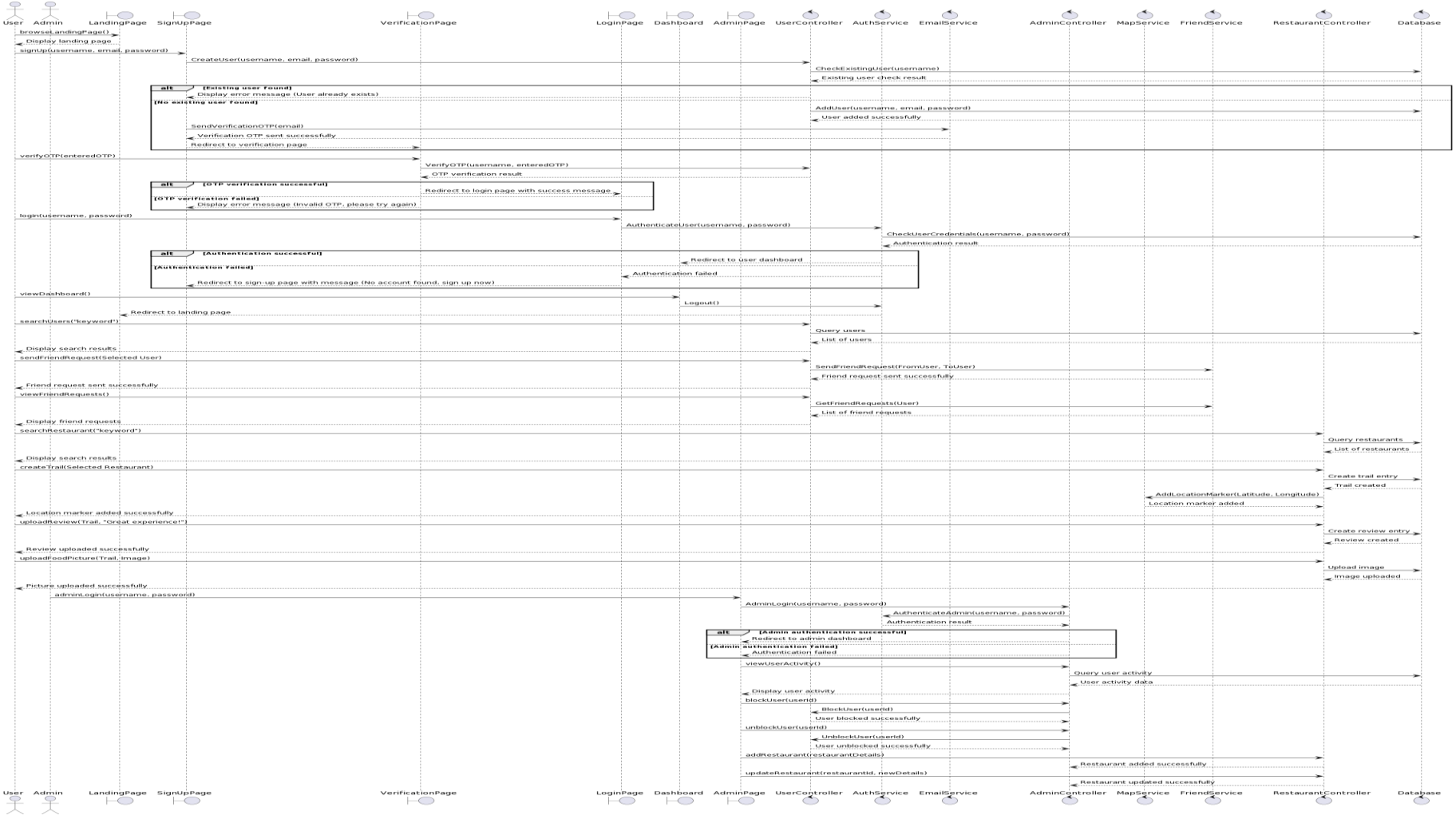
USER ACTIONS: CONNECTIONS



ADMIN ACTIONS



COMPLETE SYSTEM



Thank You