MILITARY UNIVERSITY OF TECHNOLOGY

im. Jarosława Dąbrowskiego

FACULTY OF CYBERNETICS

BUSINESS MODELING IN UML

SUBJECT PROJECT

Title: **“BICYCLE RENTAL SYSTEM”**

Group: **WCY23IV1S4**

Submitted by:	Supervisor:
inż. Jan BITKOWSKI **inż. Maciej KAWKA** **inż. Gabriela PAWŁOWSKA** **inż. Radosław RELIDZYŃSKI** **inż. Ignacy WYSOKIŃSKI**	**dr inż. Robert WASZKOWSKI**

Warszawa 2025

Table of contents

Lab Project

- **Implementation of project tasks in subgroups**
- **4-5 people per group**
- **Unique project topics**
- **Order of submission based on the "first come, first served" principle**
- **Evaluation – during the last labs**
- **Topic examples:**
  - Bicycle rental
  - Warehouse
  - Human Resources Department
  - Restaurant
  - Library

Laboratory Task (1/3)

- **Using any UML modeling tools, prepare:**
  - Requirements documentation
  - Analytical documentation
  - Design documentation

Laboratory Task (2/3)

- **The laboratory report should include documentation for the stages of IT system development, including:**
  - **Requirements documentation:**
    a) Use case diagrams
    b) Specification of non-functional requirements
  - **Analytical documentation, including:**
    a) Activity diagrams
    b) State diagrams
    c) Class diagrams
  - **Design documentation, including:**
    a) Object diagrams
    b) Sequence or collaboration diagrams

Laboratory Task (3/3)

- **Evaluation: last scheduled class**

Requirements documentation

Use case diagrams

This Use Case Diagram for the “Bike Rental System” illustrates the interactions between different system actors (users of the system) and the system’s functionalities (use cases).

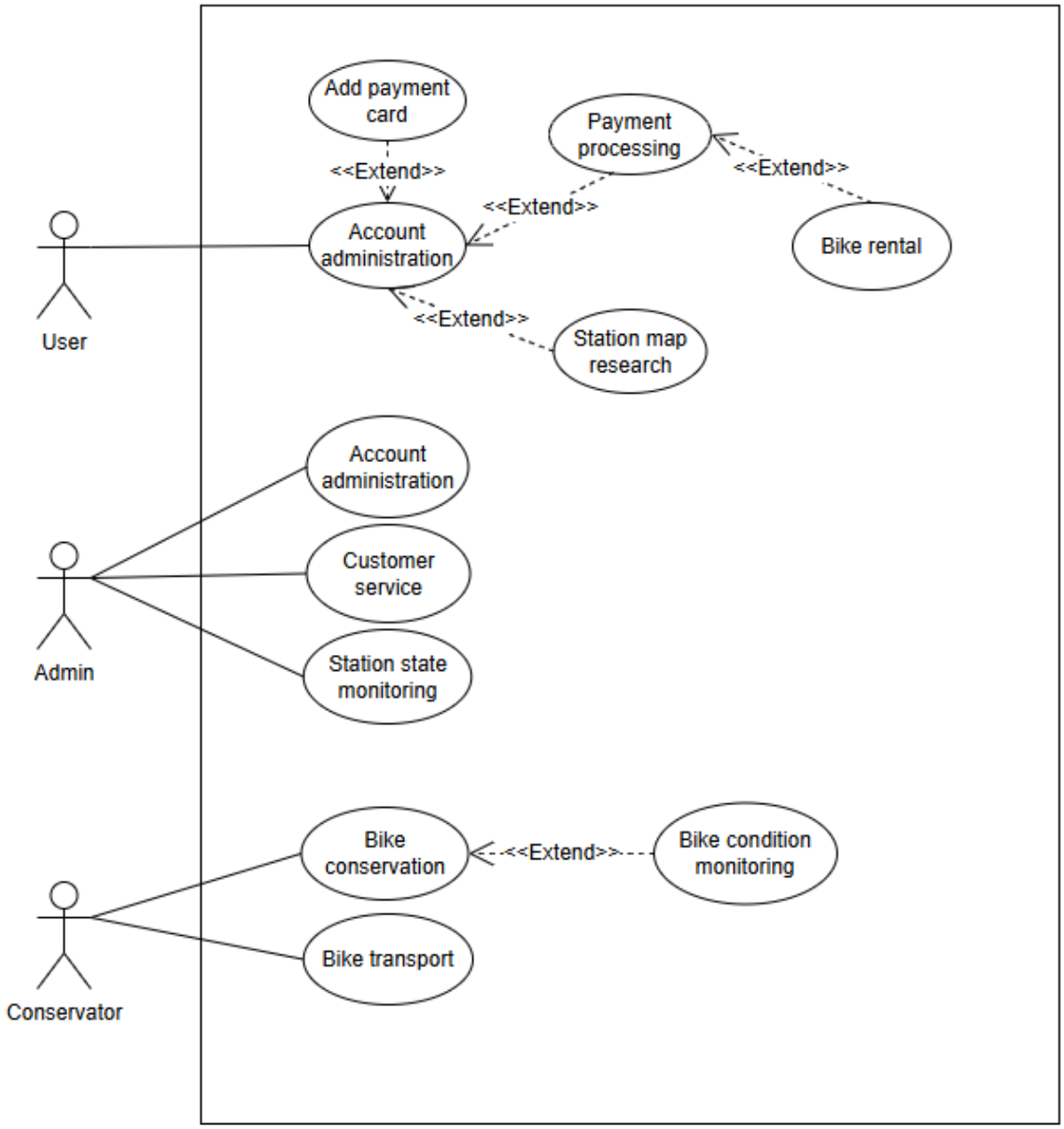


Fig. 1. Use case diagram

I.1.1. Actors

Actors are the entities that interact with the system. In this diagram, there are three actors:

1. **User**:
   - Represents a general user of the system, a customer who wants to rent a bike.
   - This actor’s use cases are related to renting bikes and managing their account.
2. **Admin**:
   - Represents an administrator of the system, responsible for managing the system's operations.

- This actor handles customer support, monitoring stations and oversees bike-related activities.

3. **Conservator**:
   - Represents a maintenance worker, technician responsible for the upkeep of the bikes.
   - This actor focuses on bike maintenance and transportation.

I.1.2. Use Cases

Use cases represent the functionalities or actions that the system provides to the actors. The diagram includes several use cases, some of which are extended by others.

Use Cases for the User

Account Administration

Description:

This use case allows the User to manage their account, such as creating, updating, or deleting their profile.

Preconditions:

- The User must have access to the bike rental system (e.g., via a mobile app or website).
- For creating an account, no prior account with the same credentials (e.g., email) must exist.
- For updating or deleting, the User must be logged into an existing account.

Basic Flow:

- The User accesses the account management section of the system (e.g., via app or website).
- The User selects an action: create, update, or delete their account.
  - If creating: The User enters required details (e.g., email, password, name).
  - If updating: The User modifies existing details (e.g., changes password or contact info).
  - If deleting: The User confirms the deletion of their account.
- The system validates the input (e.g., checks for duplicate email during creation).
- The system updates the account database accordingly.
- The system notifies the User of the successful action (e.g., “Account created successfully”).

Alternative Flow:

- A1: Invalid Input During Creation/Update
  - At step 3, if the User provides invalid data (e.g., an already-used email or weak password), the system displays an error message (e.g., “Email already in use”) and prompts the User to correct the input. The flow returns to step 2.
- A2: Failed Deletion Due to Active Rentals

- At step 2, if the User tries to delete their account but has an active bike rental, the system displays an error (e.g., “Cannot delete account with active rentals”) and prevents deletion until the rental is completed.

Extended by:

- *Add Payment Card*

  Description:

  The User can add a payment card to their account for transactions. This is an optional step (<<Extend>>) that enhances the account administration process.

  Preconditions:

  - The User must be logged into their account.
  - The User must have a valid payment card available to add.
  - The system must support the payment card type (e.g., Visa, MasterCard).

  Basic Flow:

  - During or after account administration, the User selects the option to add a payment card.
  - The User enters payment card details (e.g., card number, expiration date, CVV).
  - The system validates the card details through a payment gateway.
  - The system saves the card details to the User’s account.
  - The system confirms to the User that the card was added successfully.

  Alternative Flow:

  - A1: Invalid Card Details
    - At step 3, if the card details are invalid (e.g., incorrect number or expired card), the system displays an error (e.g., “Invalid card details”) and prompts the User to re-enter the information. The flow returns to step 2.
  - A2: Payment Gateway Failure
    - At step 3, if the payment gateway is unavailable, the system displays an error (e.g., “Payment service unavailable, try again later”) and the process ends.

- *Station Map Research*

  Description:

  The User can search for bike stations on a map, likely to find the nearest station to rent a bike. This is also an optional step (<<Extend>>) tied to account administration, possibly for convenience during account setup or usage.

  Preconditions:

  - The User must have access to the system’s map feature (e.g., via app or website).
  - The system must have up-to-date station location data available.

Basic Flow:

- During or after account administration, the User selects the option to view a station map.
- The system displays a map with nearby bike stations, using the User's current location (if location services are enabled) or a default area.
- The User browses the map to find a station and views details (e.g., number of available bikes).
- The User exits the map view or proceeds to rent a bike from a selected station.

Alternative Flow:

- A1: Location Services Disabled
  - At step 2, if the User's location services are disabled, the system prompts the User to enable them or manually enter a location to display nearby stations.
- A2: No Stations Available
  - At step 2, if no stations are available in the area, the system displays a message (e.g., "No stations found nearby") and suggests expanding the search radius.

- *Payment Processing*

Description:

This use case involves handling payments for bike rentals, such as charging the User for the rental duration.

Preconditions:

- The User must have an active account with a valid payment method linked.
- The User must have initiated a bike rental transaction requiring payment.
- The system must be connected to a payment gateway.

Basic Flow:

- The User completes a bike rental session (e.g., returns the bike or rental time ends).
- The system calculates the rental fee based on duration and pricing rules.
- The system charges the fee to the User's linked payment method via the payment gateway.
- The payment gateway confirms the transaction.
- The system updates the User's rental history and notifies the User of the successful payment (e.g., "Payment of $5.00 completed").

Alternative Flow:

- A1: Payment Declined

- At step 3, if the payment is declined (e.g., insufficient funds), the system notifies the User (e.g., “Payment failed: Insufficient funds”) and prompts them to add a new payment method. The flow may redirect to the “Add Payment Card” use case.

A2: Payment Gateway Unavailable

- At step 3, if the payment gateway is down, the system logs the pending payment and notifies the User (e.g., “Payment processing delayed, you will be charged later”). The system retries the payment later.

Extended by:

- Bike Rental (See below for details as a standalone use case).

Bike Rental

Description:

This is the core functionality for the User, allowing them to rent a bike from the system.

Preconditions:

- The User must be logged into their account.
- A payment method must be linked to the account (successfully processed via Payment Processing).
- At least one bike must be available at a station accessible to the User.
- The User must be physically present at or near a bike station (if manual pickup is required) or have access to the system for unlocking a bike remotely.

Basic Flow:

- The User selects a bike station (possibly via Station Map Research).
- The system displays available bikes at the station.
- The User selects a bike and initiates the rental process (e.g., by scanning a QR code or clicking “Rent”).
- The system unlocks the bike (e.g., via a smart lock) and starts the rental timer.
- The User uses the bike for the desired duration.
- The User returns the bike to a station, and the system locks the bike and ends the rental timer.
- The system triggers the Payment Processing use case to charge the User for the rental.

Alternative Flow:

- A1: No Bikes Available
  - At step 2, if no bikes are available at the selected station, the system notifies the User (e.g., “No bikes available at this station”) and suggests finding another station (may redirect to Station Map Research).
- A2: Bike Fails to Unlock

- At step 4, if the bike fails to unlock (e.g., due to a technical issue), the system notifies the User (e.g., “Bike unlock failed, please try another bike”) and logs the issue for the Admin (may trigger Customer Service).

Use Cases for the Admin

Account Administration

Description:

Similar to the User, the Admin can also manage accounts, but likely with higher privileges (e.g., managing all user accounts, resolving issues, etc.).

Preconditions:

- The Admin must be authenticated and logged into the system with administrative privileges.
- The system must have a database of user accounts available for management.

Basic Flow:

- The Admin accesses the account management dashboard.
- The Admin selects a user account to manage (e.g., by searching for a user’s email or ID).
- The Admin performs an action: view, update, or delete the account.
- If updating: The Admin modifies details (e.g., resets password, updates user status).
- If deleting: The Admin confirms the deletion of the account.
- The system updates the account database.
- The system notifies the Admin of the successful action (e.g., “Account updated successfully”).

Alternative Flow:

- A1: Account Not Found
  - At step 2, if the user account is not found, the system displays an error (e.g., “User not found”) and prompts the Admin to verify the search criteria.
- A2: Deletion Fails Due to Active Rentals
  - At step 3, if the account has active rentals, the system prevents deletion and notifies the Admin (e.g., “Cannot delete account with active rentals”).

Customer Service

Description:

The Admin provides customer support, such as answering queries, resolving complaints, or assisting Users with issues related to bike rentals.

Preconditions:

- The Admin must be logged into the system with access to customer support tools (e.g., ticketing system, chat interface).
- A User must have submitted a query, complaint, or request for assistance.

Basic Flow:

- The Admin accesses the customer support dashboard.
- The Admin views a list of open support tickets or queries.
- The Admin selects a ticket to address (e.g., a User reports a bike that failed to unlock).
- The Admin investigates the issue (e.g., checks system logs, contacts the User for more details).
- The Admin resolves the issue (e.g., refunds the User, marks the bike for repair).
- The system updates the ticket status to "Resolved" and notifies the User of the resolution.

Alternative Flow:

- A1: Insufficient Information
  - At step 4, if the User's query lacks sufficient details, the Admin contacts the User for more information, and the ticket remains open until the User responds.
- A2: Issue Cannot Be Resolved Immediately
  - At step 5, if the issue requires further action (e.g., bike repair), the Admin escalates the issue to the Conservator (may trigger Bike Conservation) and informs the User of the delay.

Station State Monitoring

Description:

The Admin monitors the state of bike stations, such as checking the availability of bikes, station functionality, or any operational issues.

Preconditions:

- The Admin must be logged into the system with monitoring privileges.
- Bike stations must be equipped with sensors or reporting mechanisms to send real-time data to the system.
- The system must have an active connection to station data feeds.

Basic Flow:

- The Admin accesses the station monitoring dashboard.
- The system displays real-time data for all stations (e.g., number of bikes, station status).
- The Admin selects a station to view detailed information (e.g., bike availability, any reported issues).
- The Admin takes action if needed (e.g., flags a station with no bikes for rebalancing).
- The system logs the Admin's actions and updates the station status if necessary.

Alternative Flow:

- A1: Data Feed Unavailable
  - At step 2, if the station data feed is unavailable, the system displays an error (e.g., "Station data unavailable") and suggests the Admin try again later or check the connection.

- A2: Critical Issue Detected
  - At step 3, if a critical issue is detected (e.g., station offline), the Admin escalates the issue to technical support and notifies the Conservator for physical inspection.

Use Cases for the Conservator

Bike Conservation

Description:

This use case involves the Conservator maintaining the bikes, such as repairing or servicing them to ensure they are in good condition.

Preconditions:

- The Conservator must be authenticated and logged into the system with conservator privileges.
- At least one bike must be identified as needing maintenance or repair.
- The Conservator must have access to necessary tools and parts for bike servicing.

Basic Flow:

- The Conservator accesses the bike maintenance dashboard.
- The system displays a list of bikes flagged for maintenance (e.g., via Bike Condition Monitoring).
- The Conservator selects a bike to service.
- The Conservator performs the necessary repairs or maintenance (e.g., fixes a flat tire, lubricates the chain).
- The Conservator updates the bike's status in the system (e.g., marks it as "Ready for Use").
- The system logs the maintenance activity and makes the bike available for rental again.

Alternative Flow:

- A1: Missing Parts or Tools
  - At step 4, if the Conservator lacks the necessary parts (e.g., a replacement tire), the Conservator orders the parts, and the bike remains out of service until the parts arrive.
- A2: Bike Beyond Repair
  - At step 4, if the bike is beyond repair, the Conservator marks it as "Decommissioned" in the system, and the bike is removed from the rental pool.

Extended by:

- Bike Condition Monitoring

  Description:

The Conservator monitors the condition of the bikes, likely to identify which bikes need repairs or maintenance. This is an optional step (<<Extend>>) that supports the bike conservation process.

Preconditions:

- The Conservator must be logged into the system with access to bike condition data.
- Bikes must be equipped with sensors or reporting features (if automated) or have been manually inspected and reported.

Basic Flow:

- During or before Bike Conservation, the Conservator accesses the bike condition monitoring tool.
- The system displays condition data for bikes (e.g., sensor data showing low tire pressure, or user reports of issues).
- The Conservator identifies bikes needing maintenance and flags them for repair.
- The Conservator proceeds to the Bike Conservation use case to service the flagged bikes.

Alternative Flow:

- A1: No Issues Detected
    - At step 2, if no bikes are flagged for maintenance, the system displays a message (e.g., “All bikes in good condition”), and the Conservator exits the monitoring tool.
- A2: Conflicting Reports
    - At step 2, if sensor data and user reports conflict (e.g., sensor says the bike is fine, but a user reported a broken pedal), the Conservator schedules a manual inspection to verify the bike’s condition.

Bike Transport

Description:

The Conservator is responsible for transporting bikes, possibly between stations or to a repair facility.

Preconditions:

- The Conservator must be authenticated and logged into the system.
- At least one bike must require transport (e.g., due to maintenance needs or station rebalancing).
- The Conservator must have access to transportation resources (e.g., a vehicle).

- Source and destination locations (e.g., stations or repair facilities) must be defined.

Basic Flow:

- The Conservator accesses the bike transport dashboard.
- The system displays a list of bikes needing transport (e.g., for repair or station rebalancing).
- The Conservator selects the bikes and confirms the source and destination locations.
- The Conservator physically transports the bikes to the destination (e.g., loads bikes into a van and drives to the repair facility).
- The Conservator updates the system to confirm the bikes have been delivered.
- The system updates the bike locations and availability status.

Alternative Flow:

- A1: Transport Vehicle Unavailable
  - At step 4, if no vehicle is available, the Conservator reschedules the transport and updates the system with the delay.
- A2: Destination Station Full
- At step 4, if the destination station has no available slots for bikes, the Conservator selects an alternative station and updates the system accordingly.

I.1.3. Relationships

I.1.4. **Association:** Solid lines between actors and use cases indicate that the actor directly interacts with the use case (e.g., User with "Account administration").

I.1.5. **Extend:** Dashed arrows labeled <<Extend>> show that one use case can optionally extend another. For example, "Add payment card" extends "Account administration," meaning adding a payment card is an optional part of account administration.

Summary

- The **User** focuses on renting bikes, managing their account, and making payments. They can also search for stations and add payment methods as part of their account management.
- The **Admin** oversees the system by managing accounts, providing customer service, and monitoring station states to ensure smooth operation.
- The **Conservator** ensures the bikes are in good condition by maintaining and transporting them, with a focus on monitoring their condition as needed.

I.2. Specification of non-functional requirements

Performance:

- Average customer service response time: 30 seconds
- Average application loading time: 3–4 seconds

Size:

- Required disk space: 70 MB
- Maximum number of records in the database: 50 million

Usability:

- Maximum time required to train users: 5 minutes

Portability:

- Number of target platforms: 4 (iOS, Android, HarmonyOS, Windows)

Reliability:

- Availability (percentage of time the system is operational): 99.8%
- Data is stored on backup servers

Analytical documentation

Activity diagrams

Account administration - User

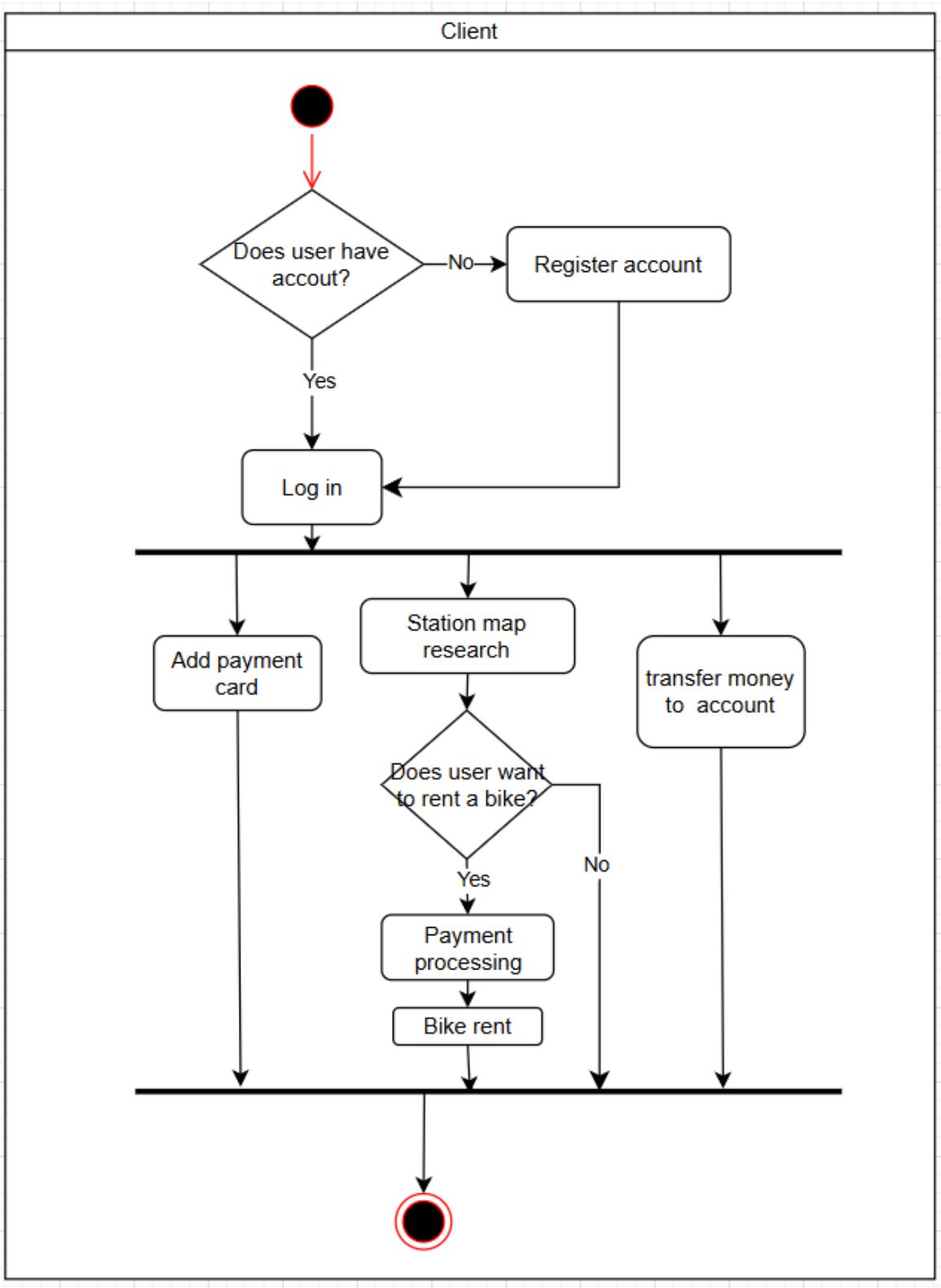
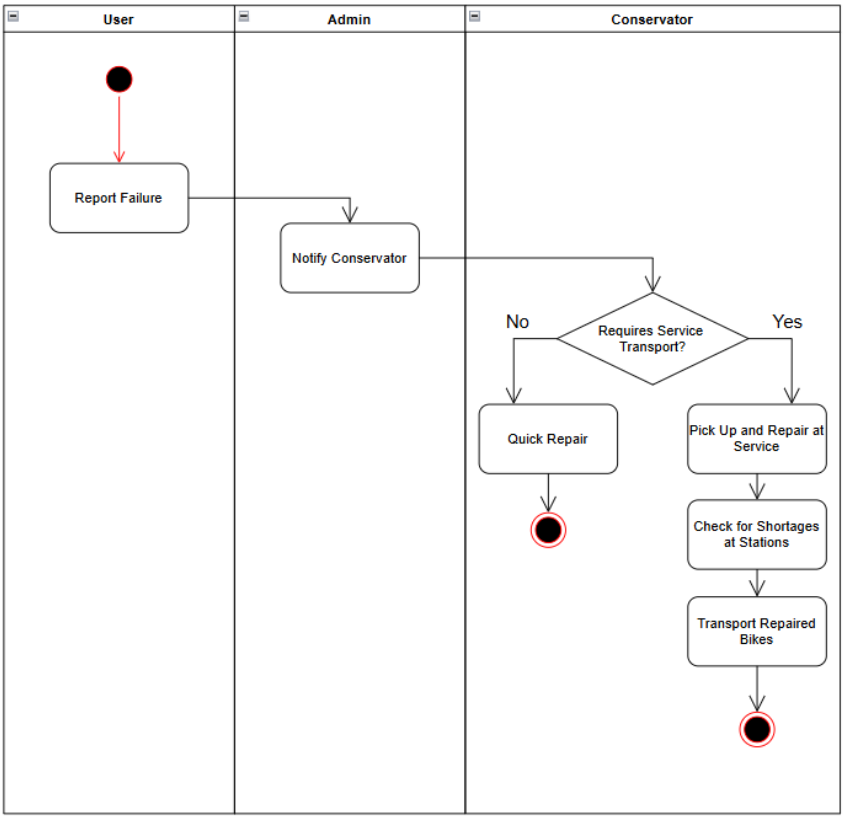


Fig. 2. Account administration for user use case activity diagram

Bike condition monitoring



The diagram illustrates the process of bike repair within a rental system, beginning with a user reporting a failure, which triggers an admin notification to the conservator; the conservator evaluates the need for service transport, deciding between a quick on-site repair or sending the bike to a service center; post-repair, the conservator checks bike availability at stations and transports the repaired bikes back.

State diagrams

This diagram represents all possible states of bike during renting or conservation process.

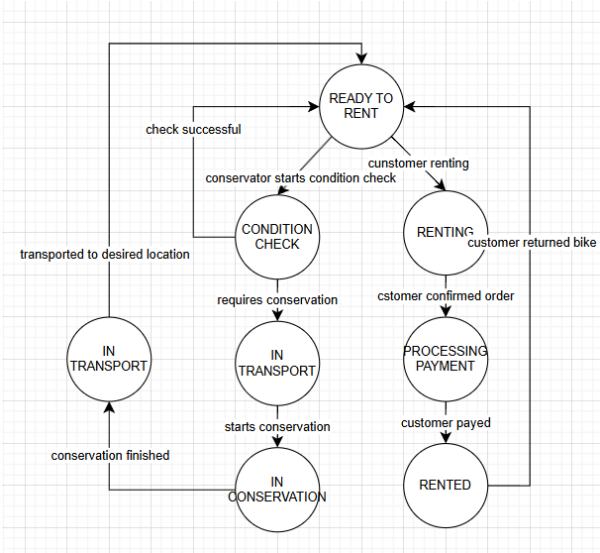


Fig. 3. State diagram

States description:

- READY TO RENT – initial state for ordering or conservation process, state when bike is ready
- RENTING – happens when client stars renting process
- PROCESSING PAYMENT – mid-state to finish renting process for client to pay for service
- RENTED – client successfully rented a bike and can return it to make it READY TO RENT back
- CONDITON CHECK – for converator to start checking of bike condition
- IN TRANSPORT – when bike needs some conservation and needs to e transported to conservation station, and then back to renting station
- IN CONSERVATION – conservation state to apply important fixes to the bike

Class diagrams

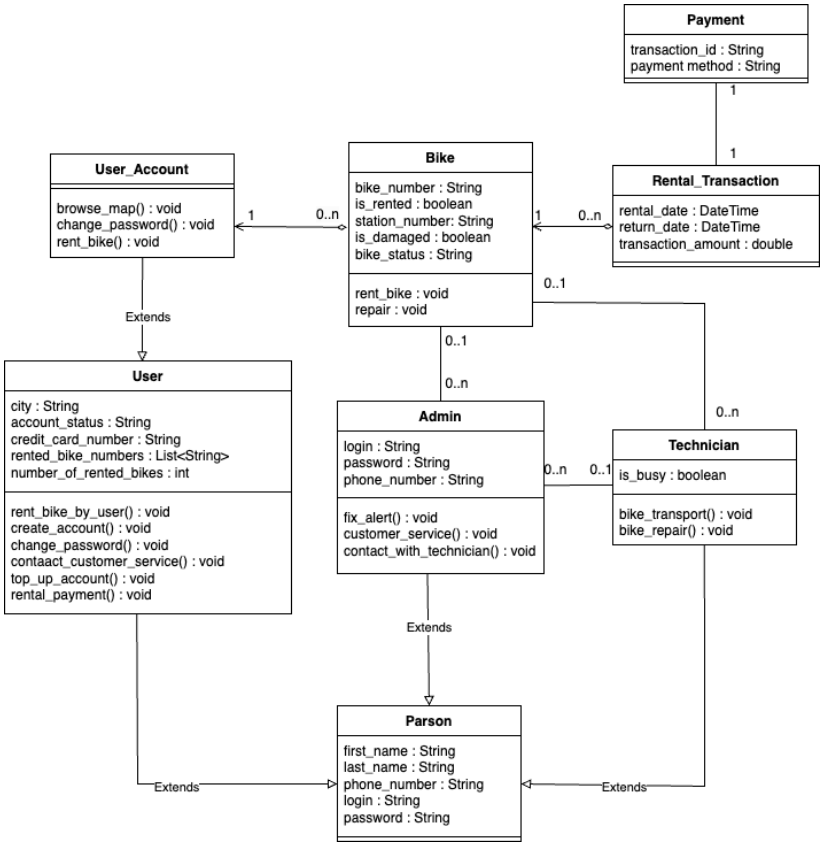


Fig. 4. Class diagram

Classes descriptions:

- User_Accpount – Represents the basic functionalities available to any user account, such as browsing the map, changing the password, and renting a bike. It serves as a base class for end users.
- Bike – Represents a single bike available in the system, with information about its condition, availability, and assigned station. It can be rented, reported as damaged, or repaired by a technician.
- Payment – Represents the payment for a bike rental, including the transaction number and the method of payment. Linked to one rental transaction.
- Rental_Transaction – Describes a single bike rental, including the rental and return dates as well as the transaction amount. Linked to a specific bike and payment.
- User – A person using the bike rental system, who has an account, contact details, and the ability to rent bikes. The user can top up their account, report issues, and contact customer support.
- Admin – A system employee responsible for customer support, assigning tasks to technicians, and reporting bike malfunctions. Manages technicians and oversees the condition of the bike infrastructure.
- Technician – A technical staff member responsible for transporting and repairing bikes. Works with the admin and carries out assigned maintenance tasks.

- Person – A base class containing common personal data such as first name, last name, login, and phone number. Inherited by User, Admin, and Technician.

Design documentation

Object diagrams

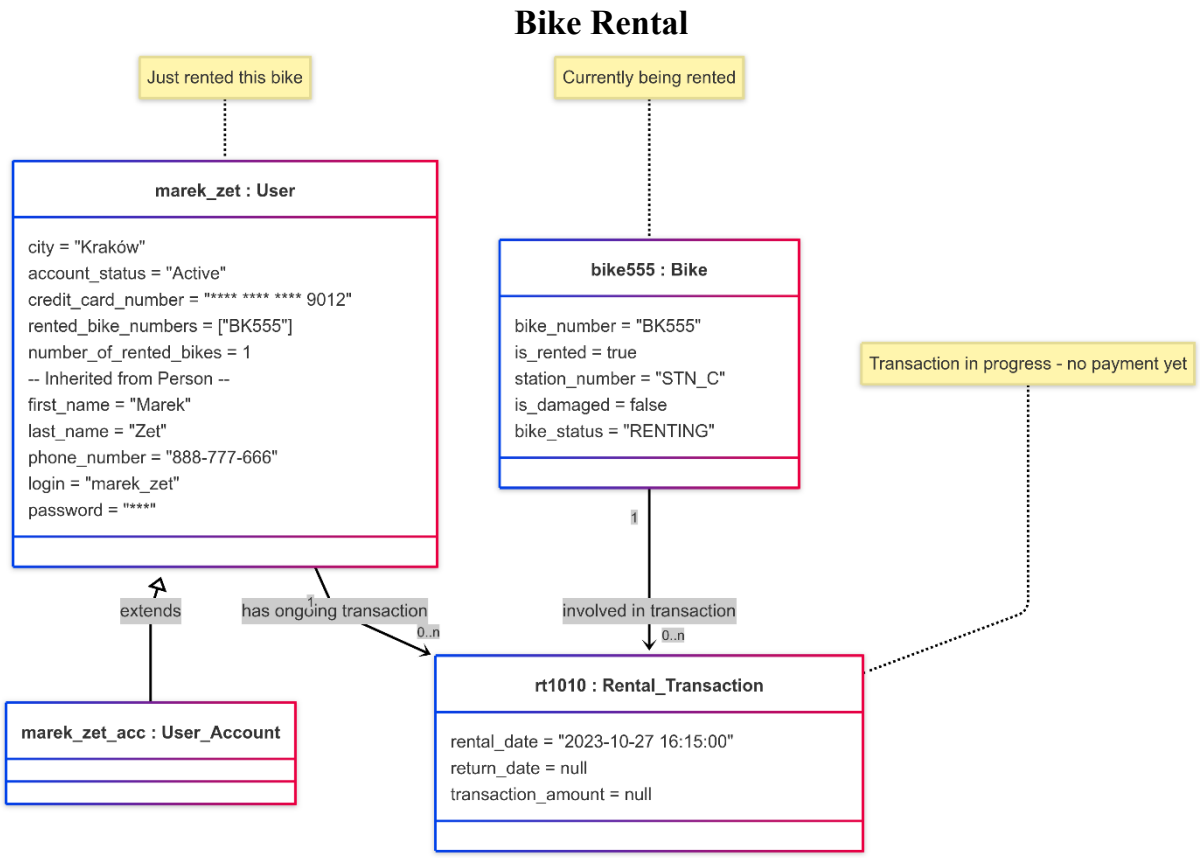


Fig. 5. Bike rental object diagram

Key Objects in the Diagram:

1. **User objects**:
    - marek_zet_acc : User_Account - Marek's basic user account
    - marek_zet : User - Marek's user profile showing he has rented bike BK555, with his personal data inherited from the Person class
2. **Bike object**:
    - bike555 : Bike - The bike that Marek has just rented
    - The bike's status is set to "RENTING" and is_rented = true
    - The bike was rented from station "STN_C"
3. **Transaction object**:
    - rt1010 : Rental_Transaction - An active rental transaction
    - Has a rental start time but no return time (null)
    - Transaction amount is null because the final cost isn't determined until the bike is returned
4. **Relationships**:
    - Inheritance showing User extends User_Account

- Associations connecting the user to the active transaction
- Association showing the bike involved in the transaction

5. **Notable aspects**:
    - No Payment object exists at this stage of the rental process
    - The transaction is ongoing with an undetermined final amount
    - The user's rented_bike_numbers array contains the ID of the currently rented bike

Bike Return

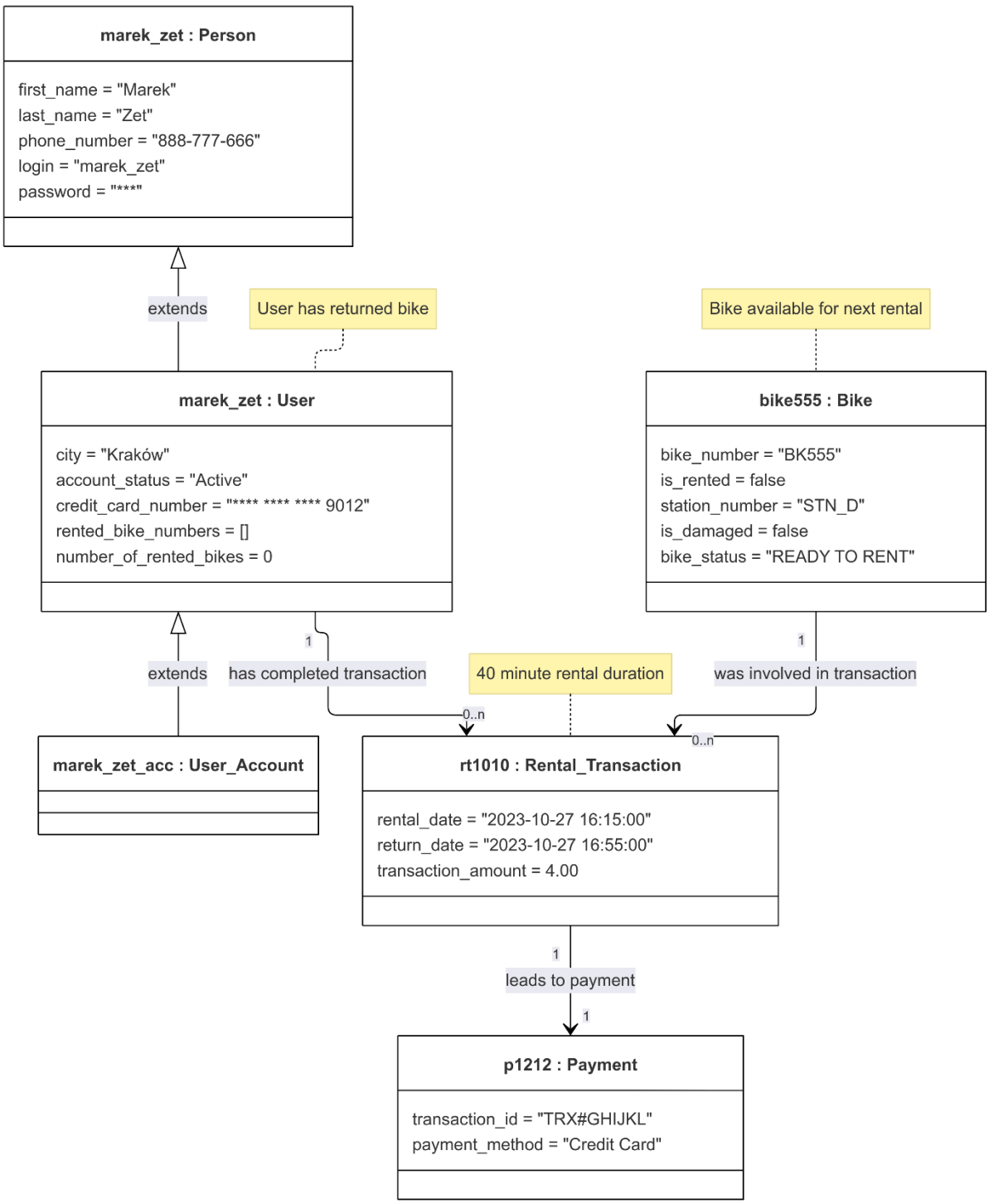


Fig. 6. Bike return object diagram

Key Objects in the Diagram:

User objects:

- marek_zet_acc : User_Account - Marek's basic user account
- marek_zet : Person - Marek's personal data
- marek_zet : User - Marek's user profile (with an empty array of rented bikes)

Bike object:

- bike555 : Bike - a bike that has been returned
- The bike's status indicates it's no longer rented and is ready for rental again

Transaction objects:

- rt1010 : Rental_Transaction - the completed rental transaction
- p1212 : Payment - the payment for the rental

Relationships:

- Inheritance between Person, User, and User_Account objects
- Associations showing the connections between user, bike, transaction, and payment

Sequence of cooperation diagrams

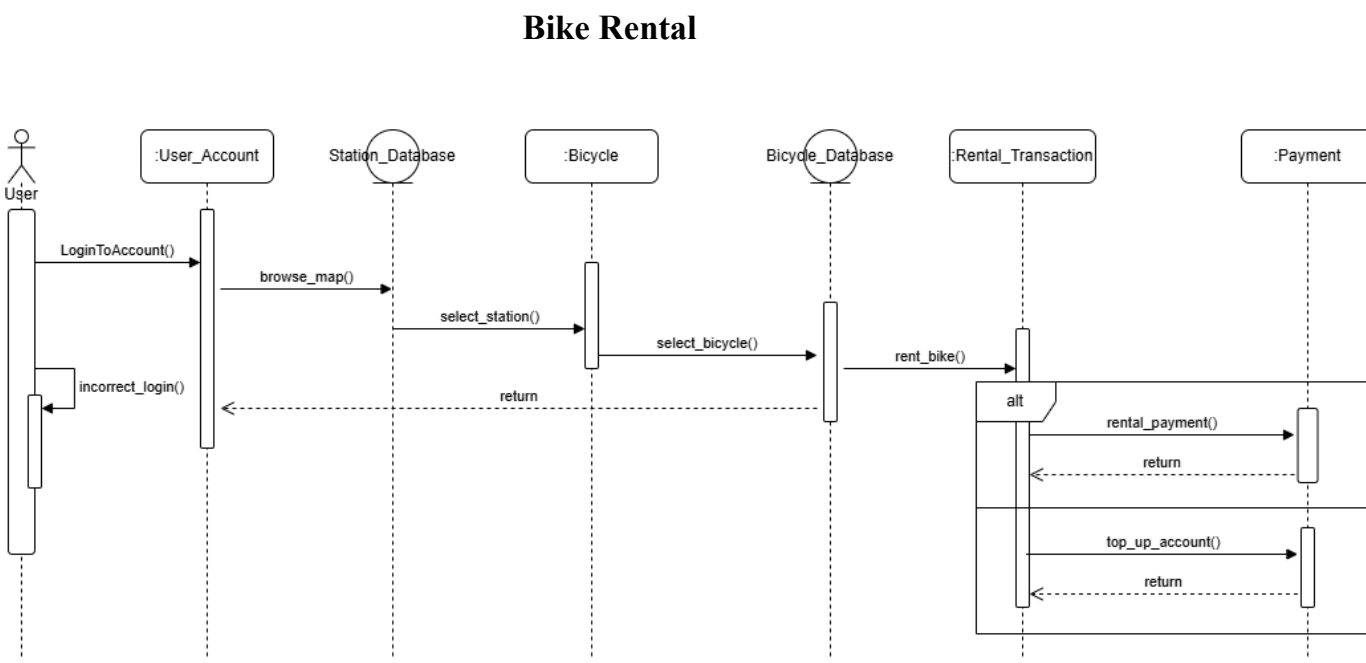


Fig. 7. Bike rental sequence diagram

This sequence diagram illustrates the bike rental process flow in the system. Here's a step-by-step description:

1. The process begins with the user logging into their account via LoginToAccount().
2. Once logged in, the user can browse the map using the browse_map() function from the User_Account, which interacts with the Station_Database.
3. The user selects a station with select_station().
4. At the chosen station, the user selects a specific bicycle using select_bicycle(), which communicates with the Bicycle_Database.
5. After selecting a bicycle, the actual rental is initiated with rent_bike(), creating a Rental_Transaction.
6. The alternative flow (shown by the "alt" section) indicates there are two potential paths:
    - The primary path involves making a payment for the rental with rental_payment(), which communicates with the Payment system
    - There appears to be an option to top up the account if needed with top_up_account()
7. Each operation returns a response back through the chain of interactions.

Identified objects:

- Client application (represented by User_Account)
- Server (represented by the Station_Database, Bicycle_Database, Rental_Transaction, and Payment components)

Identified actors:

- User (the person interacting with the system, shown as the stick figure on the left)

Bike Return

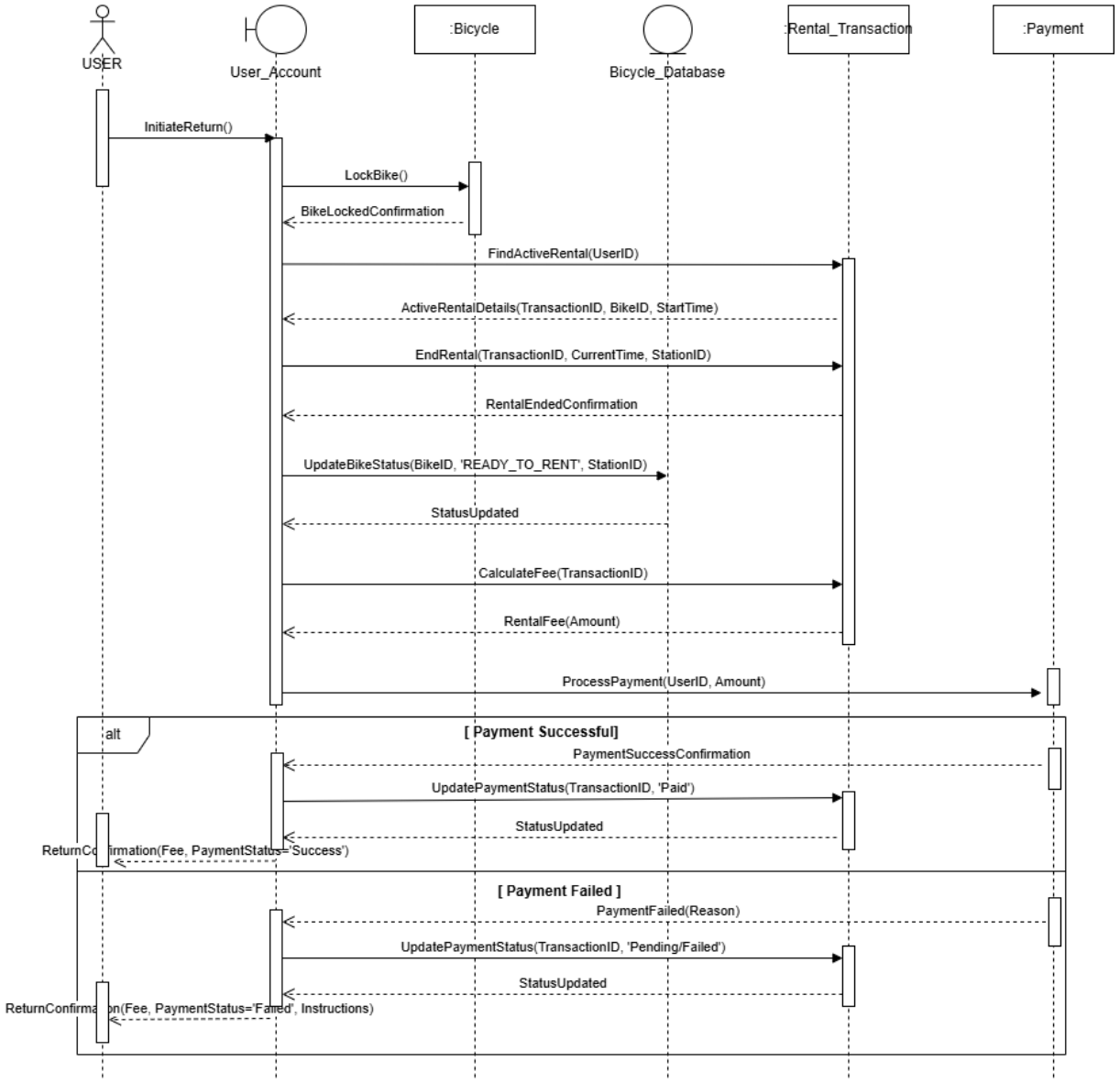


Fig. 8. Bike return sequence diagram

Looking at this sequence diagram, it illustrates the complete bike return process in the bike rental system. Here's a step-by-step description:

1. The process begins when the User initiates a bike return by calling InitiateReturn() to the User_Account.
2. The User_Account then requests LockBike() from the Bicycle object, which confirms with a BikeLockConfirmation response.
3. Next, User_Account calls FindActiveRental(UserID) on the Rental_Transaction to locate the ongoing rental associated with the user.

4. The Rental_Transaction responds with ActiveRentalDetails including transaction ID, bike ID, and rental start time.
5. User_Account then calls EndRental(TransactionID, CurrentTime, StationID) to terminate the rental, receiving a RentalEndedConfirmation.
6. The bike's status is updated with UpdateBikeStatus(BikeID, 'READY_TO_RENT', StationID) and confirmed with StatusUpdated.
7. The rental fee is calculated using CalculateFee(TransactionID), with the amount returned in RentalFee(Amount).
8. Payment processing begins with ProcessPayment(UserID, Amount) sent to the Payment system.
9. The diagram shows an alternative flow (marked by "alt"):
    - If payment is successful:
        - Payment returns PaymentSuccessConfirmation
        - Rental_Transaction status is updated to "Paid"
        - User receives ReturnConfirmation with fee and success status
    - If payment fails:
        - Payment returns PaymentFailed(Reason)
        - Rental_Transaction status is updated to "Pending/Failed"
        - User receives ReturnConfirmation with fee, failed status, and instructions