Book&Dock

Software Engineering - 2024

Authors:

Andrzej Babulewicz

Marta Dawidowska

Małgorzata Dżaman

Jan Kędziorek

Serhat Seval

Contents

1 Project Overview

1.1 Introduction

We are developing a mobile application designed to connect sailors and dock owners in the Mazurian Lakes region. The primary purpose of this application is to enable dock owners to advertise their docking facilities, while sailors can conveniently book a spot to stay overnight.

Our application addresses a real problem that arises during the summer season in Mazury. A significant number of sailors often struggle to find available spots in docks, as many docks become overcrowded. When a dock is full, locating another dock with free spaces can be both time-consuming and challenging, and in some cases, it may not even be possible.

Our app offers a solution by allowing sailors to select a dock where they wish to stay, reserve a spot in advance, and avoid the uncertainty of arriving too late to secure a place. This feature ensures that sailors can enjoy their trips without worrying about finding an overnight docking spot before sunset.

In addition to facilitating dock reservations, our app also includes travel guides written by Editors. These guides provide useful tips and recommendations for sailors traveling across Mazury, enhancing their experience and helping them make the most of their journeys.

2 System Requirements

The system represents a platform designed to facilitate interactions among sailors, dock owners, editors, and administrators. These people communicate through the application to fulfill their respective goals. Sailors use the platform to book docking spaces, explore guides, and share reviews. Dock owners manage docking availability, advertise their services, and receive bookings. Editors contribute tips, guides, and other content to help users explore and utilize the system effectively. Administrators oversee user activity, ensure compliance, and maintain smooth platform operations. By interacting with one another through the application, these users aim to create a seamless and enjoyable experience for all involved.

3 Functional Requirements

3.1 User Stories

1. **As a sailor, I want to book a space in a dock so that I have a place to dock my boat for the night.**
2. **As a sailor, I want to read a guide so that I can explore Mazury better.**
3. **As a dock owner, I want to advertise my port so that sailors can easily discover and utilize my port's services.**
4. **As an editor, I want to provide tips and guides so that sailors can learn about and explore new places.**
5. **As an admin, I want to manage users and monitor activity so that safety and compliance are ensured, and the platform operates smoothly.**

3.2 Functional Requirements

- Sailor can see the availability of docks.
- Sailor can make and cancel a reservation.
- Sailor can access the contact information of the port's owner.
- Sailor can search for a place based on date, location, and price range.
- Sailor can securely pay through the platform.
- Sailor can learn about regulations from guides.
- Sailor can discover new places from guides.

- Sailor can view a map of docking locations outside of ports from guides.
- Dock Owner can assign and update the availability of docking spots in real time.
- Dock Owner can set prices for a stay in their port.
- Dock Owner can add services and functionalities that their port offers.
- Dock Owner can receive notifications when a booking is confirmed or canceled.
- Editor can add posts or articles to the platform.
- Editor can attach pictures and links to the articles.
- Editor can reply to comments and questions.
- Editor can edit or delete existing posts.
- Editor can add promotional content.
- Admin can add, edit, or remove users and their information.
- Admin can approve posts.

3.3 Non-functional Requirements

- Sailor can leave reviews about their stay.
- Sailor can filter offers according to their preferences.
- The booking process is intuitive and easy to complete within a few steps.
- Sailor can modify their booking details.
- Sailor can leave reviews on articles they have read.
- Sailor can comment about their experiences.
- Dock Owner can own more than one port.
- Dock Owner can set booking restrictions.
- Notifications are sent instantly when changes are made by Dock Owner.
- Users can comment or react to editor's posts.
- Editor's articles are readable and accessible on mobile devices.
- The system supports multiple admins.
- Actions that process personal data are safe and compliant with data protection regulations.

4 System Design

4.1 Architecture Design

Our system has two parts, mobile app for sailor and dock owners and web app for editors and admins. The mobile app is designed for sailors and dock owners to book a space in a dock and advertise their port, respectively. The web app is designed for editors to provide tips and guides and for admins to manage users and monitor activity..

In our system sailor can book a space that dock owner added to the system. Dock owners can publish a port and set prices for a stay either by night or per person. Dock owner can choose properties of the port like electricity, water, shower, toilet, etc. and also provide additional costs for these services. Payment can be done either through platform or in person. After the stay sailor can give a review about the port. Communication between sailor and dock owner is done through notification system. Docks are saved in a database and sailors can search for docks based on location, date, and price range. Sailors can see availability of docks and make and cancel a reservation. Sailors can also see contact information of the port's owner.

Editors can add guides to the platform using the web application. Guides can be enriched by multimedia content like pictures and links. Guides can include map. Sailors can access to these guides through the mobile application. Sailors can comment on the guides and editors can reply to these comments. Editors can also add promotional content to the platform. Guides are also saved in a database and sailors can access them through the mobile app.

Admin can manage users and monitor activity on the platform. Admin can add, edit or remove users, ports and their information. Admin is responsible for approving port advertisements, guides and comments.

Integration between mobile and web app is done through a RESTful API. The API is responsible for communication between the mobile and web app. The API is also responsible for communication between the web app and the database. Message schemas will be mentioned in the communication design section.

4.2 Database Design

The database design for our system is structured to support the functionalities required by sailors, dock owners, editors, and admins. The database consists of several tables, each representing a different entity in the system. Below is an overview of the main tables and their relationships:

- **Users**: This table stores information about all users, including sailors, dock owners, editors, and admins. Each user has a unique ID, name, email, password, and role (sailor, dock owner, editor, admin).
- **Ports**: This table stores information about ports managed by dock owners. Each port has a unique ID, name, location, description, and owner ID (foreign key referencing the Users table).
- **DockingSpots**: This table stores information about individual docking spots within a port. Each docking spot has a unique ID, port ID (foreign key referencing the Ports table), availability status, and price.
- **Bookings**: This table stores information about bookings made by sailors. Each booking has a unique ID, sailor ID (foreign key referencing the Users table), docking spot ID (foreign key referencing the DockingSpots table), start date, end date, and payment status.
- **Guides**: This table stores information about guides written by editors. Each guide has a unique ID, title, content, author ID (foreign key referencing the Users table), and publication date.
- **Comments**: This table stores comments made by users on guides. Each comment has a unique ID, guide ID (foreign key referencing the Guides table), user ID (foreign key referencing the Users table), content, and timestamp.
- **Reviews**: This table stores reviews given by sailors about ports. Each review has a unique ID, port ID (foreign key referencing the Ports table), sailor ID (foreign key referencing the Users table), rating, and content.

- **Notifications**: This table stores notifications sent to users. Each notification has a unique ID, user ID (foreign key referencing the Users table), message, and timestamp.

Table Descriptions:

Table Schema with Column Types and Nullability

- **Users**
  - **userID** (`int`, **NOT NULL**, Primary Key)
  - **name** (`string`, **NOT NULL**)
  - **surname** (`string`, **NOT NULL**)
  - **email** (`string`, **NOT NULL**, Unique)
  - **phone_number** (`string`, NULLABLE)
  - **password** (`string`, **NOT NULL**, hashed)
  - **role** (`string`, **NOT NULL**, Enum: User/Admin/Owner)
- **Port**
  - **portID** (`int`, **NOT NULL**, Primary Key)
  - **name** (`varchar`, **NOT NULL**)
  - **location** (`json`, **NOT NULL**, Contains `latitude`, `longitude`, and `town` as fields)
  - **description** (`text`, NULLABLE)
  - **ownerID** (`int`, **NOT NULL**, Foreign Key referencing `Users(userID)`)
  - **dockingSpots** (`json`, NULLABLE, Contains availability, price per night, and price per person details for each spot)
  - **services** (`json`, NULLABLE, Stores a list of provided services like electricity, water, shower)
  - **isApproved** (`boolean`, **NOT NULL**)
- **DockingSpots**
  - **dockID** (`int`, **NOT NULL**, Primary Key)
  - **name** (`string`, **NOT NULL**)
  - **location** (`json`, **NOT NULL**, Contains `latitude`, `longitude` and `town` as fields)
  - **description** (`text`, NULLABLE)
  - **ownerID** (`int`, **NOT NULL**, Foreign Key referencing `Users(userID)`)
  - **services** (`string`, NULLABLE, Comma-separated list)
  - **services_pricing** (`float`, NULLABLE)
  - **price_per_night** (`float`, **NOT NULL**)
  - **price_per_person** (`float`, NULLABLE)
  - **availability** (`int`, **NOT NULL**, Enum: Avaliable/Unavailable)
- **Bookings**
  - **bookingID** (`int`, **NOT NULL**, Primary Key)
  - **sailorID** (`int`, **NOT NULL**, Foreign Key referencing `Users(userID)`)
  - **dockID** (`int`, **NOT NULL**, Foreign Key referencing `DockingSpots(spotID)`)
  - **startDate** (`date`, **NOT NULL**)
  - **endDate** (`date`, **NOT NULL**)
  - **paymentMethod** (`string`, **NOT NULL**, Enum: Online/In-person)
  - **paymentStatus** (`string`, **NOT NULL**, Enum: Paid/Unpaid)

  - **people** (`int`, **NOT NULL**)

- **Guides**
  - **guideID** (`int`, **NOT NULL**, Primary Key)
  - **title** (`varchar`, **NOT NULL**)
  - **content** (`text`, **NOT NULL**)
  - **authorID** (`int`, **NOT NULL**, Foreign Key referencing `Users(userID)`)
  - **publicationDate** (`timestamp`, **NOT NULL**)
  - **images** (`json`, NULLABLE, Stores a list of image URLs)
  - **links** (`json`, NULLABLE, Stores a list of related links)
  - **location** (`json`, **NOT NULL**, Contains `latitude` and `longitude` as fields)
  - **isApproved** (`boolean`, **NOT NULL**)
- **Comments**
  - **commentID** (`int`, **NOT NULL**, Primary Key)
  - **guideID** (`int`, **NOT NULL**, Foreign Key referencing `Guides(guideID)`)
  - **userID** (`int`, **NOT NULL**, Foreign Key referencing `Users(userID)`)
  - **content** (`text`, **NOT NULL**)
  - **timestamp** (`timestamp`, **NOT NULL**)
- **Reviews**
  - **reviewerID** (`int`, **NOT NULL**, Primary Key)
  - **rating** (`float`, **NOT NULL**, Range: 1-5)
  - **comment** (`text`, NULLABLE)
  - **dateOfReview** (`date`, **NOT NULL**)
- **Notifications**
  - **notificationID** (`int`, **NOT NULL**, Primary Key)
  - **userID** (`int`, **NOT NULL**, Foreign Key referencing `Users(userID)`)
  - **message** (`text`, **NOT NULL**)
  - **timestamp** (`timestamp`, **NOT NULL**)

4.3 Communication Design

Communication between the mobile app, web app, and the database is facilitated by a RESTful API. The API provides endpoints for various functionalities such as booking, content creation, and system management. All endpoints adhere to the REST architectural style and return responses in JSON format, ensuring consistent and predictable URL patterns and HTTP method usage. All protected endpoints require authentication using JSON Web Tokens (JWT) to ensure secure access to sensitive data.

4.4 Communication Protocols

Messages are sent between the mobile app, web app, and the database using the HTTP protocol. The RESTful API defines a set of endpoints that correspond to specific actions in the system, such as booking a docking spot, creating a guide, or managing users. Each endpoint is accessed using standard HTTP methods (GET, POST, PUT, DELETE) and returns responses in JSON format. The API follows REST principles to provide a stateless and scalable communication architecture.

Structure of the endpoints is as follows:

4.4.1 Authentication Endpoints

All protected endpoints require authentication via JWT (JSON Web Token). The token must be included in the Authorization header in the format:

```
Authorization: Bearer <jwt-token>
```

POST /usr/register

Description: This message structure is used for registering a new user in the system. The request body contains the user's details, including name, surname, email, password, phone number, and role. The response message indicates whether the registration was successful and provides the user's unique ID.

Request Body:

```
{
"name": "John",
"surname": "Doe",
"email": "sailor123@example.com",
"password": "securepassword",
"phone_number": "1234567890",
"role": "sailor" // Options: sailor, dock_owner, editor
}
```

Response:

– 201 Created

```
{
"message": "User registered successfully."
"userID": 101
}
```

POST /usr/login

Description: This message structure is used for logging in an existing user. The request body contains the user's email and password. The response message indicates whether the login was successful and provides an authentication token.

Request Body:

```
{
"email": "sailor_exp@pw.edu.pl",
"password": "reallysafepassword"
}
```

Response:

– 200 OK

```
{
"token": "jwt-token"
}
```

POST /usr/logout

Description: This message structure is used for logging out a user. The request body contains the user's email. The response message indicates whether the logout was successful.

Response Body:

```
{
"message": "User logged out successfully."
}
```

4.5 Sailor Endpoints

GET /docks

Description: This message structure is used for retrieving a list of available docks. The response message contains an array of dock objects, each representing a dock's details, including name, location, description, and availability status. Price information can be provided either per night or per person depending on posting.

Query Parameters:

- **location** (optional): Filter docks by location.
- **date** (optional): Filter docks by availability on a specific date.
- **price** (optional): Filter docks by price range.
- **services** (optional): Filter docks by available services.
- **availability**: Filter docks by availability status.

Response:

– 200 OK

```
[
    {
        "dock_id": 12,
        "name": "Lomza Dock",
        "location": {
            "latitude": 54.12345,
            "longitude": 21.12345,
            "town": "Lomza"
        },
        "price_per_night": 25.00,
        "price_per_person": NULL,
        "availability": "Available"
        "services": ["electricity", "water", "shower"]
    }
]
```

POST /bookings

Description: This message structure is used for booking a docking spot in a dock. The request body contains the sailor's ID, dock ID, start date, and end date. The response message indicates whether the booking was successful and provides a booking ID.

Request Body:

```
    {
    "sailorID": 101,
    "dockID": 12,
    "startDate": "2022-08-01",
    "endDate": "2022-08-05"
    "people": 2
    "payment": "online" // Options: online, in-person
    }
```

Response:

– 201 Created

```
    {
    "message": "Booking created successfully."
    "bookingID": 201
    }
```

DELETE /bookings/:bookingID

Description: This message structure is used for canceling a booking. The response message indicates whether the booking was successfully canceled.

Response:

– 200 OK

```
{
"message": "Booking canceled successfully."
}
```

POST /docks/:dockID/reviews

Description: This message structure is used for submitting a review about a dock. The request body contains the sailor's ID, dock ID, rating, and review content. The response message indicates whether the review was successfully submitted.

Request Body:

```
{
"sailorID": 101,
"dockID": 12,
"rating": 4,
"content": "Great place to dock!"
}
```

Response:

– 201 Created

```
{
"message": "Review submitted successfully."
"reviewID": 301
}
```

4.6 Dock Owner Endpoints

POST /ports

Description: This message structure is used for adding a new port to the system. The request body contains the port's name, location, description, owner ID, and docking spots. The response message indicates whether the port was successfully added and provides the port's unique ID.

Request Body:

```
{
"name": "Lomza Dock",
"location": {
    "latitude": 54.12345,
    "longitude": 21.12345,
    "town": "Lomza"
},
"description": "A beautiful dock in Lomza.",
"ownerID": 201,
"dockingSpots": [
    {
        "availability": "Available",
        "price_per_night": "10",
        "price_per_person": NULL,
    }
]
"services": ["electricity", "water", "shower"]
}
```

Response:

– 201 Created

```
{
"message": "Port added successfully."
"portID": 301
}
```

PUT /ports/:portID

Description: This message structure is used for updating an existing port in the system. The request body contains the updated port details, including name, location, description, and docking spots. The response message indicates whether the port was successfully updated.

Request Body:

```
{
"name": "Lomza Dock",
"location": {
    "latitude": 54.12345,
    "longitude": 21.12345,
    "town": "Lomza"
},
"description": "A beautiful dock in Lomza.",
"dockingSpots": [
    {
        "availability": "Available",
        "price_per_night": NULL,
        "price_per_person": "15",
    }
]
"services": ["electricity", "water", "shower"]
}
```

Response:

– 200 OK

```
{
"message": "Port updated successfully."
}
```

GET /dock-owner/bookings

Description: This message structure is used for retrieving a list of bookings made for a dock owner's port. The response message contains an array of booking objects, each representing a booking's details, including sailor ID, start date, end date, and payment status.

Query Parameters:

- **date** (optional): Filter bookings by date range.
- **people** (optional): Filter bookings by number of people.

Response:

– 200 OK

```
[
    {
        "sailorID": 101,
        "startDate": "2022-08-01",
        "endDate": "2022-08-05",
        "paymentStatus": "Paid"
    }
]
```

4.7 Editor Endpoints

POST /guides

Description: This message structure is used for adding a new guide to the system. The request body contains the guide's title, content, author ID, publication date and lists of pictures and links to referenced websites. The response message indicates whether the guide was successfully added and provides the guide's unique ID.

Request Body:

```
{
"title": "Exploring Lomza",
"content": "A guide to the best spots in Lomza.",
"authorID": 301,
"publicationDate": "2022-07-01",
"pictures": [
    {
        "url": "https://example.com/lomza1.jpg",
        "description": "A scenic view of Lomza's Old Town."
    },
    {
        "url": "https://example.com/lomza2.jpg",
        "description": "The Narew River in Lomza."
    }
],
"links": [
    {
        "url": "https://tourism.lomza.com",
        "description": "Official Lomza Tourism Website"
    },
    {
        "url": "https://example.com/best-cafes-lomza",
        "description": "Top cafes to visit in Lomza"
    }
]
}
```

Response:

– 201 Created

```
{
"message": "Guide added successfully."
"guideID": 401
}
```

PUT /guides/:guideID

Description: This message structure is used for updating an existing guide in the system. The request body contains the updated guide details, including title, content, publication date, and lists of pictures and links to referenced websites. The response message indicates whether the guide was successfully updated.

Request Body:

```
{
"title": "Exploring Lomza",
"content": "A guide to the best spots in Lomza.",
"publicationDate": "2022-07-01"
"pictures": [
    {
        "url": "https://example.com/lomza1.jpg",
        "description": "A scenic view of Lomza's Old Town."
    },
```

```
        {
            "url": "https://example.com/lomza2.jpg",
            "description": "The Narew River in Lomza."
        }
    ],
    "links": [
        {
            "url": "https://tourism.lomza.com",
            "description": "Official Lomza Tourism Website"
        },
        {
            "url": "https://example.com/best-cafes-lomza",
            "description": "Top cafes to visit in Lomza"
        }
    ]
    }
```

Response:

– 200 OK

```
    {
    "message": "Guide updated successfully."
    }
```

DELETE /guides/:guideID
Description: This message structure is used for deleting an existing guide from the system. The response message indicates whether the guide was successfully deleted.
Response:

– 200 OK

```
    {
    "message": "Guide deleted successfully."
    }
```

POST /guides/:guideID/comments
Description: This message structure is used for adding a new comment to a guide. The request body contains the guide ID, user ID, and comment content. The response message indicates whether the comment was successfully added.
Request Body:

```
    {
    "userID": 101,
    "content": "Great guide!"
    }
```

Response:

– 201 Created

```
    {
    "message": "Comment added successfully."
    "commentID": 501
    }
```

4.8 Admin Endpoints

GET /admin/users

Description: This message structure is used for retrieving a list of all users in the system. The response message contains an array of user objects, each representing a user's details, including name, email, phone number, and role.

Response:

– 200 OK

```
[
    {
        "userID": 101,
        "name": "John",
        "surname": "Doe",
        "email": "sailor123@example.com",
        "password": "securepassword",
        "phone_number": "1234567890",
        "role": "sailor" // Options: sailor, dock_owner, editor, admin
    }
]
```

PUT /admin/users/:userID

Description: This message structure is used for updating an existing user in the system. The request body contains the updated user details, including name, email, phone number, and role. The response message indicates whether the user was successfully updated.

Request Body:

```
    {
        "name": "John",
        "surname": "Doe",
        "email": "sailor123@example.com",
        "password": "securepassword",
        "phone_number": "1234567890",
        "role": "sailor" // Options: sailor, dock_owner, editor, admin
    }
```

Response:

– 200 OK

```
    {
    "message": "User updated"
    }
```

DELETE /admin/users/:userID

Description: This message structure is used for deleting an existing user from the system. The response message indicates whether the user was successfully deleted.

Response:

– 200 OK

```
    {
    "message": "User deleted"
    }
```

POST /admin/posts

Description: This message structure is used for approving a new post in the system. The request body contains the post's details, including title, content, author ID, and publication date. The response message indicates whether the post was successfully approved.

Response:

– 201 OK

```
{
"message": "Post approved"
}
```

Response for Rejection:

– 400 Bad Request

```
{
"message": "Post rejected",
"reason": "Inappropriate content"
}
```

DELETE /admin/posts/:postID
Description: This message structure is used for deleting an existing post from the system. The response message indicates whether the post was successfully deleted.
Response:

– 200 OK

```
{
"message": "Post deleted"
}
```

4.9 Diagrams

4.9.1 Use Case Diagrams

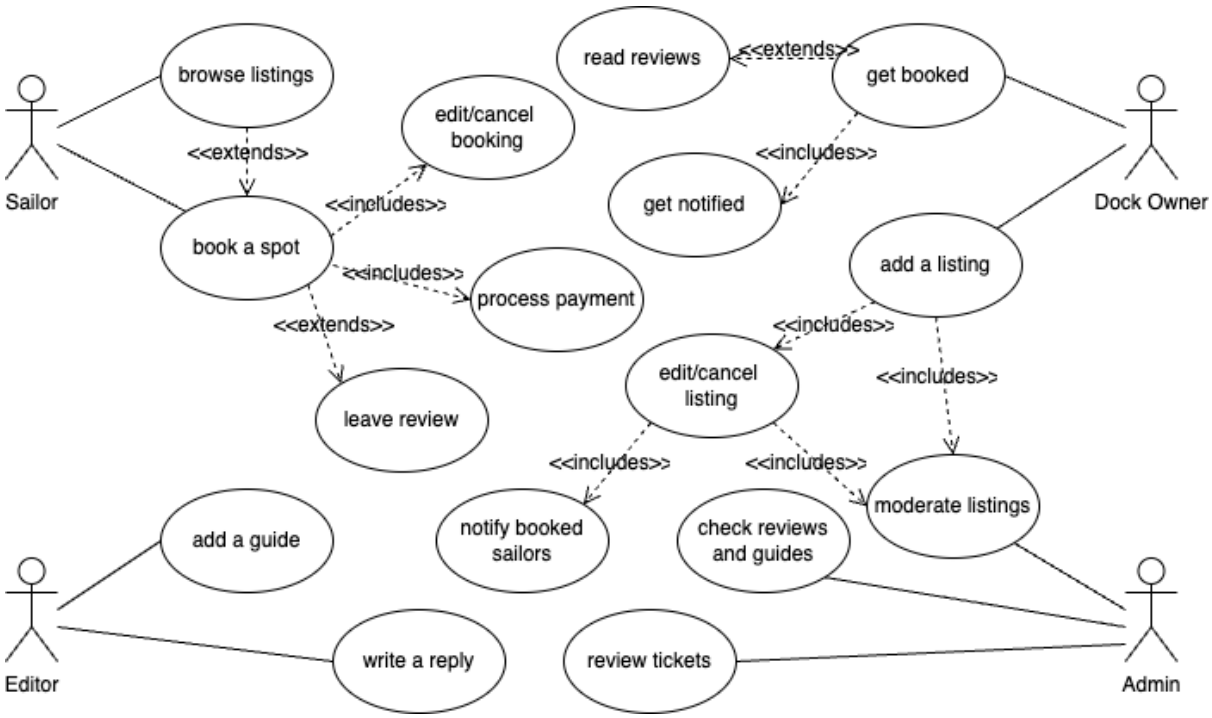

Figure 1: Use Case Diagram

Our application should combine three main functionalities: a platform for renting docking spots, the ability to post educational and informative content, and the system for moderating the community and the content.

Sailors create the demand for dock renting services. They can browse dock listings. They can read reviews of docks and book the ones that are available. Before payment process they can cancel or modify their bookings. After payment they can leave a review of the dock listing.

Dock owners supply the platform with renting services. They have the ability to add a new or edit/cancel an existing dock listing. In case of editing or canceling of a listing, every sailor that is booked for this listing gets notified. Dock owners are notified when their listing gets booked by a sailor. They can also read reviews of their listings.

Editor enriches our application with educational content in form of guides and replies to sailor's reviews. The subject of a guide can range from a list of interesting landmarks in a vicinity to tips and tricks for beginner sailors.

Admin protects our application from harmful or untrue content. He approves every newly created or modified listing before it is being published on the renting platform. He also checks reviews and guides for violations of terms of services.

4.9.2 Class Diagrams

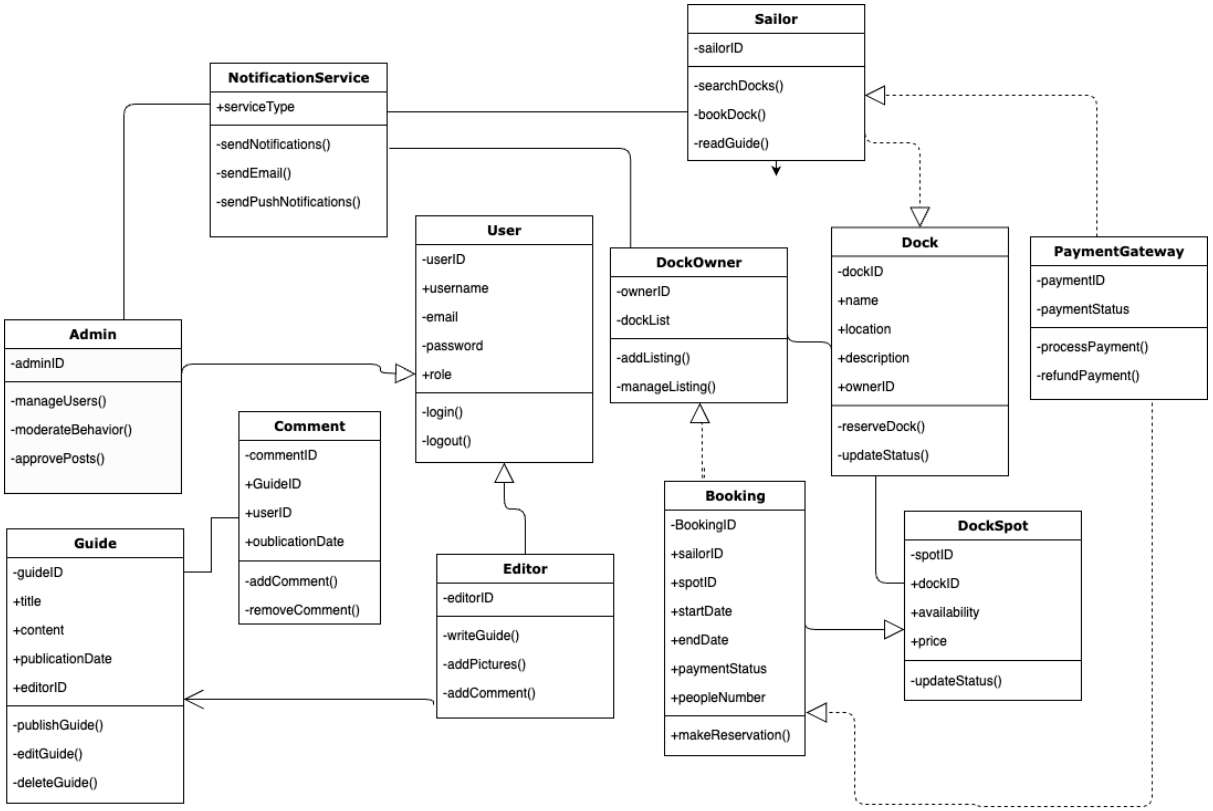

Figure 2: Class Diagram

This class diagram represents a *dock management system* with key roles and entities:

1. **User**: Base class with login/logout functionality.
   - **Subclasses:**
     - **Admin**: Manages users and content.
     - **DockOwner**: Manages dock listings.
     - **Sailor**: Searches, books docks, and reads guides.
     - **Editor**: Writes and edits guides.
2. **Dock**: Represents docks with operations to update status and reserve docks.
3. **Guide**: Contains guide content with options to publish, edit, or delete.

4. **PaymentGateway**: Handles payments and refunds.
5. **NotificationService**: Sends notifications, emails, and push alerts.

Relationships:

- *Admin*, *DockOwner*, *Sailor*, and *Editor* inherit from *User*.
- *Sailor* interacts with *Dock* and *Guide*.
- *DockOwner* manages *Dock*.
- *PaymentGateway* processes payments.
- *NotificationService* supports notifications across the system.

4.9.3 Sequence Diagrams

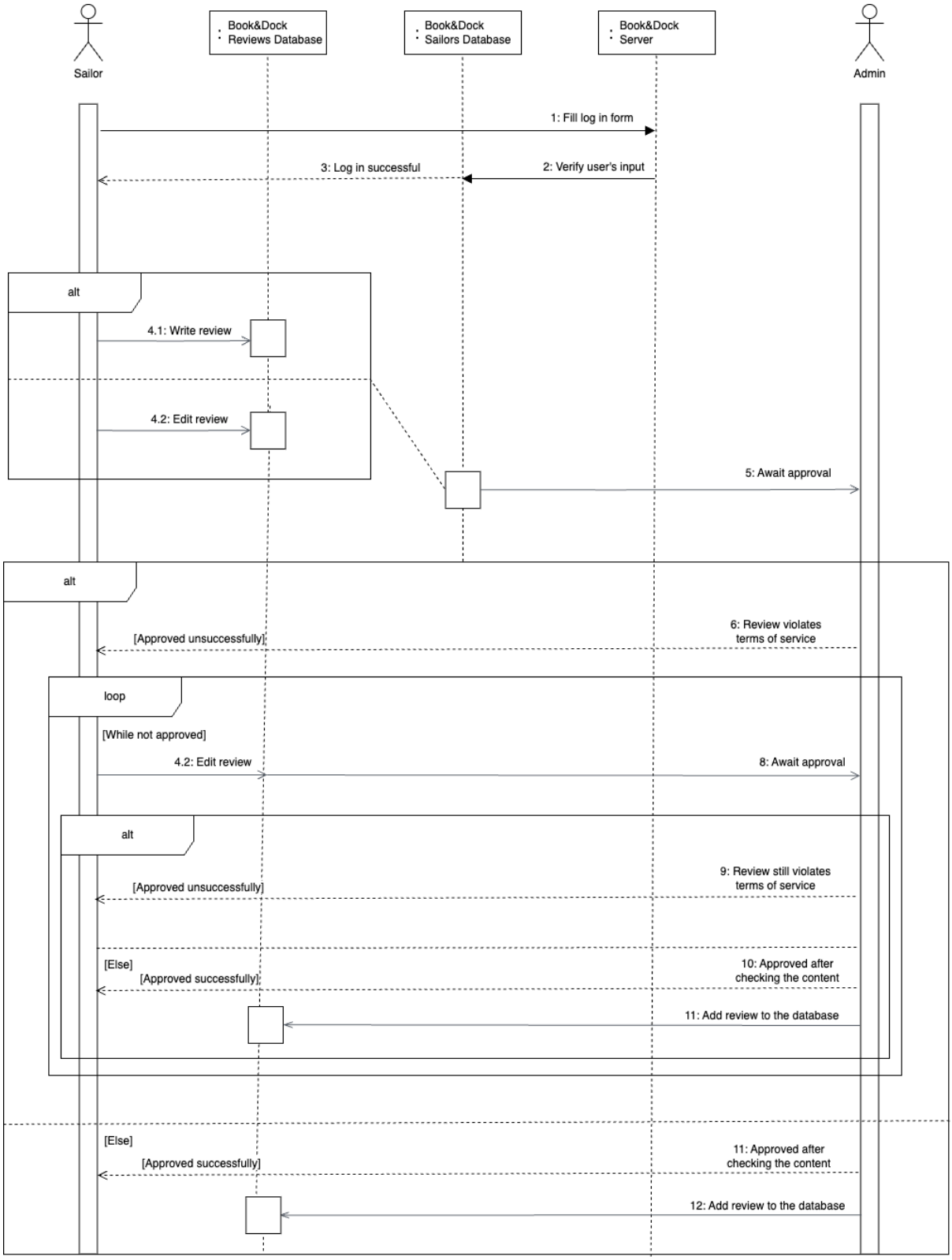

Figure 3: Sailor-Admin Sequence Diagram

This diagram describes the communication between sailor posting a review and admin. Sailor either writes a new review or modifies an existing one. The server that holds the review sends a request for approval to admin. If admin approves the review it is added to the database, it becomes visible for other

users and the sailor recieves a notification about the success. If admin does not approve the review then the sailor is notified about the need for further modifications. Until the listing is not approved the owner has a possibility to correct the review and await admin's approval.

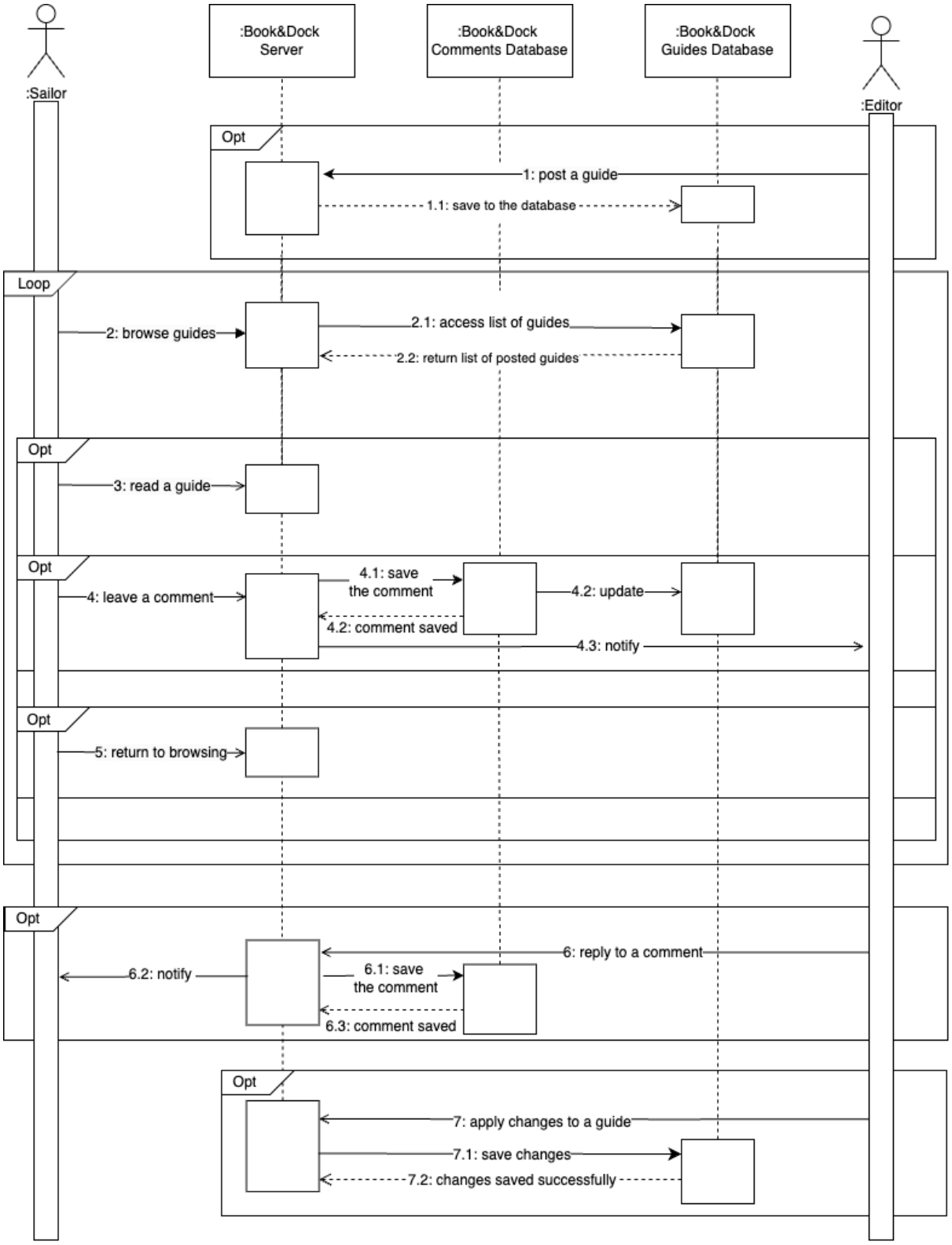

Figure 4: Sailor-Editor Sequence Diagram

This diagram describes the communication between editor posting a guide and sailor reading it. Editor posts a guide to the server. It is then saved to the database. To browse guides sailor needs to

wait for the server to fetch the list of posted guides from the database. Then sailor can read a guide and leave a comment. If he does the latter the comment is saved to the database and the editor is notified. Editor can reply to a comment and modify the guide, where in both cases the input is saved to the database.

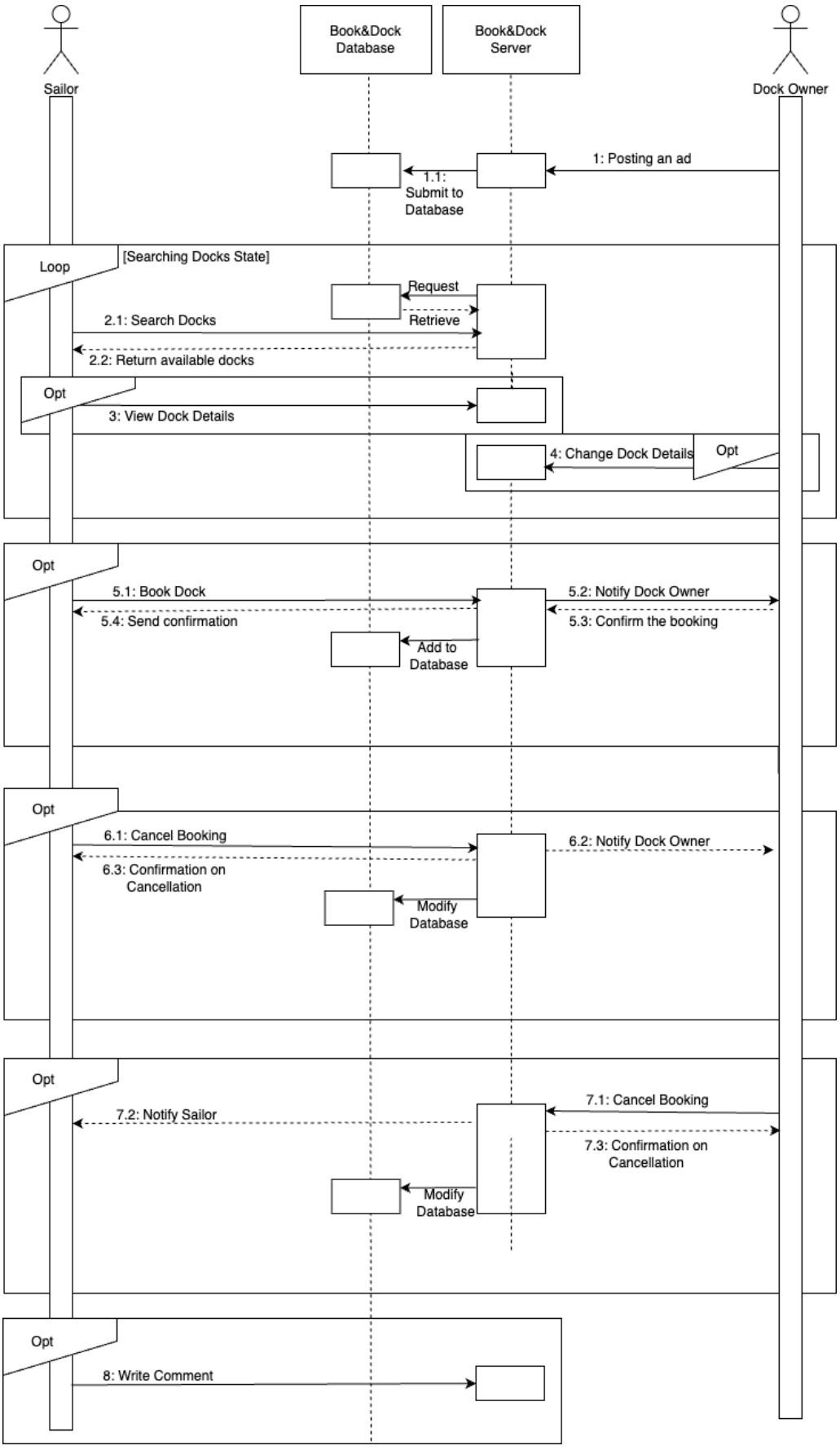

Figure 5: Sailor-Dock Owner Sequence Diagram

This diagram describes the communication between sailor and dock owner. Dock owner posts a listing to the server which is then added to the database. After that, sailor can request the server to fetch the list of available docks from the database. When the list is returned to the sailor he can view dock details, book a dock, cancel an ongoing booking or write a review.

When sailor books a dock, the dock owner is sent request for confirmation. If he confirms, the booking is saved to the database and the sailor is notified.

When either sailor or dock owner cancels a booking, the database is modified. After that both sides are notified of the cancelation.

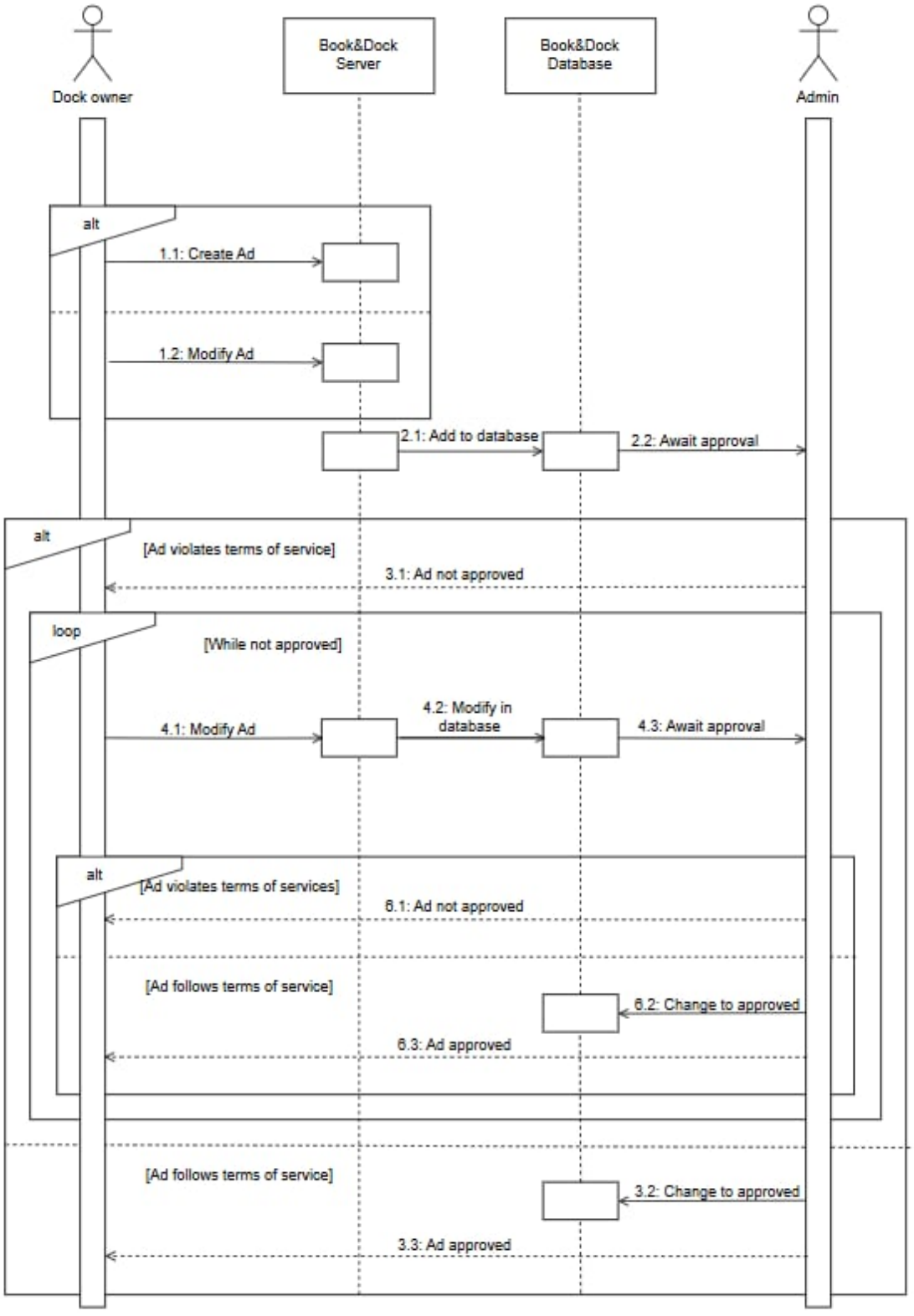

Figure 6: Dock Owner-Admin Sequence Diagram

This diagram describes the communication between dock owner posting a dock listing/ad and admin moderating it. Dock owner either creates a new listing or modifies an existing one. The server saves the

ad in the database with isApproved flag set to false. Then, the database sends a request for approval to admin. If admin approves the listing, its isApproved flag is set to true and the ad becomes available for sailors and the dock owner receives a notification about the success. If admin does not approve the listing then the dock owner is notified about the need for further modifications. Until the listing is not approved the owner has a possibility to correct the listing and await Admin's approval.

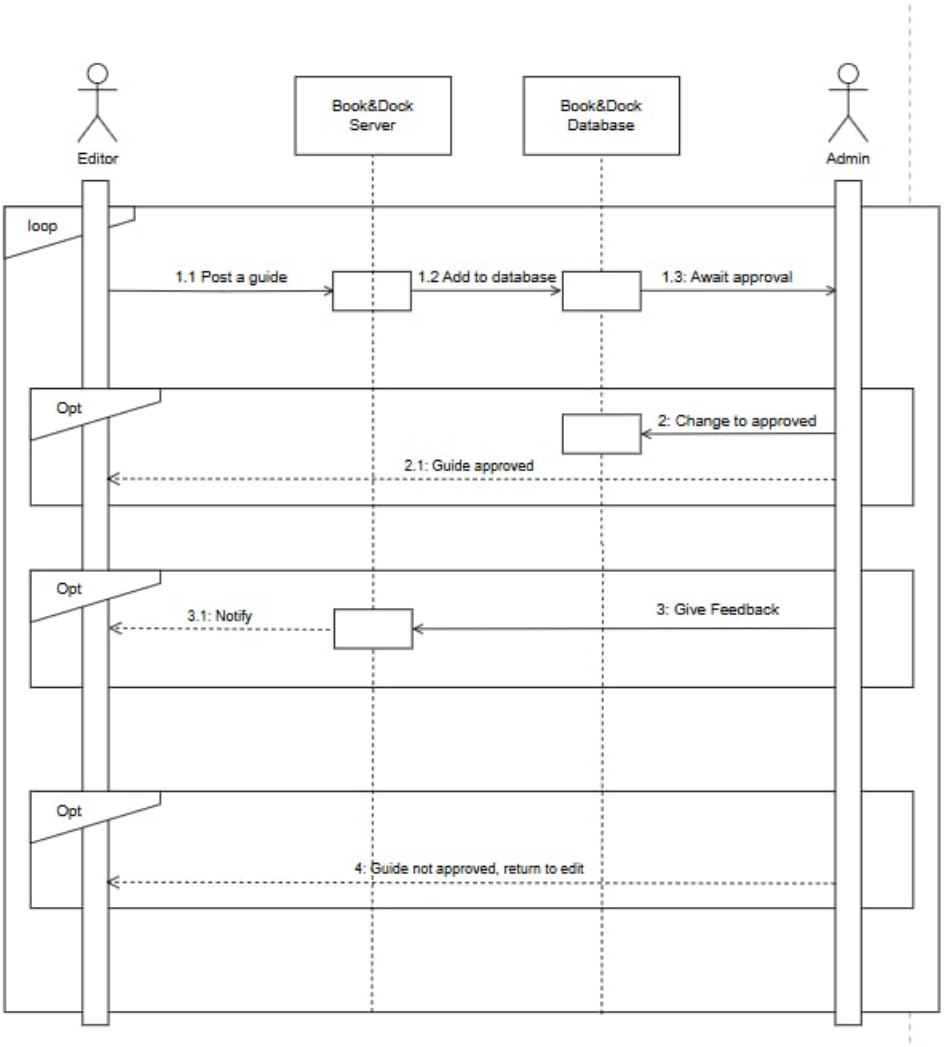

Figure 7: Editor-Admin Sequence Diagram

This diagram describes the communication between editor and admin. When the editor posts a guide, it is added to the database with isApproved flag set to false. Admin either accepts it, setting the isApproved to true or rejects it. In both cases, the editor is notified about the decision. The admin can also send feedback to the editor with additional information.

4.9.4 Activity Diagrams

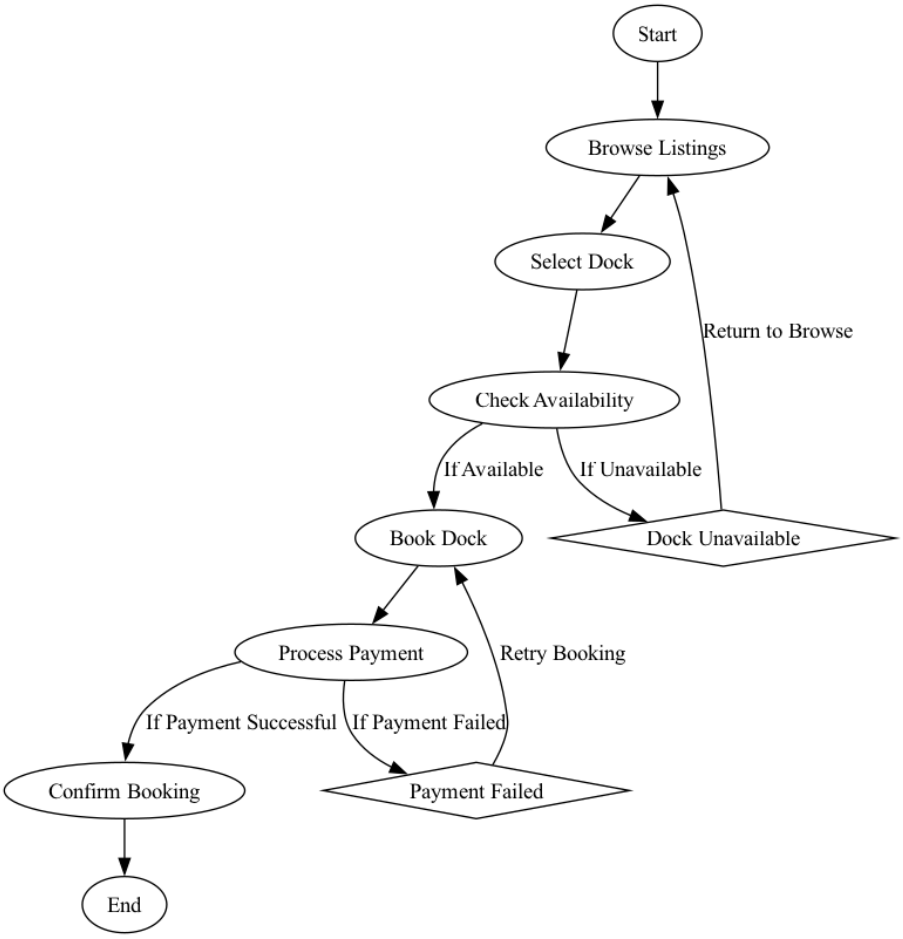

Figure 8: Sailor Activity Diagram for Booking a Dock

This diagram describes the process of booking a dock by a sailor. The sailor first searches for available docks. Then he selects a dock and books it. After booking, payment is made and the booking is confirmed.

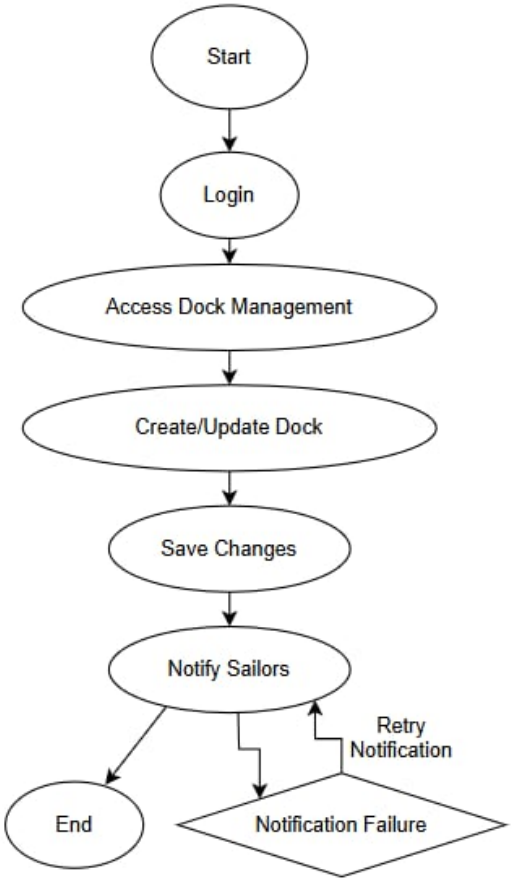

Figure 9: Dock Owner Diagram for Updating/Adding Docks

This diagram describes the process of updating or adding a dock by a dock owner. The dock owner first selects a dock to update or adds a new one. Then they save changes and it notifies sailors about the changes.

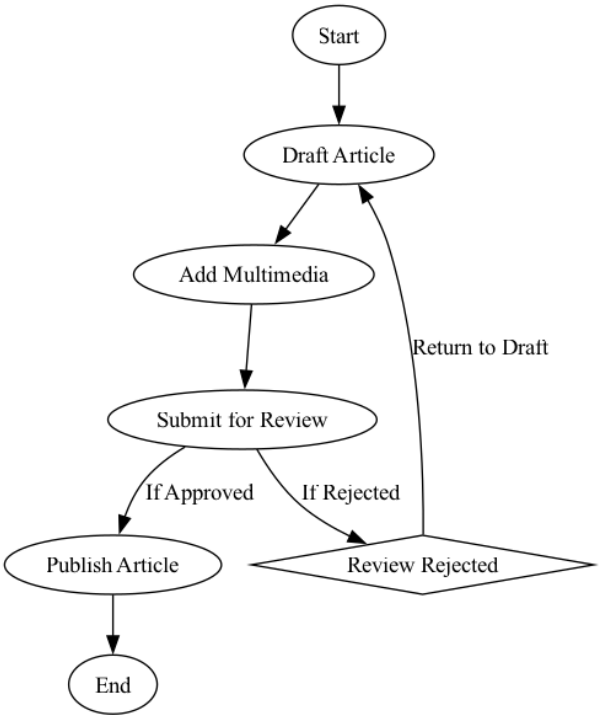

Figure 10: Editor Activity Diagram for Adding Guides

This diagram describes the process of adding a guide by an editor. The editor first creates a new guide, then saves it to the database. The guide is then published and sailors can access it.

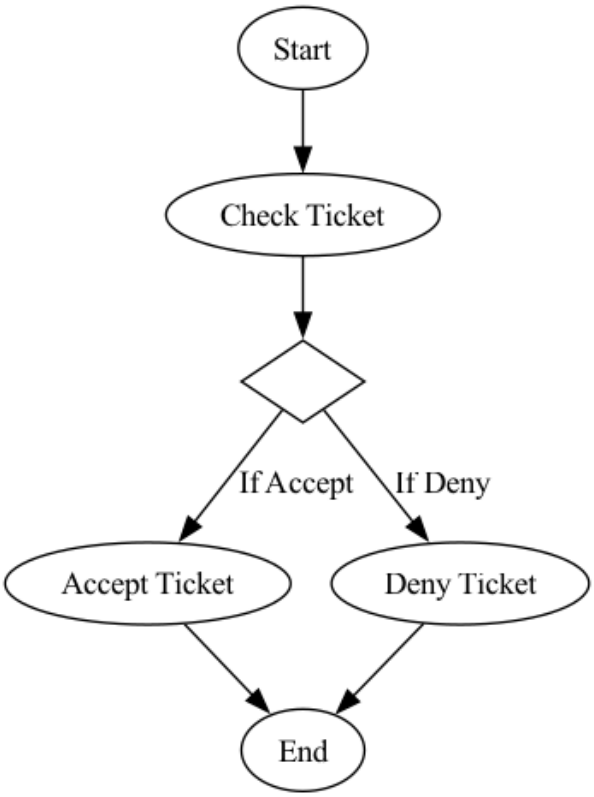

Figure 11: Admin Activity Diagram for Approving

This diagram describes the process of approving a post by an admin. The admin first receives a

request to approve a post, then reviews the post. If the post is approved, it is added to the database. If the post is rejected, the admin provides feedback to the user.

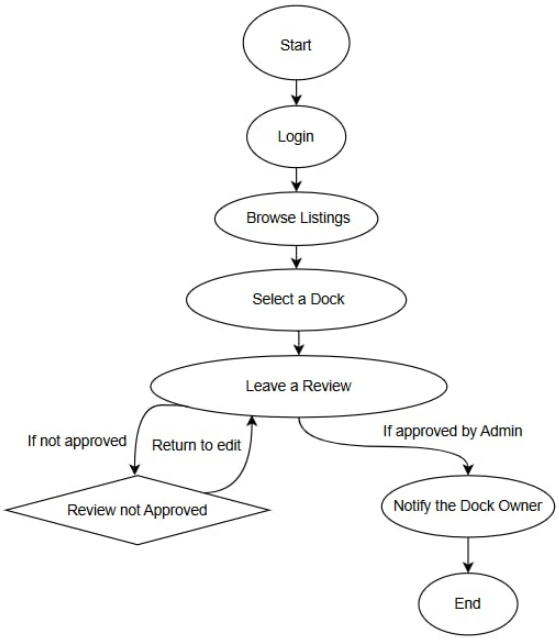

Figure 12: Sailor Activity Diagram for Leaving Reviews

This diagram describes the process of sailor leaving a Review of a Dock. After successful login, Sailor browses through Listings, selects a Dock and leaves a Review of it. If the Review is not approved by Admin, Sailor returns to editing. If the Review is approved, the Dock Owner is notified and the Review becomes visible to other users.

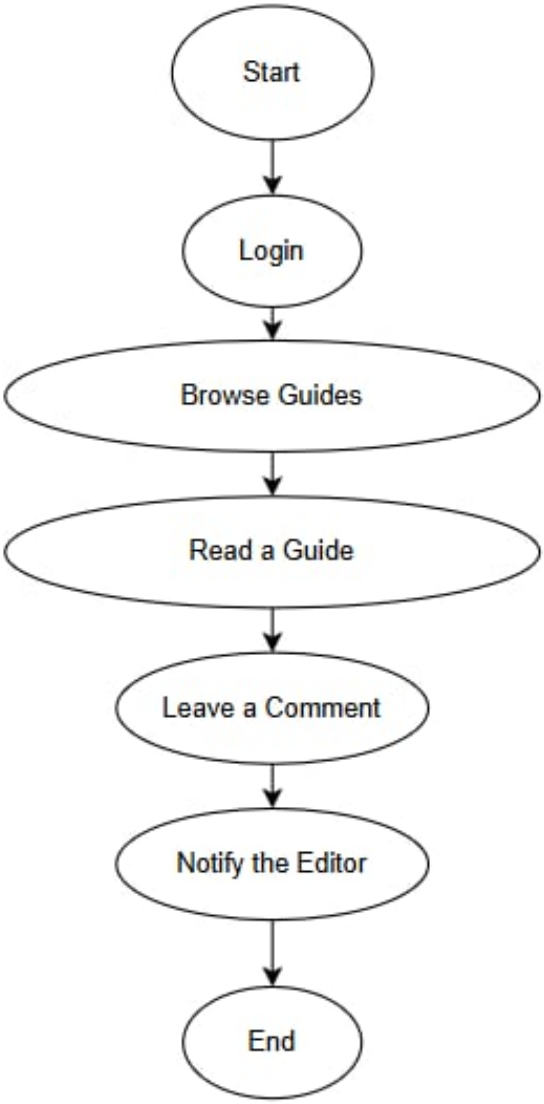

Figure 13: Sailor Activity Diagram for Reading Guides

This diagram describes the process of sailor leaving a Comment of a Guide. After successful login, Sailor can browse for a Guide, read it and leave a Comment under it. When a Comment is left, the Editor is notified.

4.9.5 State Diagrams

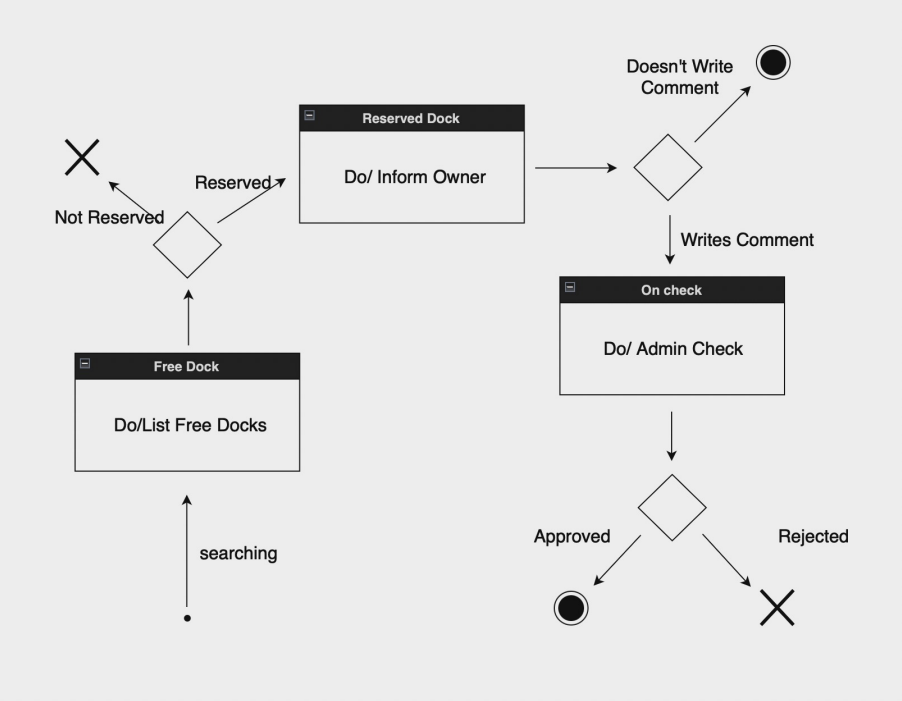

Figure 14: Sailor State Diagram

This diagram describes the states of a sailor in the system. The sailor can be in the following states: browsing docks, reserving a dock, and reviewing a dock.

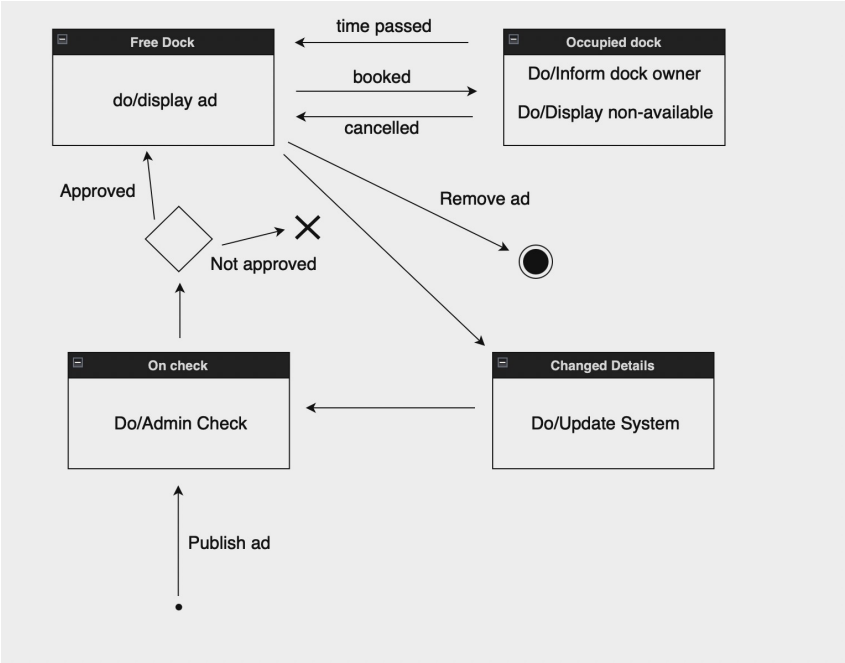

Figure 15: Dock Owner State Diagram

This diagram describes the states of a dock owner in the system. The dock owner can be in the following states: adding a dock, updating a dock, and managing bookings.

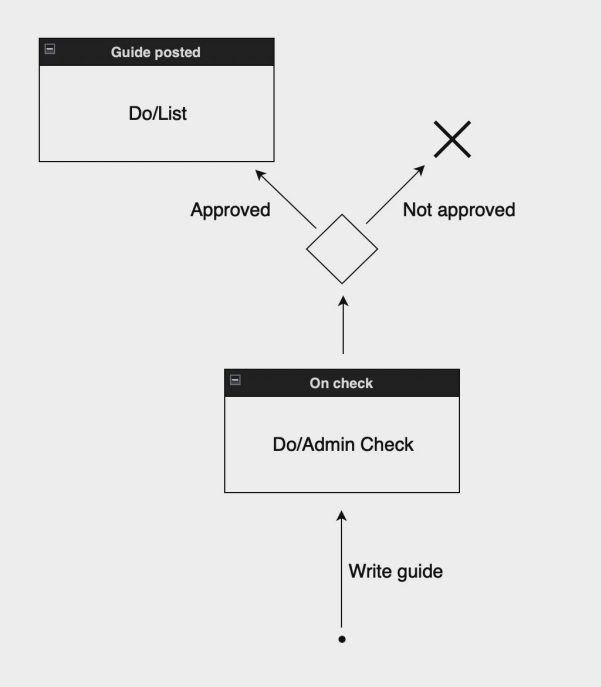

Figure 16: Editor State Diagram

This diagram describes the states of an editor in the system. The editor writes a guide and goes on check then it is in the state of publishing the guide.

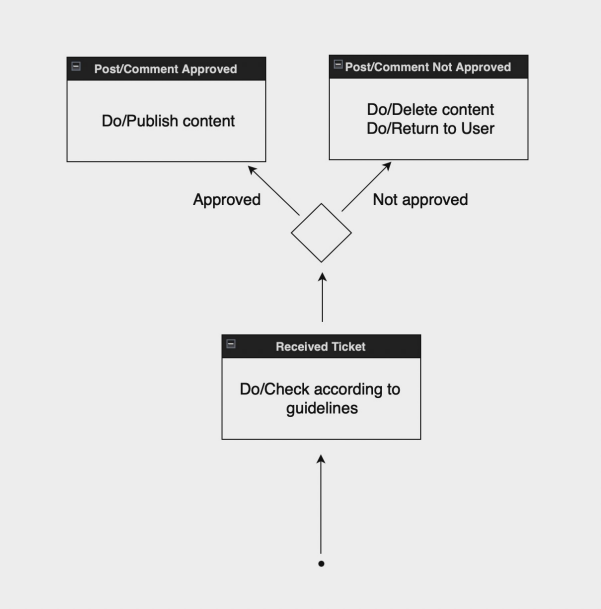

Figure 17: Admin State Diagram

This diagram describes the states of an admin in the system. The admin can be in the following states: reserving tickets, approving tickets or rejecting.

5 Glossary

- Dock - a place where boats are parked for a certain period of time.
- Port - a place where boats can dock, synonym for dock.

- Sailor - a person who sails a boat.
- Dock Owner/Manager - a person who owns/manages a dock.
- Editor - a person who writes articles and posts them on the platform.
- Admin - a person who manages users and monitors activity on the platform.
- Guide - an article that provides tips and information about a certain topic.