Software Requirements Specification

for

SchoolAid : Helper &Seeker System

Prepared by KAVIYA B

13-09-2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

SchoolAid : Helper & Seeker Platform

1.INTRODUCTION:

1.1.Purpose:

The aim of *SchoolAid* is to create a supportive academic platform that connects students in schools who seek help with those willing to provide it.The goal is to ensure no student lags behind in their studies due to absence, doubts, or lack of access to learning materials.This application is specifically designed for **schools** and can be used by students, teachers, and school management to promote collaborative learning.The project was developed to address common issues faced by school students, such as missing notes during absences, difficulty in understanding concepts, and lack of proper communication with peers or faculty. *SchoolAid* enables better academic support through an organized and accessible system.

1.2 Document Conventions

This Software Requirements Specification (SRS) follows standard formatting and writing conventions to ensure clarity and consistency. The conventions used in this document are as follows:

- **Bold text** is used to highlight section headings, important feature names, and module titles.
- *Italic text* is used for newly introduced terms or role definitions such as *Helper* or *Seeker*.
- Numbered headings follow a hierarchical structure (e.g., 1, 1.1, 1.2) for better organization.
- Bullet points (•) are used to list features, benefits, or grouped requirements.
- All requirements are assumed to have equal priority unless a specific priority is mentioned.
- The document is prepared using **Times New Roman**, size **12 pt**, with **1.15 line spacing**.

1.3.Intended Audience:

- ***Seekers*** (students, faculty):Users who request help — such as notes, explanations, or guidance — when they miss a class or struggle with a topic.
- ***Helpers*** (students, faculty):
  Users who respond to help requests by sharing notes, solving doubts, or providing mentorship.
- ***Admin****:*
  A faculty member or system administrator responsible for managing users, moderating content, and maintaining the platform.

1.4.Project Scope:

In the future, *SchoolAid* may include advanced features such as AI-based helper recommendations, real-time chat support, offline access, gamified rewards for active users, and integration with existing school learning platform**s.**

1.5.References:

the *SchoolAid* project is not yet developed and is currently in the planning and documentation phase. Therefore, no existing system documents, style guides, or previous versions are available for reference.

2.Overall Description

2.1.Product Perspective

The **SchoolAid Management System** is a new, self-contained product designed to connect students (seekers) with mentors, teachers, and volunteers (helpers). Unlike traditional school management systems that focus only on administrative tasks, this product emphasizes **collaborative learning and knowledge-sharing**.

This system acts as an independent platform but can also be integrated with existing educational systems to enhance support for students.

2.2. Product Features

1. User Management
    - Separate access for Admin, Helpers, and Seekers.
    - Secure login and profile management for each role.
2. **Helper Functions**
    - Upload and share learning resources (documents, notes, videos).
    - Provide guidance and answer queries from seekers.
    - Track seeker engagement and feedback.
3. **Seeker Functions**
    - Access shared resources from helpers.
    - Post queries or requests for help.
    - Receive personalized guidance and track progress.
4. **Admin Functions**
    - Manage users (add/remove helpers and seekers).
    - Monitor system activities and ensure security.
    - Generate reports on system usage and interactions.

5. **Collaboration & Communication**
    - Direct communication between helpers and seekers.
    - Notification system for updates, new resources, and responses.

2.3.User Classes and Characteristics

1. ***Seeker***

Characteristics:

- Can register/login and create a profile.
- Posts questions or help requests (subject/topic specific).
- Can browse helpers and choose one.
- Can view answers, suggestions, or resources shared by helpers.
- Can give feedback or ratings to helpers.
- Profile may include: Name, Grade/Year, Subjects of interest, Learning goals.

2. ***Helper***

Characteristics:

- Can register/login and create a profile.
- Views help requests posted by seekers.
- Provides answers, explanations, or study resources.
- Can specialize in subjects (tagged expertise).
- Gains ratings/reputation points from seekers.
- Profile may include: Name, Expertise, Experience level, Availability status.

3. ***Admin***

Characteristics:

- Manages user accounts (approve, block, verify).
- Monitors seeker–helper interactions for quality and safety.
- Maintains the reputation/points system.
- Manages content (remove abusive/irrelevant posts).
- Generates reports on usage statistics.

2.4 Operating Environment

The SchoolAid platform is designed to operate in a modern client–server environment with support for both web and mobile access.

- **Hardware Platform**
    - Server: Minimum quad-core processor, 8 GB RAM, 250 GB storage; recommended octa-core processor, 16 GB RAM, 500 GB+ storage.
    - Client Devices: Desktops/laptops (4 GB RAM minimum) and smartphones/tablets (2 GB RAM minimum).
- **Operating System**
    - Server: Linux (Ubuntu 20.04 or later) or Windows Server 2019+.
    - Client: Windows 10/11, macOS, Android 9+, iOS 13+.
- **Software Components**
    - Backend: Python 3.9+ with FastAPI framework.
    - Database: PostgreSQL 13+.
    - Web Server: Uvicorn/Gunicorn for serving APIs.
    - Development & Testing: Standard IDEs, API testing tools (e.g., Postman), Git/GitHub for version control.
- **Coexistence with Other Applications**
    - Designed to integrate with school management systems and learning management systems (LMS).
    - Supports APIs for future integration with third-party educational platforms.

2.5 Design and Implementation Constraints

The following factors restrict the design and development of SchoolAid:

- **Technology Constraints**
    - The backend must be implemented in Python using the FastAPI framework.
    - PostgreSQL is the mandatory database technology.
    - Deployment must rely on Uvicorn/Gunicorn as the application server.
- **Hardware Limitations**
    - System performance depends on minimum server resources (8 GB RAM, quad-core CPU).
    - Low-end client devices may face delays when loading media-heavy content.

- **Integration Constraints**
    - The platform must provide RESTful APIs for potential integration with existing school systems.
    - Compatibility with standard web browsers is required.
- **Security Considerations**
    - Authentication and authorization are mandatory for all users (seekers, helpers, admins).
    - All data must be encrypted during transmission (HTTPS).
    - Sensitive student data must comply with basic privacy standards.
- **Standards and Conventions**
    - Development must follow PEP 8 (Python coding standards).
    - Database schema must adhere to normalization rules (3NF minimum).
    - Version control using Git is required for collaborative development.

2.6 User Documentation

The following user documentation will be delivered with the SchoolAid system:

- **User Manual (Digital/PDF):** Step-by-step guide for seekers, helpers, and admins.
- **Online Help:** Built-in FAQ and help section within the application.
- **API Documentation:** Swagger/OpenAPI documentation auto-generated for developers and integrators.
- **Tutorials:** Video or slide-based quick start guides for onboarding.

All documentation will be provided in English and accessible through the platform's help menu or official website.

2.7 Assumptions and Dependencies

- **Assumptions**
    - Students, teachers, and admins have access to devices with stable internet.
    - Schools provide basic IT infrastructure (server/cloud hosting, admin oversight).
    - Users possess basic digital literacy skills.
    - Network connectivity is sufficient for uploading/downloading learning materials.
- **Dependencies**

- The system depends on the PostgreSQL database for storage.
- Uvicorn/Gunicorn must be used as the ASGI server for FastAPI deployment.
- Future expansion (e.g., mobile app) depends on availability of cross-platform frameworks (Flutter/React Native).
- External APIs may be used in future for AI-based recommendations and chat integrations.

3. System Features

3.1 User Management

3.1.1 Description and Priority

The User Management feature allows secure access and management of users on the platform. Users include **Seekers**, **Helpers**, and **Admins**. This feature is **High priority** as it ensures controlled access and data integrity.

Priority Components:

- Benefit: 9
- Penalty: 8
- Cost: 5
- Risk: 7

3.1.2 Stimulus/Response Sequences

- **Stimulus:** A new user attempts to register on the platform.
- **Response:** The system validates user details, assigns the correct role, and creates a secure profile.
- **Stimulus:** An existing user logs in.
- **Response:** The system authenticates credentials and redirects the user to their dashboard based on role.
- **Stimulus:** Admin modifies or deletes a user.
- **Response:** The system updates the user database and confirms the operation.

4. External Interface Requirements

4.1 User Interfaces

The SchoolAid system provides an intuitive graphical user interface (GUI) for all users (Seekers, Helpers, Admins). The interface is designed for ease of use, accessibility, and consistency across devices.

Characteristics:

- **Navigation:** Menu bar with main sections: Dashboard, Resources, Queries, Notifications, Profile, and Admin Panel (for Admins).
- **Layout:**
  - Header: System logo, notifications icon, user profile menu.
  - Sidebar: Role-specific menu items (e.g., Post Query for Seekers, Upload Resource for Helpers).
  - Main Panel: Displays content dynamically based on selected menu option.
- **Buttons & Controls:** Standard buttons for Submit, Upload, Download, Cancel, Reply, and Edit.
- **Forms:** Input validation for user registration, query posting, and file uploads. Error messages displayed in red below input fields.
- **Help & Documentation:**
  - "Help" button present in the header for accessing FAQs and tutorials.
  - Tooltips provided for interactive buttons and features.
- **Keyboard Shortcuts:**
  - Ctrl+H: Open Help
  - Ctrl+L: Logout
  - Ctrl+N: Create New Query (Seeker) / Upload Resource (Helper)
- **Consistency:** UI design follows school portal standards, with uniform color schemes, font styles (Times New Roman or similar), and responsive design for mobile and desktop.

Example Screens:

1. Seeker Dashboard: Lists pending queries, available resources, and notifications.
2. Helper Dashboard: Upload resources, view seeker queries, track engagement.
3. Admin Panel: User management interface, activity logs, report generation.

4.2 Hardware Interfaces

SchoolAid interacts with both server and client hardware to ensure smooth operation.

Server Requirements:

- Minimum: Quad-core CPU, 8 GB RAM, 250 GB storage
- Recommended: Octa-core CPU, 16 GB RAM, 500 GB+ storage
- Storage: SSD recommended for faster database access

Client Devices:

- Desktops/Laptops: Minimum 4 GB RAM, modern CPU
- Smartphones/Tablets: Minimum 2 GB RAM
- Supported display resolutions: 1366x768 and higher

Communication with Hardware:

- File uploads/downloads via standard HTTP/HTTPS protocols
- Media rendering compatible with GPU acceleration on clients (if available)
- Input devices: Keyboard, mouse, touch (mobile/tablet)

4.3 Software Interfaces

SchoolAid integrates with several software components for operation and data management.

Backend & Database:

- Python 3.9+ (FastAPI framework) for API services
- PostgreSQL 13+ for relational data storage
- Uvicorn/Gunicorn as ASGI servers for hosting APIs

Client/Front-End:

- Web browser support: Chrome, Firefox, Edge, Safari (latest versions)
- Mobile: Android 9+, iOS 13+ (via responsive web app or future mobile app)

Third-party Tools:

- Git/GitHub for version control
- Postman for API testing
- Swagger/OpenAPI for auto-generated API documentation

Data Interchange:

- JSON format for all API requests/responses
- Data validation required at both client and server sides

Integration Considerations:

- RESTful APIs designed for integration with existing School Management Systems (SMS) or LMS
- All APIs must follow standardized endpoints and authentication mechanisms (JWT tokens)

4.4 Communications Interfaces

The SchoolAid system requires reliable communication for notifications, query management, and real-time updates.

Protocols & Standards:

- **HTTP/HTTPS**: All client-server interactions must use HTTPS for encrypted data transfer
- **WebSocket (optional)**: For real-time notifications and chat functionality between Seekers and Helpers
- **SMTP/Email API**: For sending email alerts to users

Message Formatting:

- JSON messages for API communication
- Emails follow standard MIME format with HTML and plain-text versions

Security & Encryption:

- TLS 1.2 or higher for all client-server communications
- Sensitive user data (passwords, personal details) must be encrypted at rest and in transit

Data Transfer Requirements:

- Minimum bandwidth: 2 Mbps for standard operations
- Synchronization mechanisms to handle offline or delayed connectivity for mobile devices

5.Other Nonfunctional Requirements

5.1 Performance Requirements

- **System Response Time:**
  - Page load and API responses for standard requests (e.g., retrieving resources, posting queries) must be under **2 seconds** under normal load (up to 500 concurrent users).
  - File uploads/downloads must complete within **10 seconds for files up to 50 MB**.
- **Scalability:**
  - The system should support up to **2000 concurrent users** with minimal degradation in performance.
  - Database queries for resource retrieval must scale linearly with the number of users and resources.
- **Availability:**
  - The system must be available **99.5% of the time** during school hours (6 AM–10 PM).
- **Real-Time Notifications:**
  - Notifications (e.g., query responses or resource uploads) must be delivered within **5 seconds** under normal network conditions.

5.2 Safety Requirements

- **Data Integrity:**
  - Prevent accidental deletion or corruption of learning materials and user records.
  - Implement transactional controls in the database to ensure consistency.
- **Safe Uploads:**
  - Restrict uploaded files to safe formats (e.g., PDF, DOCX, MP4) and scan for malware.
- **User Protection:**
  - Prevent harassment through content moderation tools, user reporting mechanisms, and restricted messaging if abuse is detected.
- **Regulatory Compliance:**
  - Follow local education and privacy regulations regarding student data.

5.3 Security Requirements

- **Authentication & Authorization:**
  - All users must register with unique credentials (email/ID and password).
  - Role-based access control to enforce permission levels (Seeker, Helper, Admin).
- **Data Encryption:**
  - All sensitive data must be encrypted in transit using HTTPS/TLS and at rest in the database.
- **Password Policy:**
  - Minimum 8 characters, must include letters and numbers. Passwords stored as salted hashes.
- **Audit & Logging:**
  - All system actions (login attempts, resource uploads, query responses) must be logged.
  - Logs should be protected from tampering and retained for a minimum of 1 year.

- **Future Compliance:**
    - System design should support compliance with GDPR, FERPA, or similar student data privacy regulations.

5.4 Software Quality Attributes

- **Usability:**
    - Interface must be intuitive and user-friendly; minimal training required.
    - Consistent UI across web and mobile devices.
- **Reliability:**
    - System must handle at least 2000 concurrent users without crashes.
    - Automatic recovery mechanisms for server or database failures.
- **Maintainability:**
    - Code should follow **PEP 8** standards and be modular to allow future updates.
- **Portability:**
    - Web platform should function on major browsers (Chrome, Firefox, Edge, Safari).
    - Mobile responsiveness ensures usability on Android 9+ and iOS 13+.
- **Interoperability:**
    - RESTful APIs should support integration with existing school systems or third-party LMS.
- **Testability:**
    - Automated tests should cover major features (user management, queries, resource uploads).
    - APIs should be testable via Postman or similar tools.
- **Robustness:**
    - Handle invalid input gracefully with proper error messages.
    - System should maintain operations even under network delays or partial server failures.

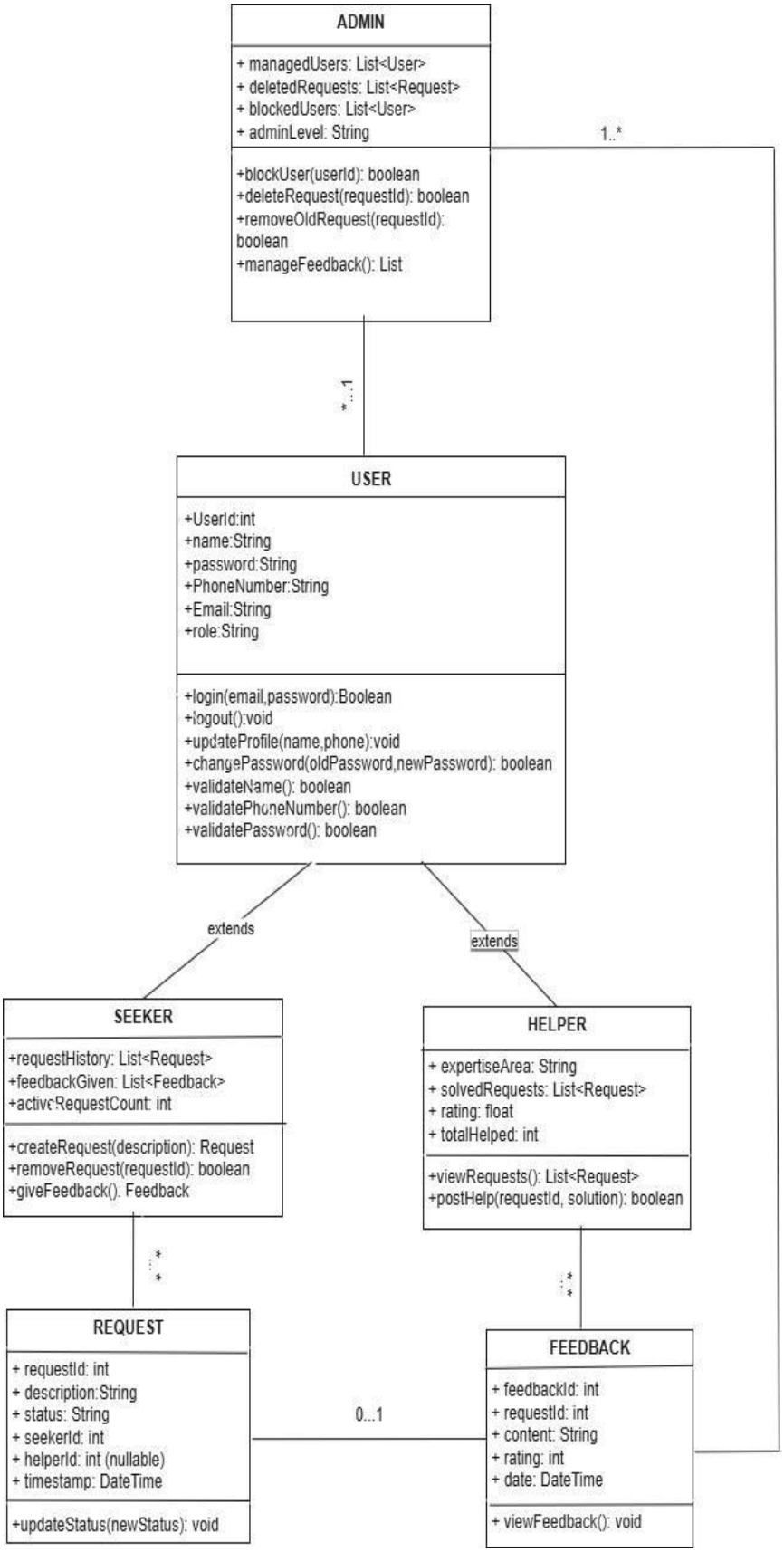
6. Other Requirements

- **Database Requirements:**
    - Use **PostgreSQL 13+** with normalized schema (3NF or higher).
    - Daily backups and disaster recovery plans must be implemented.
- **Internationalization Requirements:**
    - Support for English; future support for additional languages should be planned.
- **Legal Requirements:**
    - Compliance with student data protection regulations (local/state law).
    - Copyright compliance for shared learning materials.
- **Reuse Objectives:**
    - Code modules (e.g., user authentication, file handling) should be reusable for future educational apps.
- **Documentation Requirements:**
    - User manuals, API documentation (Swagger/OpenAPI), and tutorials must be maintained and updated with every major release.
- **Future Enhancements:**
    - AI-based helper recommendations, gamification for engagement, offline access for mobile users.
    - Integration with school LMS and third-party educational platforms.

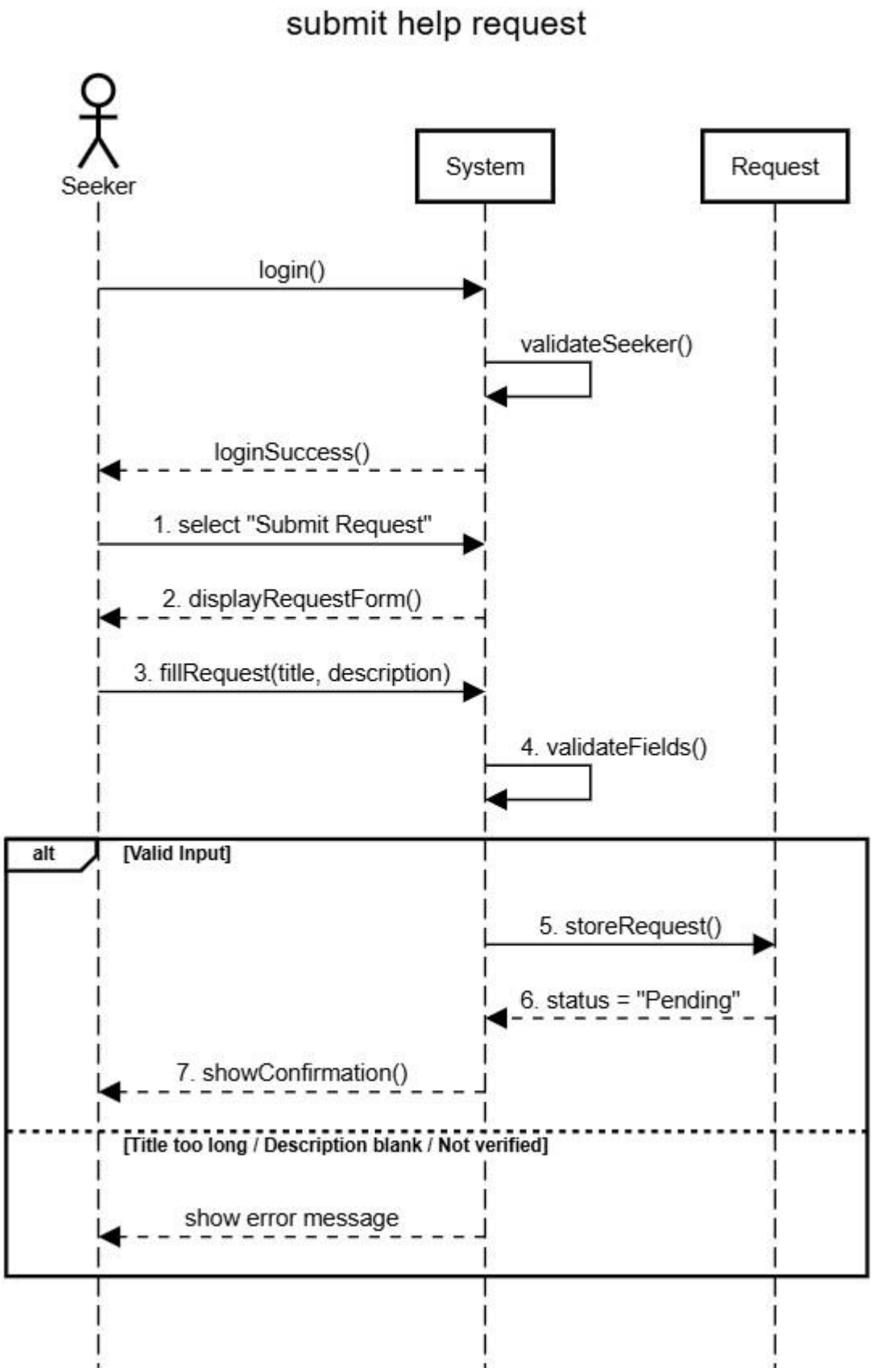
6.1.APPENDIX:

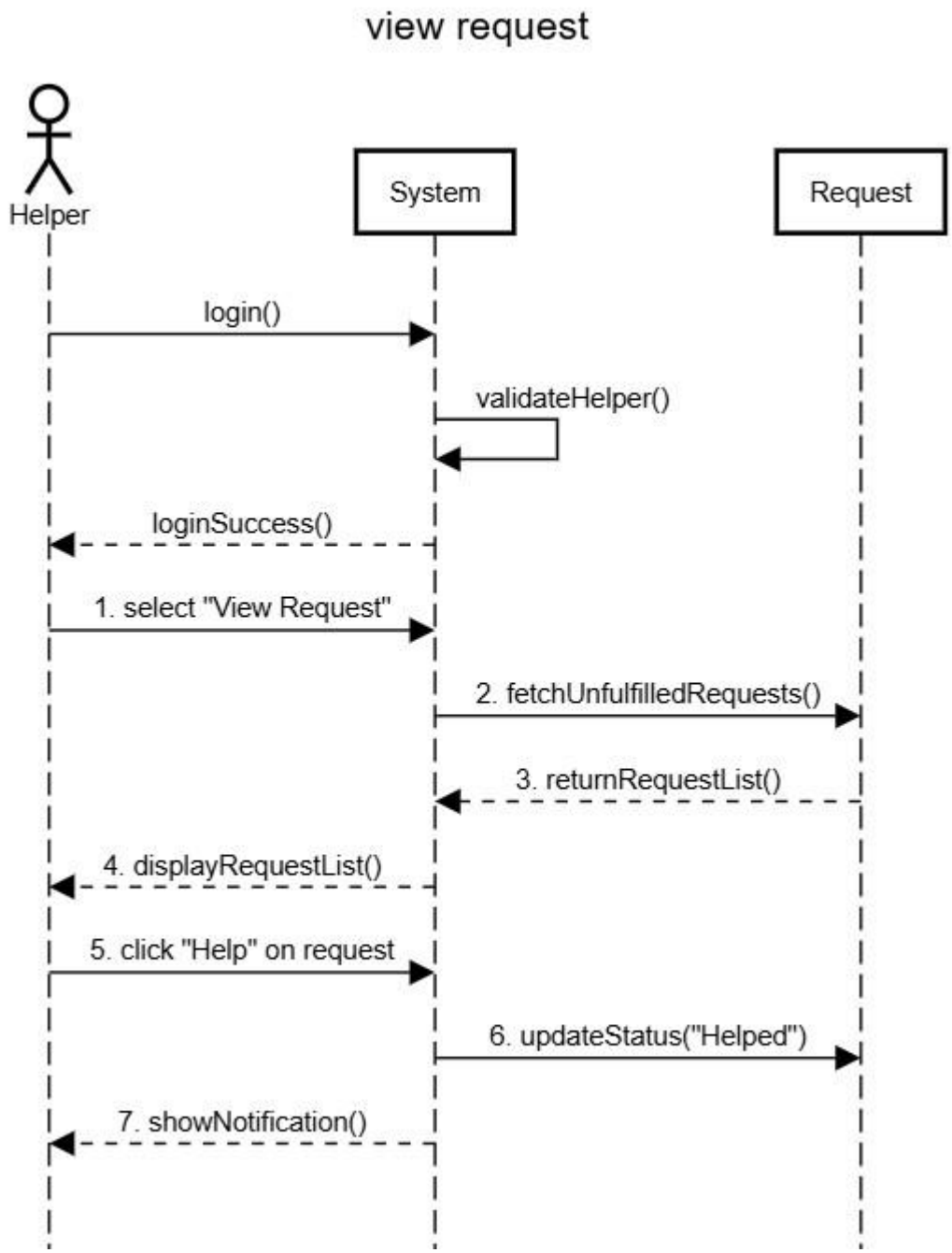
ANALYSIS MODEL:

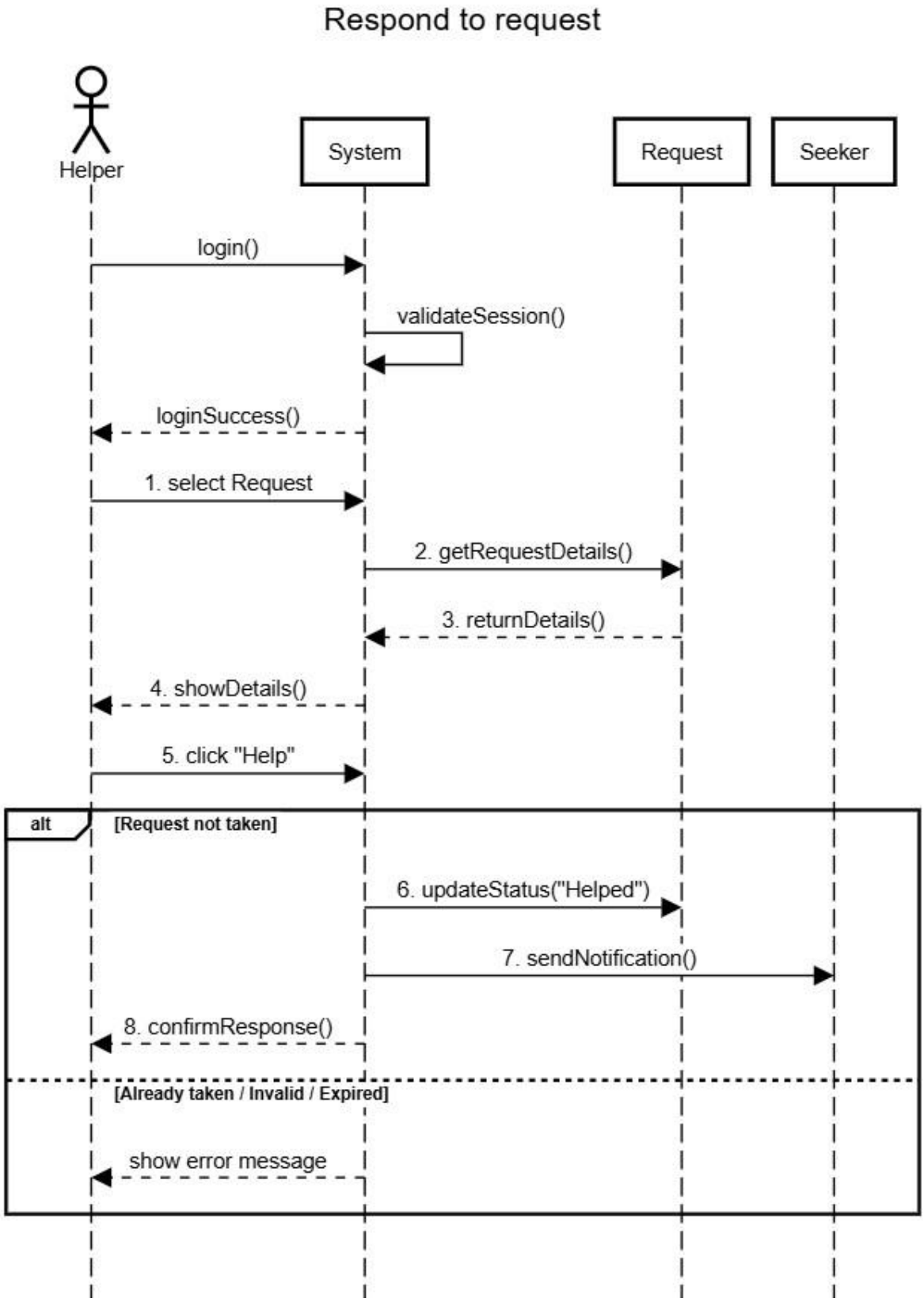
6.1.1.CLASS DIAGRAM:

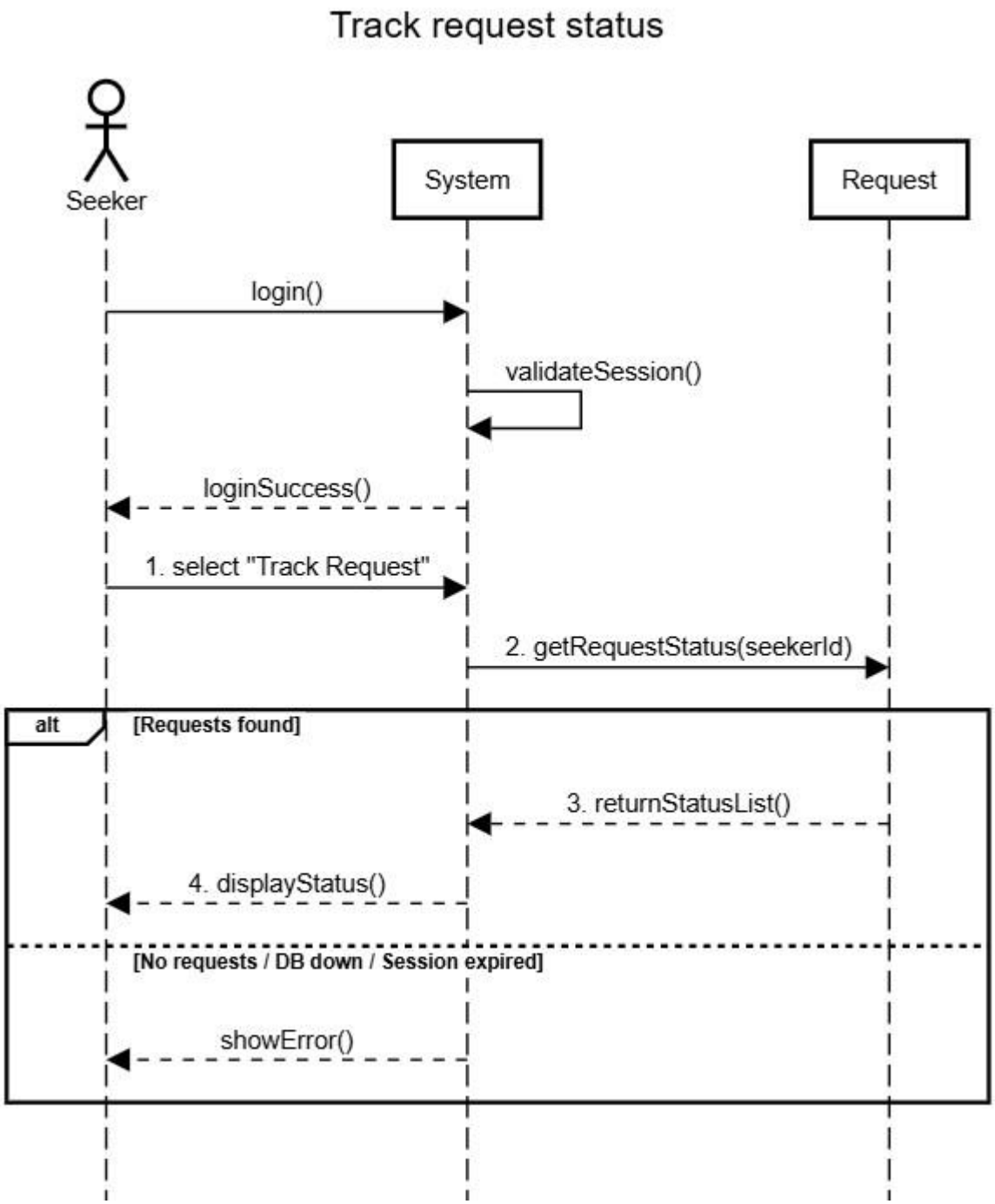


6.1.1.2.Sequence diagram:

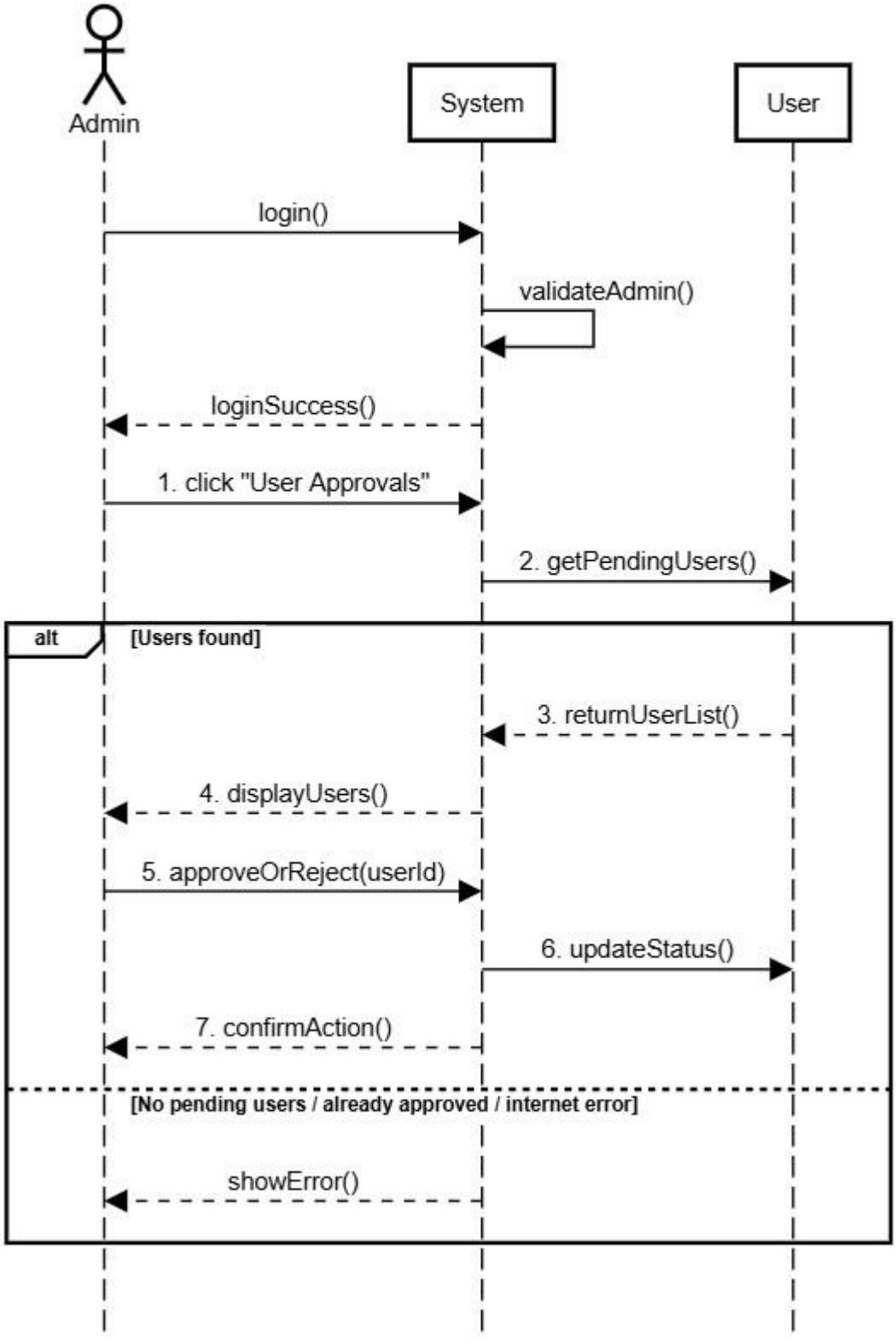

submit help request
Seeker
System
Request
login()
validateSeeker()
loginSuccess()
1. select "Submit Request"
2. displayRequestForm()
3. fillRequest(title, description)
4. validateFields()
alt
[Valid Input]
5. storeRequest()
6. status = "Pending"
7. showConfirmation()
[Title too long / Description blank / Not verified]
show error message

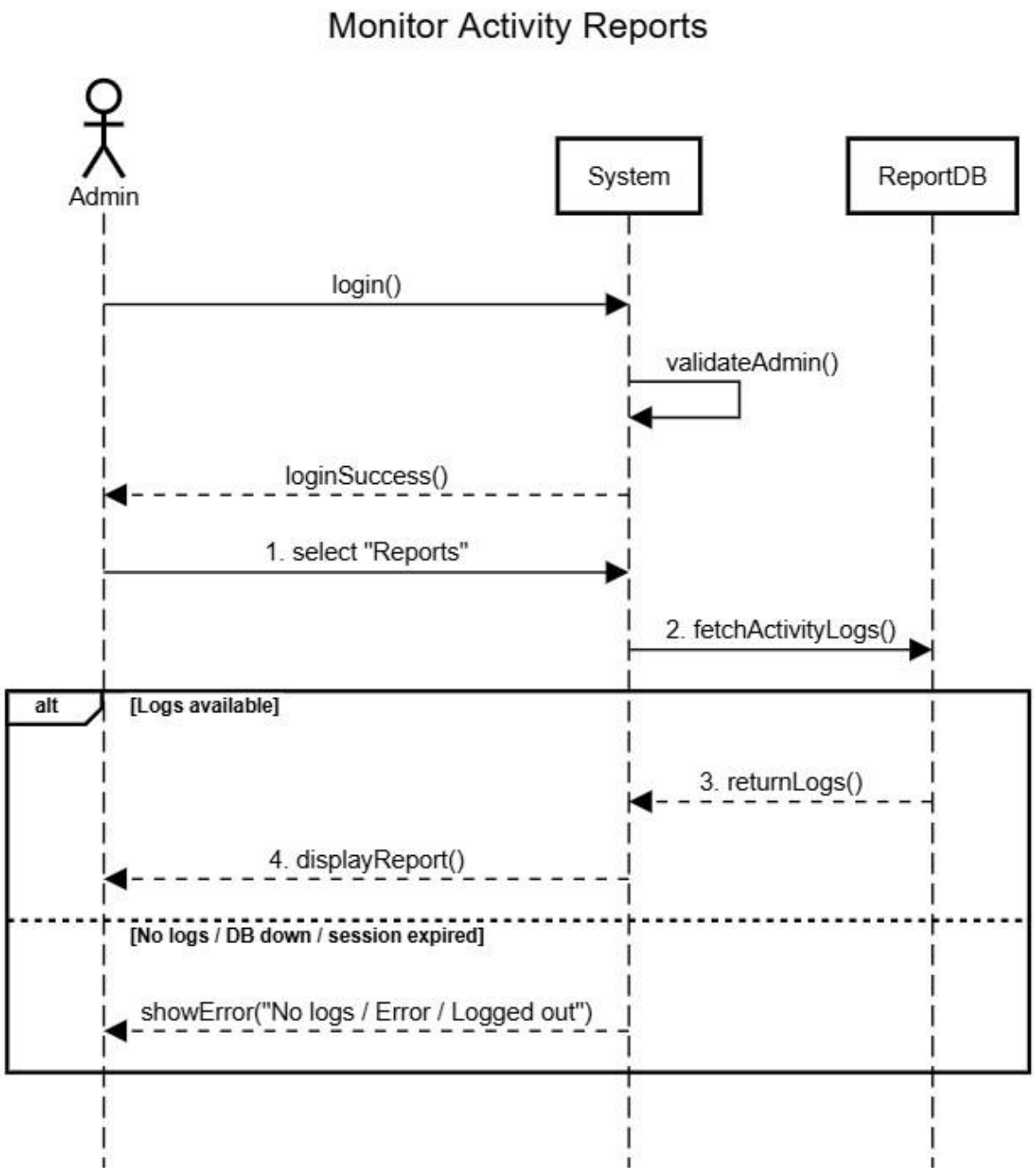

view request
Helper
System
Request
login()
validateHelper()
loginSuccess()
1. select "View Request"
2. fetchUnfulfilledRequests()
3. returnRequestList()
4. displayRequestList()
5. click "Help" on request
6. updateStatus("Helped")
7. showNotification()


Respond to request
Helper
System
Request
Seeker
login()
validateSession()
loginSuccess()
1. select Request
2. getRequestDetails()
3. returnDetails()
4. showDetails()
5. click "Help"
alt
[Request not taken]
6. updateStatus("Helped")
7. sendNotification()
8. confirmResponse()
[Already taken / Invalid / Expired]
show error message

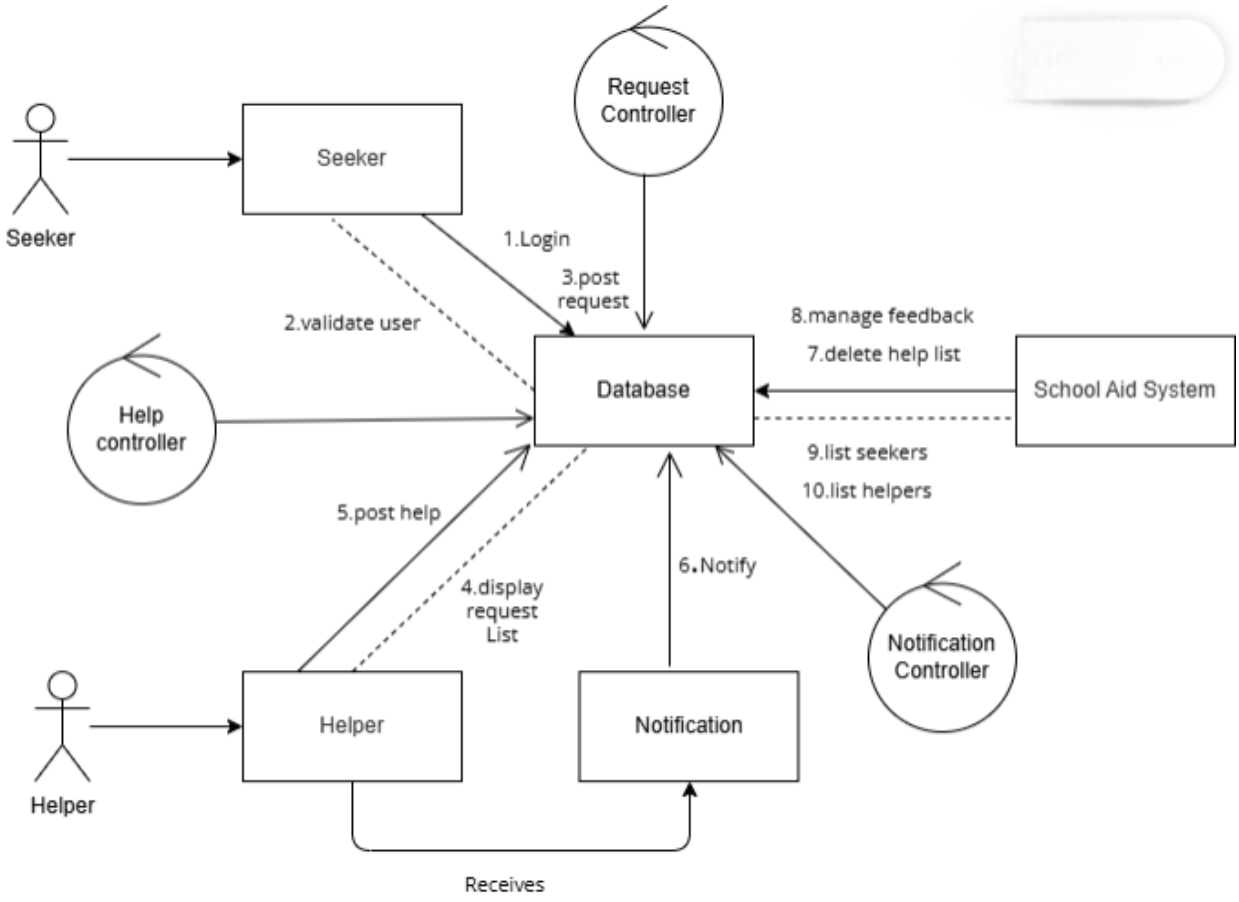

Track request status
Seeker
System
Request
login()
validateSession()
loginSuccess()
1. select "Track Request"
2. getRequestStatus(seekerId)
alt
[Requests found]
3. returnStatusList()
4. displayStatus()
[No requests / DB down / Session expired]
showError()

Approve New Users

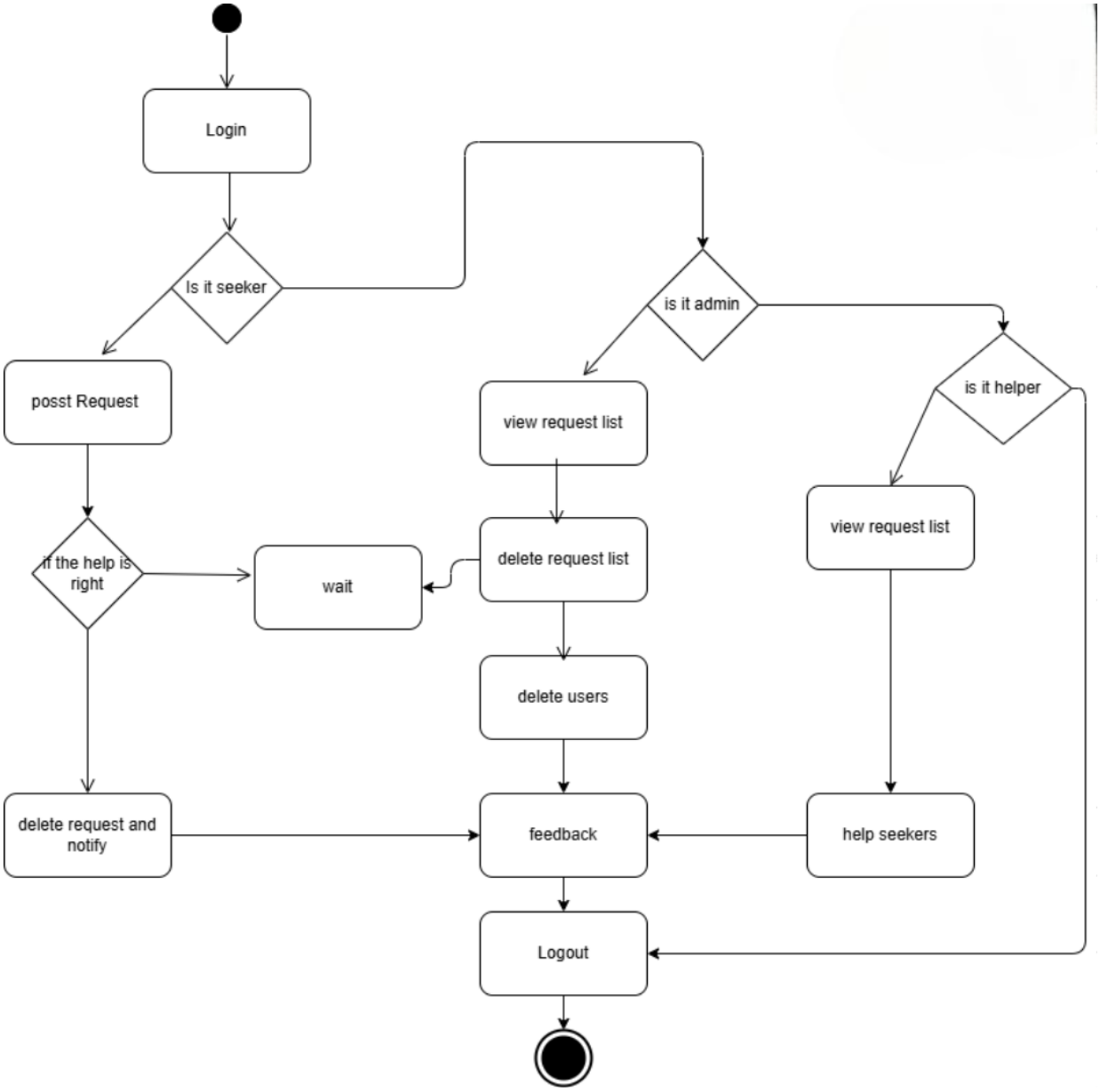



Monitor Activity Reports
Admin
System
ReportDB
login()
validateAdmin()
loginSuccess()
1. select "Reports"
2. fetchActivityLogs()
alt
[Logs available]
3. returnLogs()
4. displayReport()
[No logs / DB down / session expired]
showError("No logs / Error / Logged out")

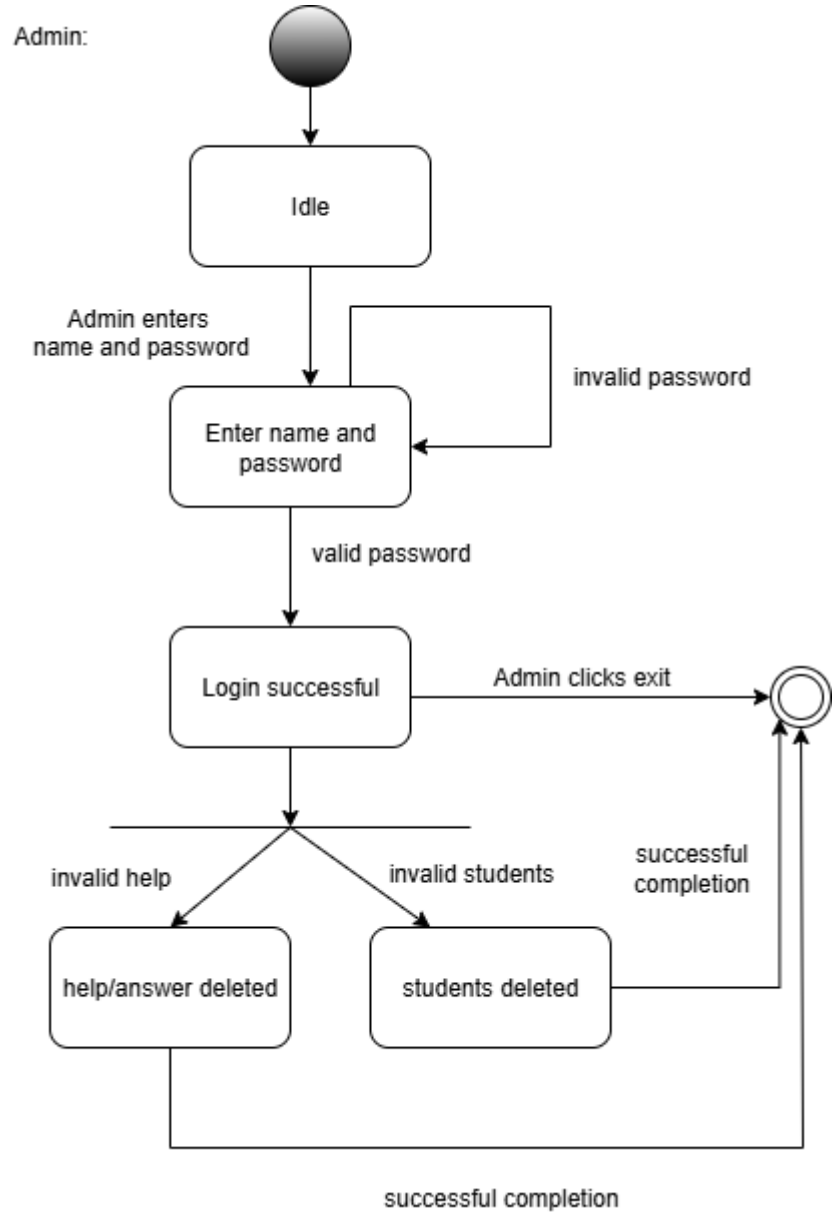
6.1.1.3.COLLABORATION DIAGRAM:

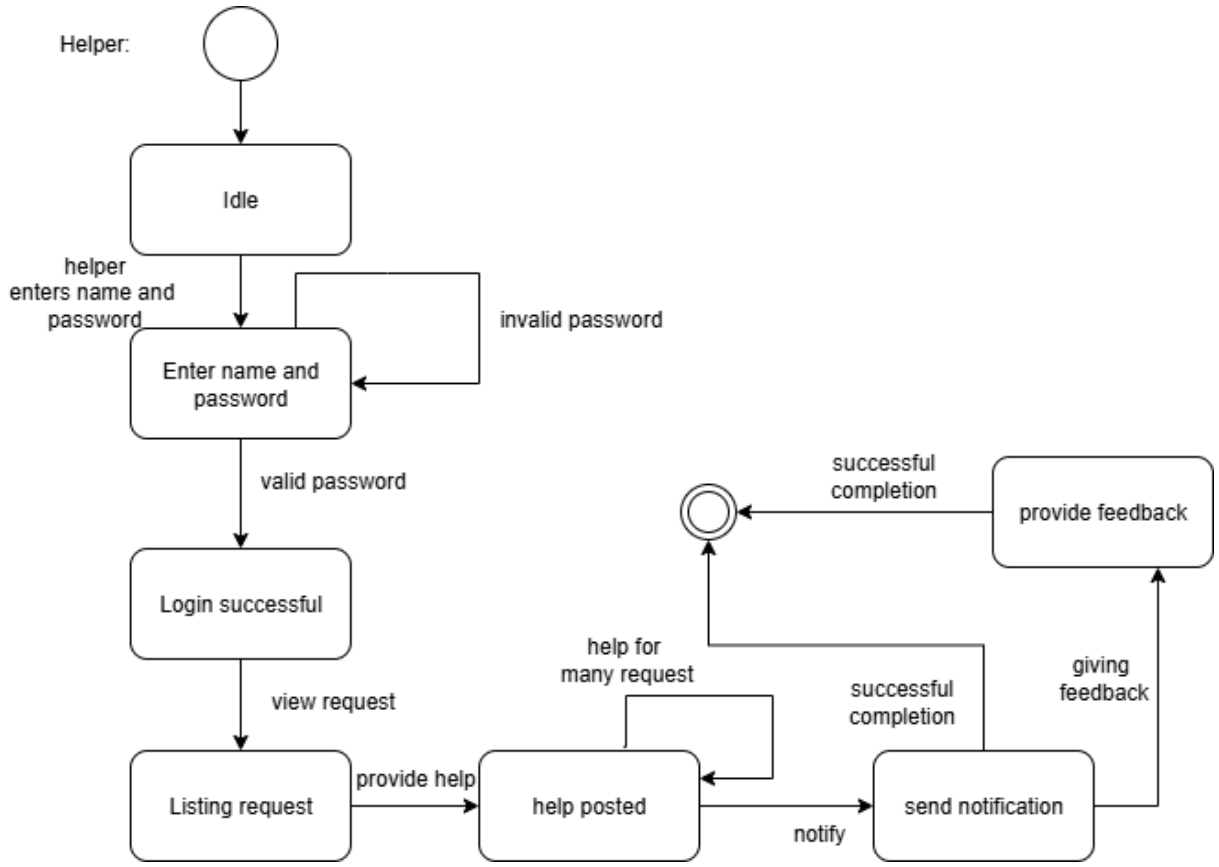


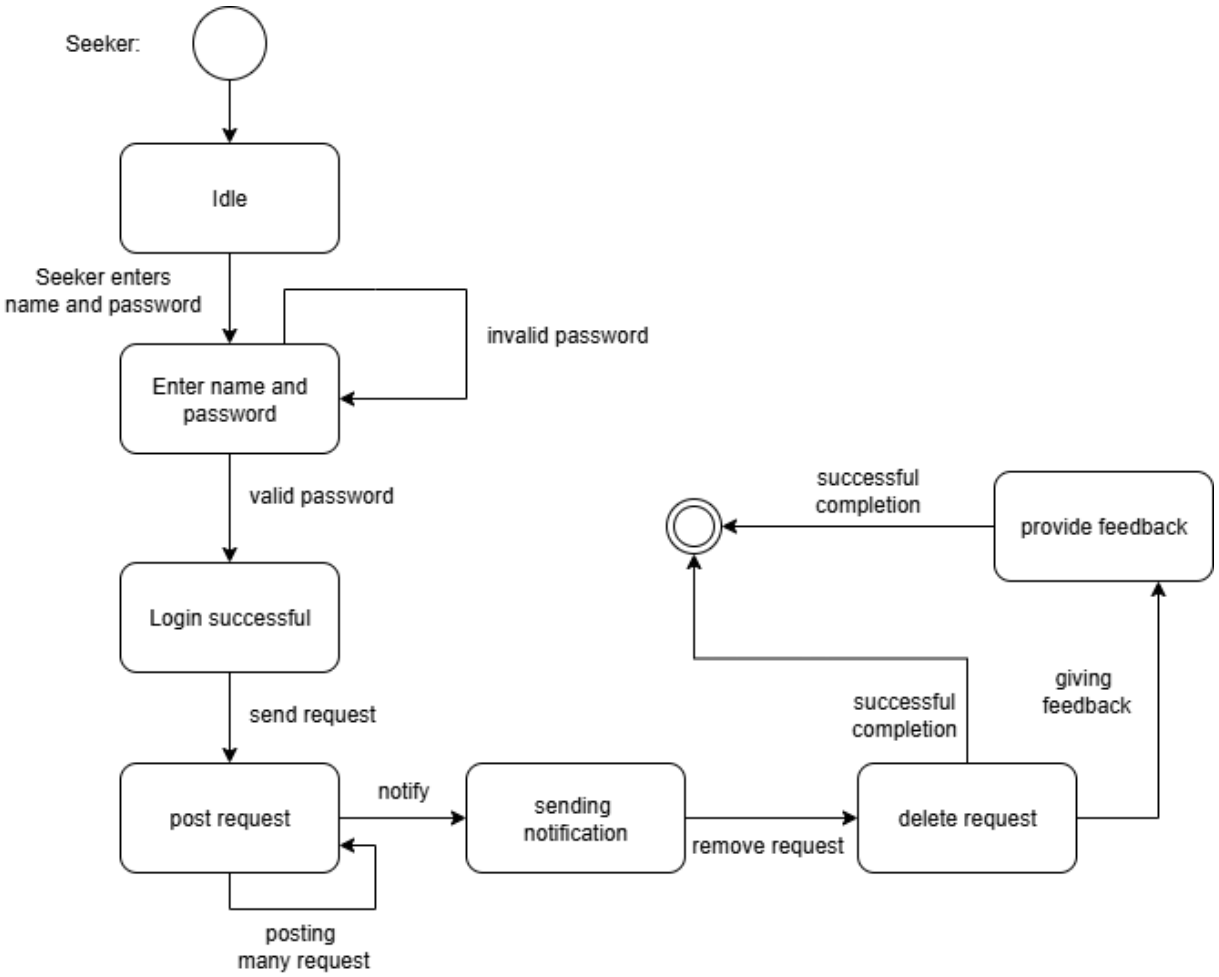
6.1.1.4.ACTIVITY DIAGRAM:



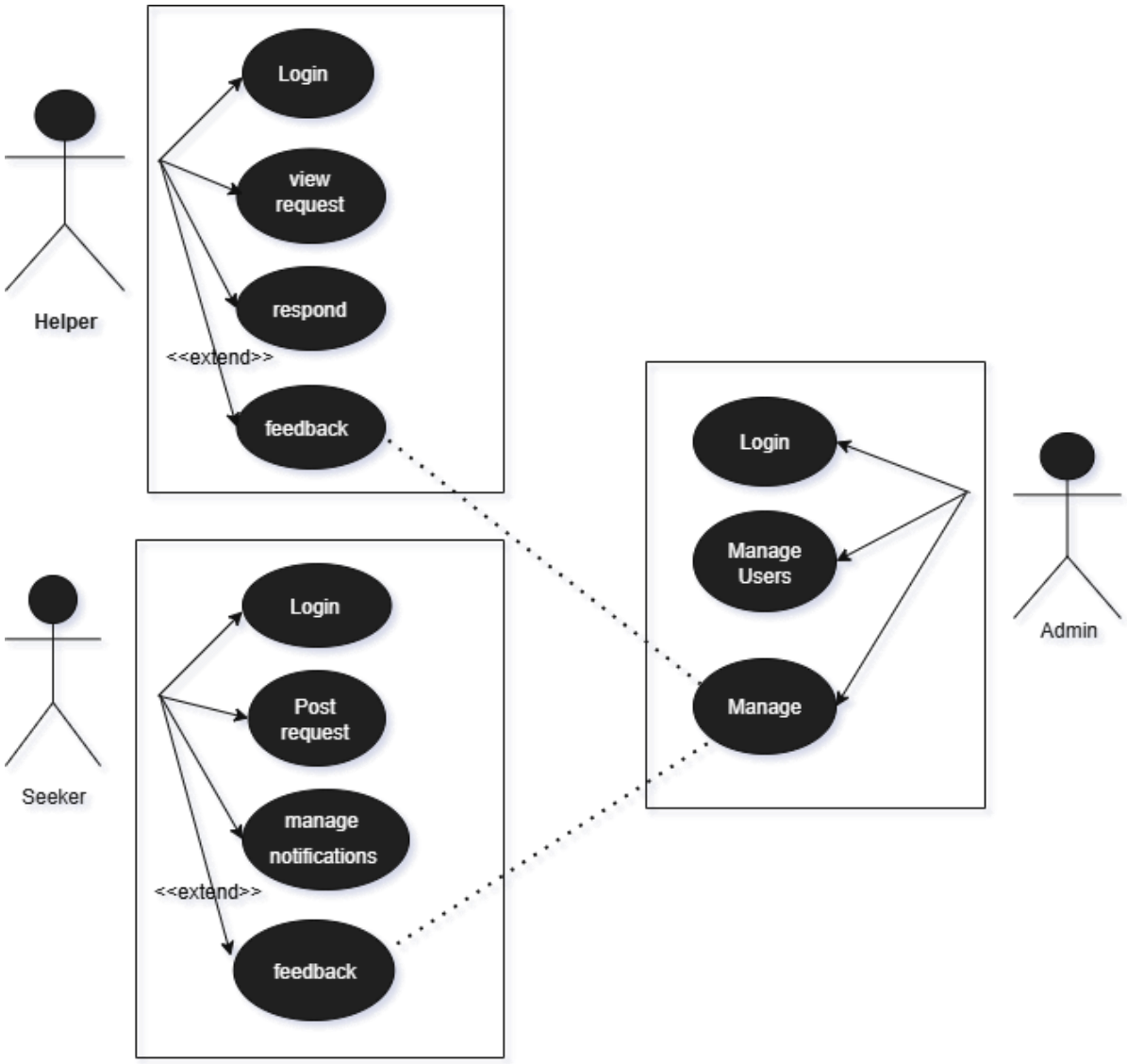
6.1.1.5.STATE CHART DIAGRAM:




Helper:
Idle
helper enters name and password
invalid password
Enter name and password
valid password
Login successful
view request
Listing request
provide help
help posted
help for many request
notify
send notification
successful completion
giving feedback
successful completion
provide feedback


Seeker:
Idle
Seeker enters
name and password
invalid password
Enter name and
password
valid password
Login successful
send request
post request
notify
sending
notification
remove request
delete request
successful
completion
provide feedback
successful
completion
giving
feedback
posting
many request

6.1.1.6.USECASE:



UseCase 1: Submit Help Request

Primary Actor: Seeker
Secondary Actor: Admin
Precondition: Seeker has logged in successfully.
Postcondition: Seeker's request is submitted and stored in the system as "Pending". Trigger: Seeker clicks the "Submit Request" button.

Main success scenario:

1. Seeker selects "Submit Request".
2. System displays the request form.
3. Seeker fills in the request details.
4. Seeker clicks "Submit".
5. System stores the request and marks it as "Pending".

6. A confirmation message is shown.

Exception:

1. If the description field is empty, System will inform “Description cannot be blank!”.
2. If the title exceeds 100 characters, System will inform “Title too long!”.
3. If the Seeker is not verified, System will inform “Only verified users can submit requests”.

UseCase 2: View Request

Primary Actor: Helper (Student/Teacher)
Secondary Actor: Admin
Precondition: Helper has logged in successfully.
Postcondition: Helper views a list of pending requests. Trigger:
Request is selected for viewing.

Main success scenario:

1. Helper selects “View Request”.
2. System displays all unfulfilled requests.
3. Helper selects a request and clicks the help option to respond.
4. System sends a notification and updates the request status as “Helped”.

Exception:

1. If the password is less than 8, then System will inform “Password is too long!”.
2. If the name contains special characters like @#$%^&*, System will inform “Name should not contain special characters”.
3. If the mobile number length is not equal to 10, then System will inform “Phone number should contain 10 numbers”.

UseCase 3: Respond to Request

Primary Actor: Helper (Student/Teacher)
Secondary Actor: Admin
Precondition: Helper has logged in successfully and selected a request.
Postcondition: Request is marked as "Helped" and assigned to the Helper. Trigger: Helper clicks the “Help” button on a request.

Main success scenario:

1. Helper selects a specific request.
2. System shows the request details.
3. Helper clicks the “Help” button.

4. System updates the status as “Helped” and sends notification to the Seeker.

Exception:

1. If the request is already taken, System will inform “This request has already been accepted”.
2. If Helper session expired, System will redirect to login page.
3. If request ID is missing, System will inform “Invalid request selected”.

Use Case 4: Track Request Status

Primary Actor: Seeker
Secondary Actor: Admin
Precondition: Seeker has logged in successfully.
Postcondition: Seeker sees the updated status of submitted requests. Trigger: Seeker selects "Track Request" from the menu.

Main success scenario:

1. Seeker clicks on “Track Request”.
2. System fetches all requests made by the Seeker.
3. System displays each request's current status (Pending / Helped).
4. Seeker reads the status.

Exception:

1. If Seeker has no previous requests, System will inform “No requests found!”.
2. If session expired, System will inform “Please login again!”.
3. If database connection fails, System will inform “Unable to fetch status. Try again later.”

Use Case 5: Approve New Users

Primary Actor: Admin
Secondary Actor: None
Precondition: Admin has logged in successfully.
Postcondition: User is either approved or rejected. Trigger: Admin clicks “User Approvals”.

Main success scenario:

1. Admin selects “User Approvals”.
2. System shows list of pending user registrations.
3. Admin clicks “Approve” or “Reject” for each user.
4. System updates user status accordingly.

Exception:

1. If no users are pending, System will inform “No pending users!”.
2. If Admin tries to approve an already approved user, System will inform “User already approved!”.
3. If internet fails, System will inform “Network error! Try again.”

Use Case 6: Monitor Activity Reports

Primary Actor: Admin
Secondary Actor: None
Precondition: Admin has logged in successfully.
Postcondition: System shows logs and activity history. Trigger: Admin clicks “Reports”.

Main success scenario:

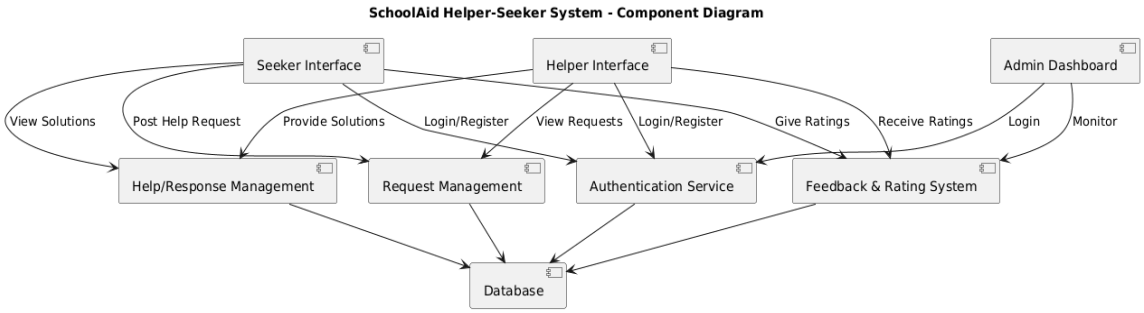
1. Admin selects “Reports”.
2. System fetches activity logs and usage stats.
3. System displays the report to Admin.
4. Admin reviews the report.

Exception:

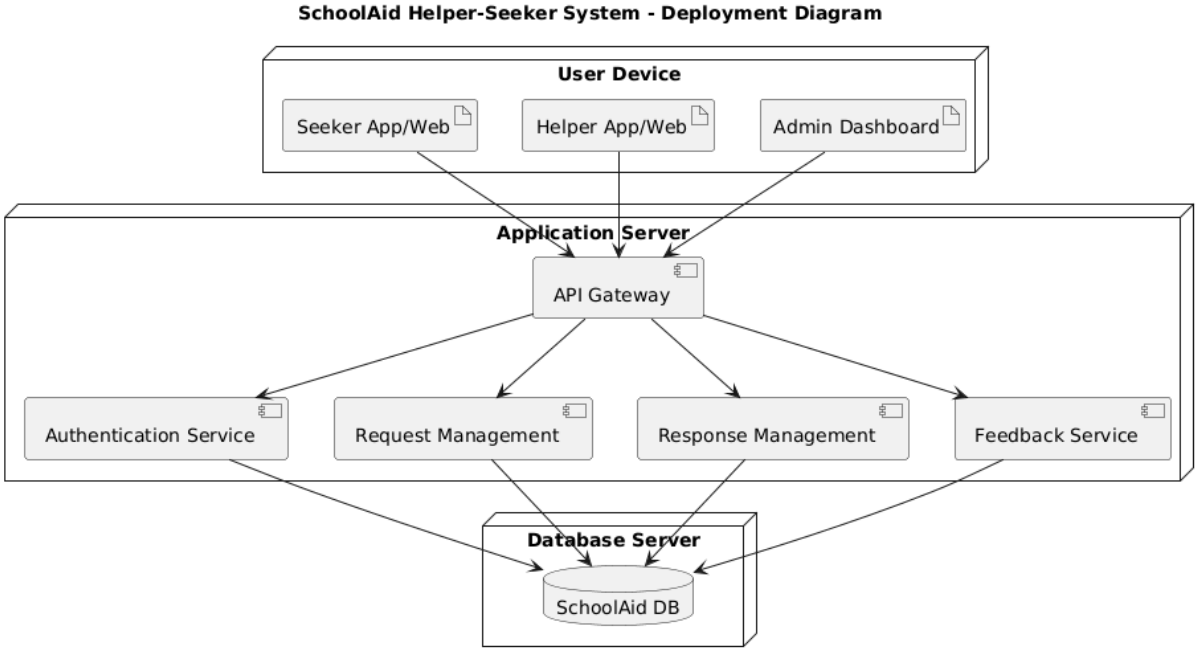
1. If no activity found, System shows “No logs available!”.
2. If database is down, System shows “Error retrieving reports!”.

3.If session is inactive, System logs out Admin.

6.1.1.7.Component Diagram



6.1.1.8.Deployment Diagram



SOURCE CODE:
main.py

```
from fastapi import FastAPI, Form, Request
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
import psycopg2
from psycopg2.extras import RealDictCursor

app = FastAPI()
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")

def connect_database():
    return psycopg2.connect(
        database="SchoolAid",
        user="postgres",
        host="localhost",
        password="kaviya",
        port="5432"
    )

@app.get("/")
def display_login(request: Request, error: str = ""):
    return templates.TemplateResponse("login.html", {"request": request, "error": error})

@app.post("/")
def login_user(request: Request, username: str = Form(...), password: str = Form(...)):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM loggedusers WHERE username=%s;", (username,))
    user = cur.fetchone()
    if user:
        if user["password"] == password:
            cur.close()
            conn.close()
            return templates.TemplateResponse("welcome.html", {"request": request})
        else:
            cur.close()
            conn.close()
            return templates.TemplateResponse("login.html", {"request": request, "error": "Invalid
Password!"})
    else:
        cur.execute("INSERT INTO loggedusers (username, password) VALUES (%s, %s);",
(username, password))
        conn.commit()
        cur.close()
```

```
        conn.close()
        return templates.TemplateResponse("welcome.html", {"request": request})

@app.get("/welcome")
def display_welcome(request: Request):
    return templates.TemplateResponse("welcome.html", {"request": request})

@app.get("/admin")
def display_admin(request: Request):
    return templates.TemplateResponse("admin.html", {"request": request})

@app.post("/admin")
def login_admin(request: Request, name: str = Form(...), password: str = Form(...)):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM admin WHERE adminname=%s;", (name,))
    user = cur.fetchone()
    if user:
        if user["password"] == password:
            cur.close()
            conn.close()
            return RedirectResponse(url="/adminop", status_code=303)
        else:
            cur.close()
            conn.close()
            return templates.TemplateResponse("admin.html", {"request": request, "error": "Invalid
Password!"})
    else:
        cur.close()
        conn.close()
        return templates.TemplateResponse("admin.html", {"request": request, "error": "You don't
have admin Registration!"})

@app.get("/adminop")
def display_adminoperation(request: Request):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM seekers;")
    seekers = cur.fetchall()
    cur.close()
    conn.close()
    return templates.TemplateResponse("adminop.html", {"request": request, "seekers":
seekers})

@app.get("/viewseekers")
def view_seekers(request: Request):
    conn = connect_database()
```

```
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM seekers;")
    seekers = cur.fetchall()
    cur.close()
    conn.close()
    return templates.TemplateResponse("viewseekers.html", {"request": request, "seekers":
seekers})

@app.post("/delete_seeker/{seeker_id}")
def delete_seeker(seeker_id: int):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("DELETE FROM helps WHERE seeker_id=%s;", (seeker_id,))
    cur.execute("DELETE FROM seekers WHERE id=%s;", (seeker_id,))
    conn.commit()
    cur.close()
    conn.close()
    return RedirectResponse(url="/viewseekers", status_code=303)

@app.get("/viewhelpers")
def view_helpers(request: Request):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM helps;")
    helpers = cur.fetchall()
    cur.close()
    conn.close()
    return templates.TemplateResponse("viewhelpers.html", {"request": request, "helpers":
helpers})

@app.post("/delete_helper/{help_id}")
def delete_helper(help_id: int):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("DELETE FROM helps WHERE id=%s;", (help_id,))
    conn.commit()
    cur.close()
    conn.close()
    return RedirectResponse(url="/viewhelpers", status_code=303)

@app.get("/viewfeedbacks")
def view_feedbacks(request: Request):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM feedback;")
    feedbacks = cur.fetchall()
    cur.close()
```

```
    conn.close()
    return templates.TemplateResponse("viewfeedbacks.html", {"request": request, "feedbacks":
feedbacks})

@app.post("/delete_feedback/{feedback_id}")
def delete_feedback(feedback_id: int):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("DELETE FROM feedback WHERE id=%s;", (feedback_id,))
    conn.commit()
    cur.close()
    conn.close()
    return RedirectResponse(url="/viewfeedbacks", status_code=303)

@app.get("/feedback")
def feedback_form(request: Request):
    return templates.TemplateResponse("feedback.html", {"request": request, "message": ""})

@app.post("/feedback")
def submit_feedback(request: Request, name: str = Form(...), email: str = Form(...), rating: int =
Form(...), description: str = Form(...)):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("INSERT INTO feedback (name, email, rating, description) VALUES (%s, %s,
%s, %s);", (name, email, rating, description))
    conn.commit()
    cur.close()
    conn.close()
    return templates.TemplateResponse("feedback.html", {"request": request, "message":
"Thank you for your feedback!"})

@app.get("/seeker")
def display_seeker(request: Request, message: str = ""):
    return templates.TemplateResponse("seeker.html", {"request": request, "message":
message})

@app.post("/seeker")
def submit_seeker(request: Request, name: str = Form(...), help: str = Form(...)):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("SELECT id FROM seekers WHERE name=%s;", (name,))
    existing = cur.fetchone()
    if existing:
        cur.execute("UPDATE seekers SET help_needed=%s, status='pending' WHERE id=%s;",
(help, existing[0]))
    else:
```

```
        cur.execute("INSERT INTO seekers (name, help_needed, status) VALUES (%s, %s,
%s);", (name, help, "pending"))
    conn.commit()
    cur.close()
    conn.close()
    return templates.TemplateResponse("seeker.html", {"request": request, "message": "Your
request has been submitted successfully!"})

@app.get("/helper")
def display_helper(request: Request):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT * FROM seekers WHERE status='pending';")
    seekers = cur.fetchall()
    cur.execute("SELECT * FROM helps;")
    helps = cur.fetchall()
    helps_dict = {}
    for h in helps:
        helps_dict.setdefault(h["seeker_id"], []).append(h)
    cur.close()
    conn.close()
    return templates.TemplateResponse("helper.html", {"request": request, "seekers": seekers,
"helps": helps_dict})

@app.post("/help/{seeker_id}")
def help_seeker(seeker_id: int, request: Request, helperName: str = Form(...), solution: str =
Form(...)):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("INSERT INTO helps (seeker_id, helper_name, solution) VALUES (%s, %s,
%s);", (seeker_id, helperName, solution))
    conn.commit()
    cur.close()
    conn.close()
    return display_helper(request)

@app.get("/profile")
def profile_login_page(request: Request, error: str = ""):
    return templates.TemplateResponse("profile.html", {"request": request, "error": error})

@app.post("/profile")
def profile_login(request: Request, name: str = Form(...), password: str = Form(...)):
    conn = connect_database()
    cur = conn.cursor(cursor_factory=RealDictCursor)
    cur.execute("SELECT id, name FROM seekers WHERE name=%s AND password=%s
AND status='pending';", (name, password))
    seeker = cur.fetchone()
```

```
    if not seeker:
        cur.close()
        conn.close()
        return templates.TemplateResponse("profile.html", {"request": request, "error": "Invalid
Name/Password or Already Satisfied!"})
    cur.execute("SELECT id, helper_name, solution FROM helps WHERE seeker_id = %s;",
(seeker["id"],))
    helps = cur.fetchall()
    cur.close()
    conn.close()
    return templates.TemplateResponse("profile.html", {"request": request, "name":
seeker["name"], "helps": helps, "error": None})

@app.post("/satisfied/{help_id}")
def mark_satisfied(help_id: int):
    conn = connect_database()
    cur = conn.cursor()
    cur.execute("SELECT seeker_id FROM helps WHERE id = %s;", (help_id,))
    seeker_row = cur.fetchone()
    if seeker_row:
        seeker_id = seeker_row[0]
        cur.execute("DELETE FROM helps WHERE seeker_id=%s;", (seeker_id,))
        cur.execute("UPDATE seekers SET status='satisfied' WHERE id=%s;", (seeker_id,))
        conn.commit()
    cur.close()
    conn.close()
    return RedirectResponse(url="/profile", status_code=303)

login.html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Login</title>
    <link rel="stylesheet" href="{{ url_for('static', path='external.css') }}">
</head>
<body style="background-image: url('https://w0.peakpx.com/wallpaper/841/88/HD-wallpaper-
magical-book-reading-open-stories-book.jpg'); background-size: cover; background-repeat: no-
repeat; background-attachment: fixed;">
     <h1 style="text-align:center;color: whitesmoke;">School Aid Management</h1><br>
    <marquee behavior="scroll" direction="left" scrollamount="15" class="class1" >SchoolAid
Management System connects seekers with helpers, enabling students to get academic support
and share resources. It fosters collaborative learning by allowing knowledge exchange in a
structured, easy-to-use platform.</marquee>
    <br><br>
    <div class="container">
    <form method="post" action="/">
        <table>
```

```
        <tr>
          <td>Name</td>
          <td><input type="text" name="username" placeholder="Enter Name"
required></td>
        </tr>
        <tr>
          <td>Password</td>
          <td><input type="password" name="password" placeholder="Enter Password"
required></td>
        </tr>
        <tr>
          <td></td>
          <td>{% if error %}
      <p style="color: red; text-align:center;">{{ error }}</p>
   {% endif %}
   </td>
        </tr>
        <tr>
          <td><button onclick="window.location.href='/'" class="stylish-
btn">Refresh</button></td>
          <td><button type="submit">Login</button></td>
        </tr>
      </table>
   </form>
   </div>
</body>
</html>
Welcome.html
<!DOCTYPE html>
<html>
<head>
  <title>Seeker Profile</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #33c6f0;      /* 🔵 Blue background */
      display: flex;
      justify-content: center;
      align-items: flex-start;
      padding: 40px;
      margin: 0;
    }
    .card {
      background: #fff;         /* White card for contrast */
      padding: 30px 40px;
      border-radius: 12px;
      box-shadow: 0 4px 12px rgba(0,0,0,0.1);
```

```
  max-width: 650px;
  width: 100%;
}
h2 {
  margin-top: 0;
  text-align: center;
  color: #333;
}
h3 {
  color: #555;
  margin-top: 20px;
  margin-bottom: 10px;
  text-align: center;
}
ul {
  list-style: none;
  padding: 0;
}
li {
  background: #f8f9fa;
  margin-bottom: 12px;
  padding: 12px;
  border-radius: 8px;
  line-height: 1.4;
  display: flex;
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
}
li b {
  color: #0077cc;
}
a {
  color: #0077cc;
  text-decoration: none;
  word-break: break-word;
  margin-right: 12px;
}
a:hover {
  text-decoration: underline;
}
.error {
  color: red;
  text-align: center;
  margin-bottom: 10px;
}
form {
```

```
      display: flex;
      flex-direction: column;
      align-items: center;
    }
    input {
      width: 80%;
      padding: 10px;
      margin: 8px 0;
      border: 1px solid #ccc;
      border-radius: 6px;
    }
    button, .back-btn {
      padding: 8px 16px;
      background: #2196f3;
      color: white;
      border: none;
      border-radius: 6px;
      cursor: pointer;
      text-decoration: none;
      font-size: 14px;
      margin-top: 10px;
    }
    button:hover, .back-btn:hover {
      background: #1976d2;
    }
    .satisfy-btn {
      background: #4caf50;
    }
    .satisfy-btn:hover {
      background: #388e3c;
    }
    .top-actions {
      display: flex;
      justify-content: space-between;
      margin-bottom: 20px;
    }
  </style>
</head>
<body>
  <div class="card">
    {% if not helps %}
      <h2>Seeker Profile Login</h2>
      <form method="post" action="/profile">
        <input type="text" name="name" placeholder="Enter Name" required>
        <input type="password" name="password" placeholder="Enter Password" required>
        {% if error %}
          <p class="error">{{ error }}</p>
```

```
      {% endif %}
      <button type="submit">Login</button>
      <a href="/welcome" class="back-btn">← Back</a>
    </form>
  {% else %}
    <div class="top-actions">
      <h2>Welcome, {{ name }}!</h2>
      <!-- Back Button -->
      <a href="/home" class="back-btn">← Back</a>
    </div>

    <h3>Helps sent to you:</h3>
    <ul>
      {% for h in helps %}
        <li>
          <div>
            <b>{{ h.helper_name }}</b>:
            <a href="{{ h.solution }}" target="_blank">
              {{ h.solution }}
            </a>
          </div>
          <!-- Satisfy button sends POST to /satisfied/<id> -->
          <form method="post" action="/satisfied/{{ h.id }}" style="margin:0;">
            <button type="submit" class="satisfy-btn">Satisfied</button>
          </form>
        </li>
      {% endfor %}
    </ul>
  {% endif %}
 </div>
</body>
</html>
```

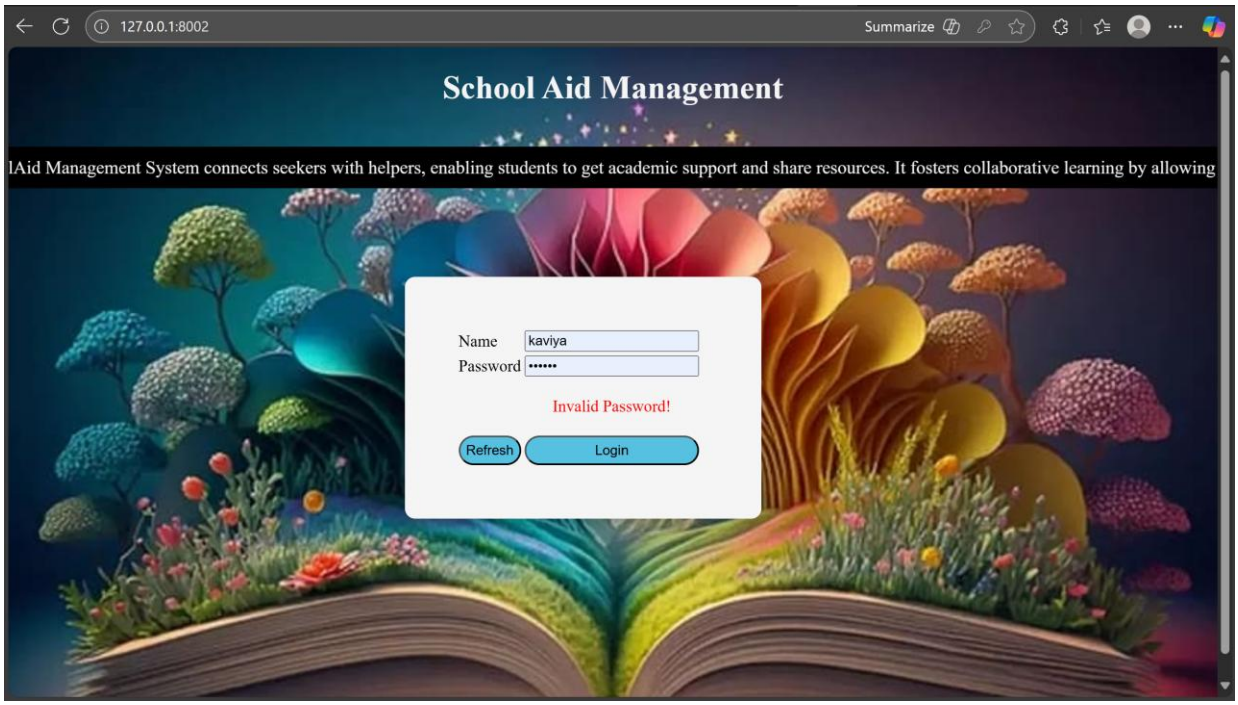
Seeker.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Seeker Form</title>
  <link rel="stylesheet" href="{{ url_for('static', path='external.css') }}">
</head>
<body>
  <h1 style="text-align: center; color: whitesmoke; background: black; padding: 10px;">
    Seeker Request Form
  </h1>

  <div style="width: 50%; margin: auto; margin-top: 50px;">
    <form method="post" action="/seeker">
```

```
        <label for="name">Your Name:</label><br>
        <input type="text" id="name" name="name" placeholder="Enter your name"
required><br><br>

        <label for="help">What help do you need?</label><br>
        <textarea id="help" name="help" placeholder="Describe the help needed" rows="4"
cols="50" required></textarea><br><br>

        <button type="submit">Submit Request</button>
          <td><button onclick="window.location.href='/welcome'" class="stylish-
btn">Back</button></td>
      </form>
   </div>

   {% if message %}
      <p style="text-align: center; color: green; font-weight: bold;">
         {{ message }}
      </p>
   {% endif %}
</body>
</html>
```

OUTPUT:

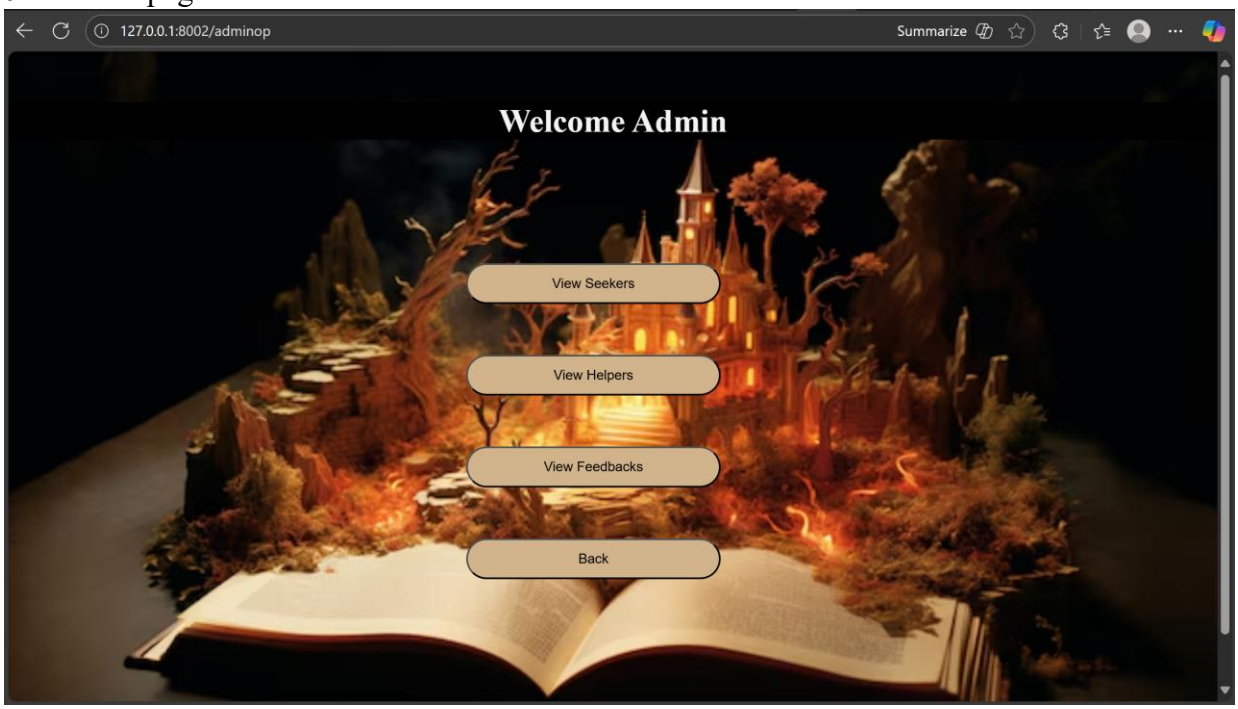
6.1.Login page



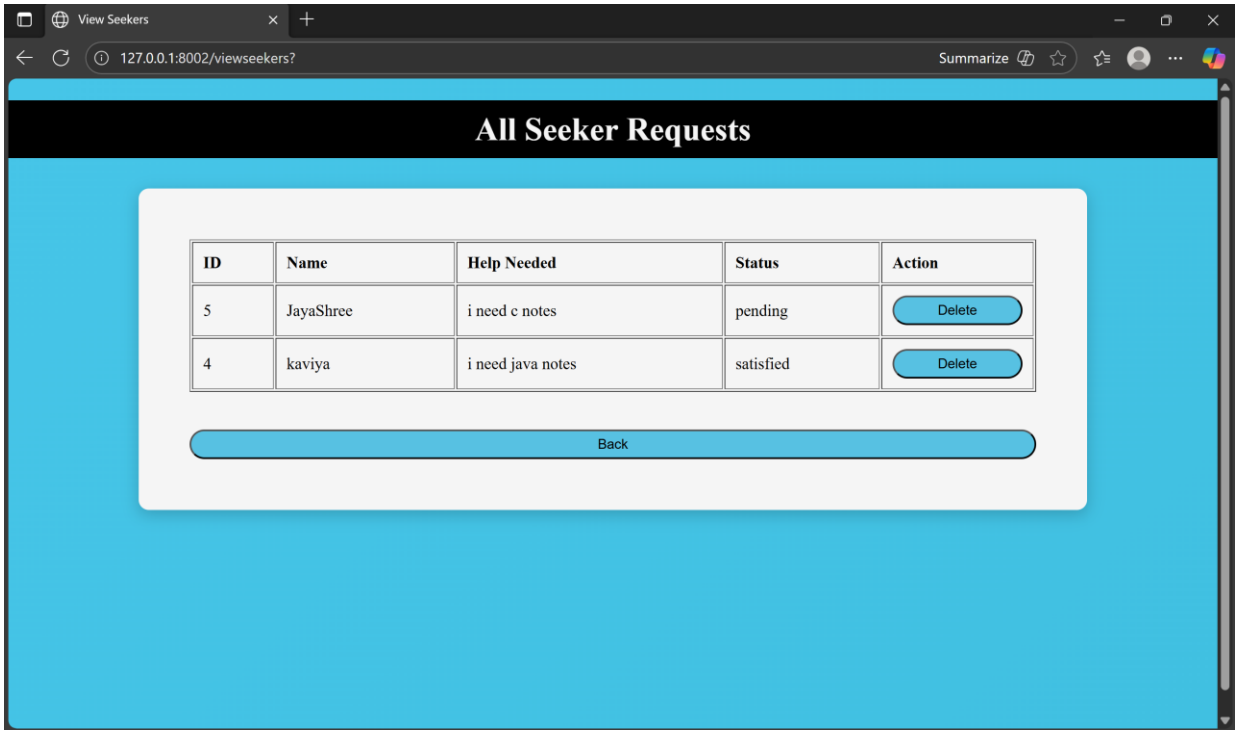
6.2.home page

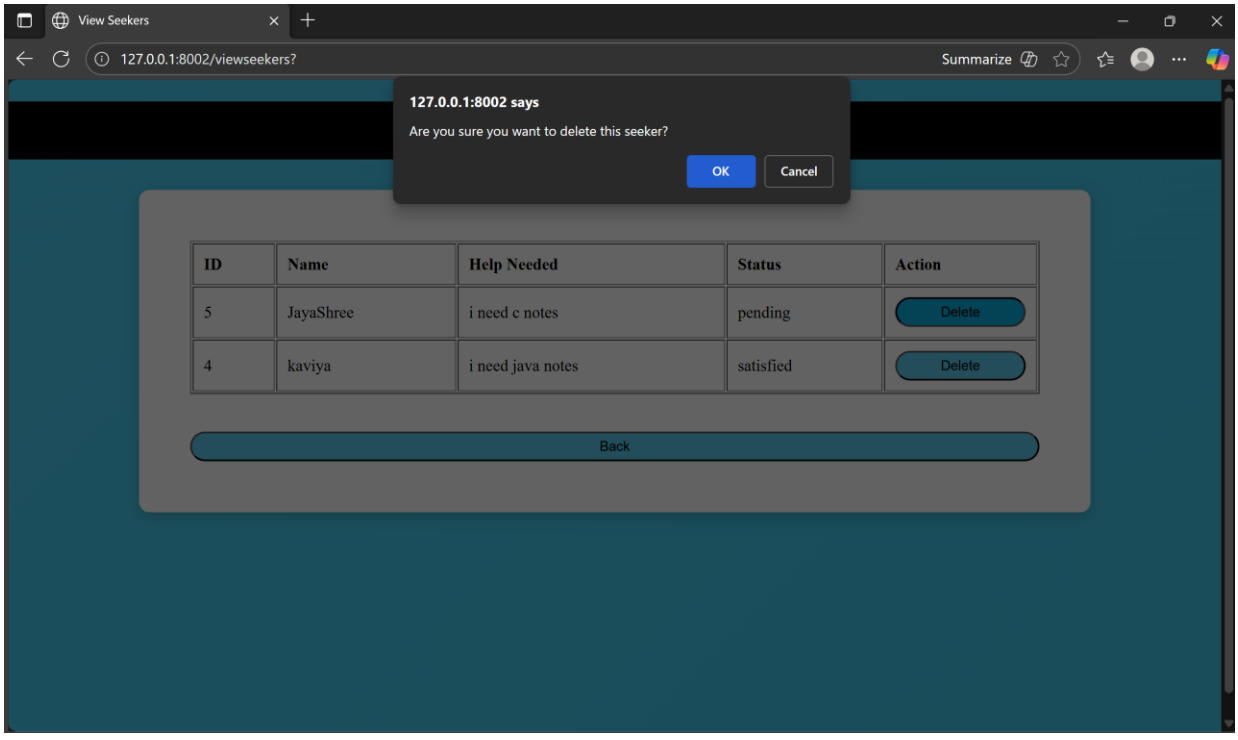
6.3.Admin login page

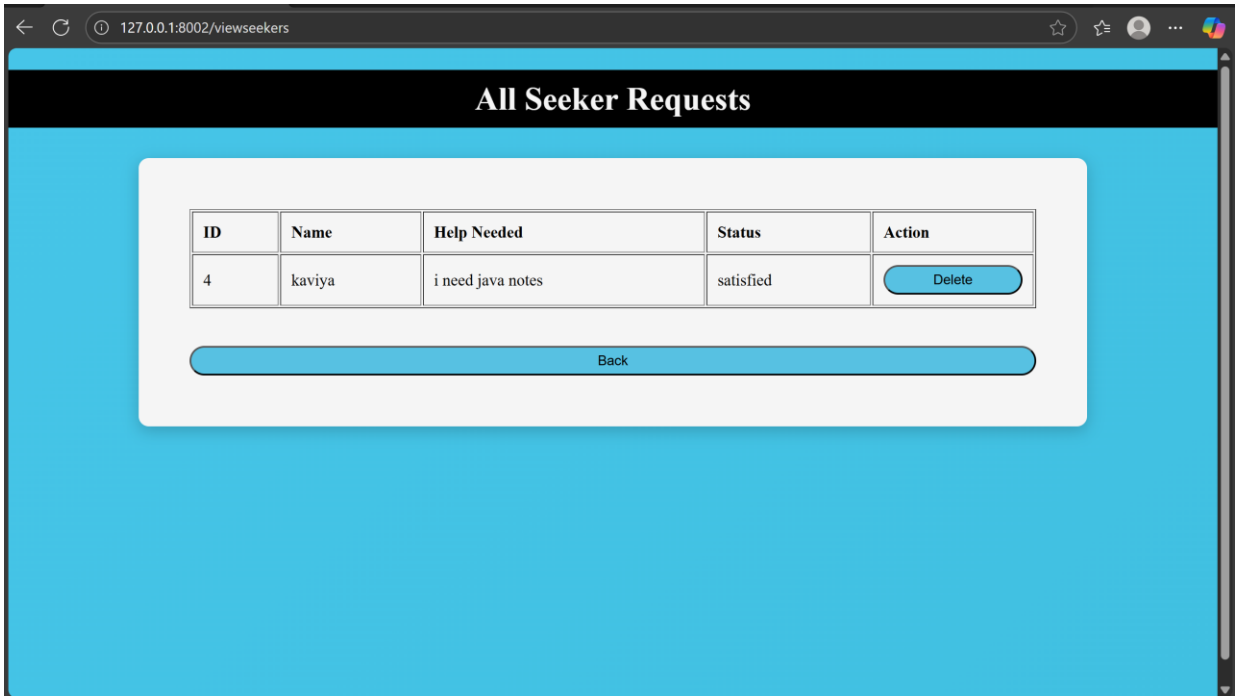
6.4.Admin page



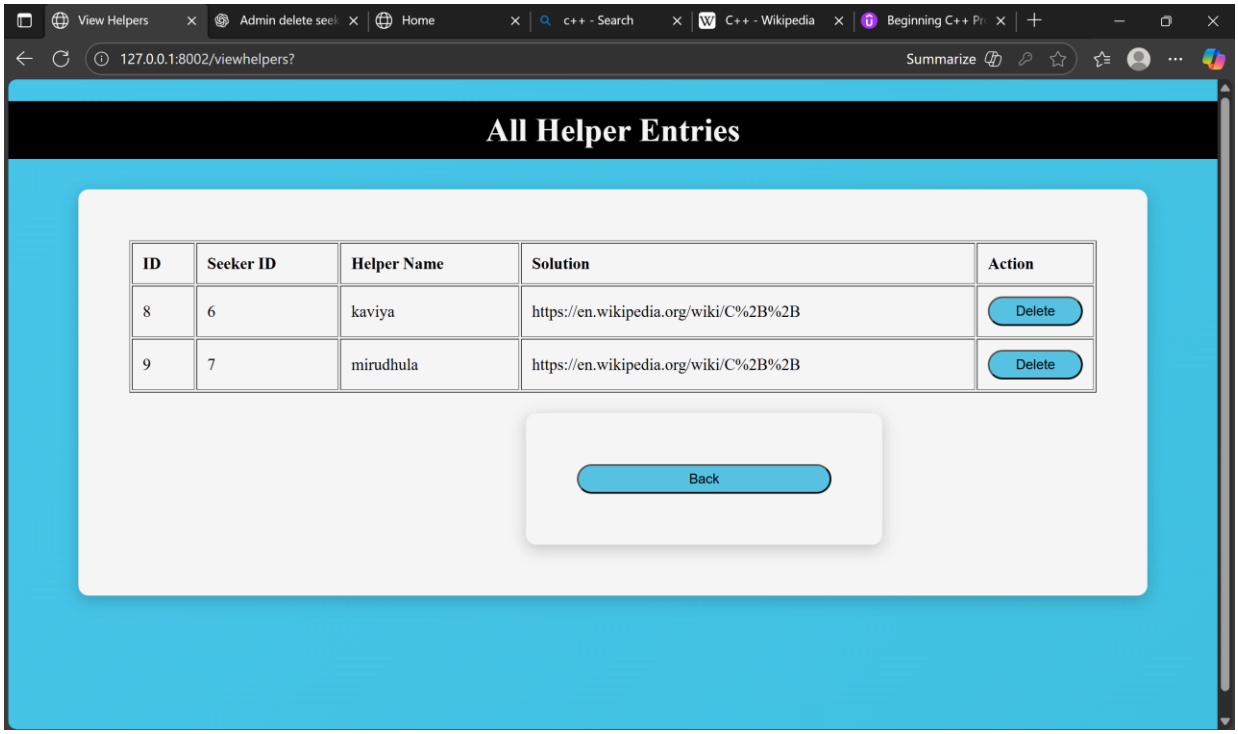
6.5.View Seekers

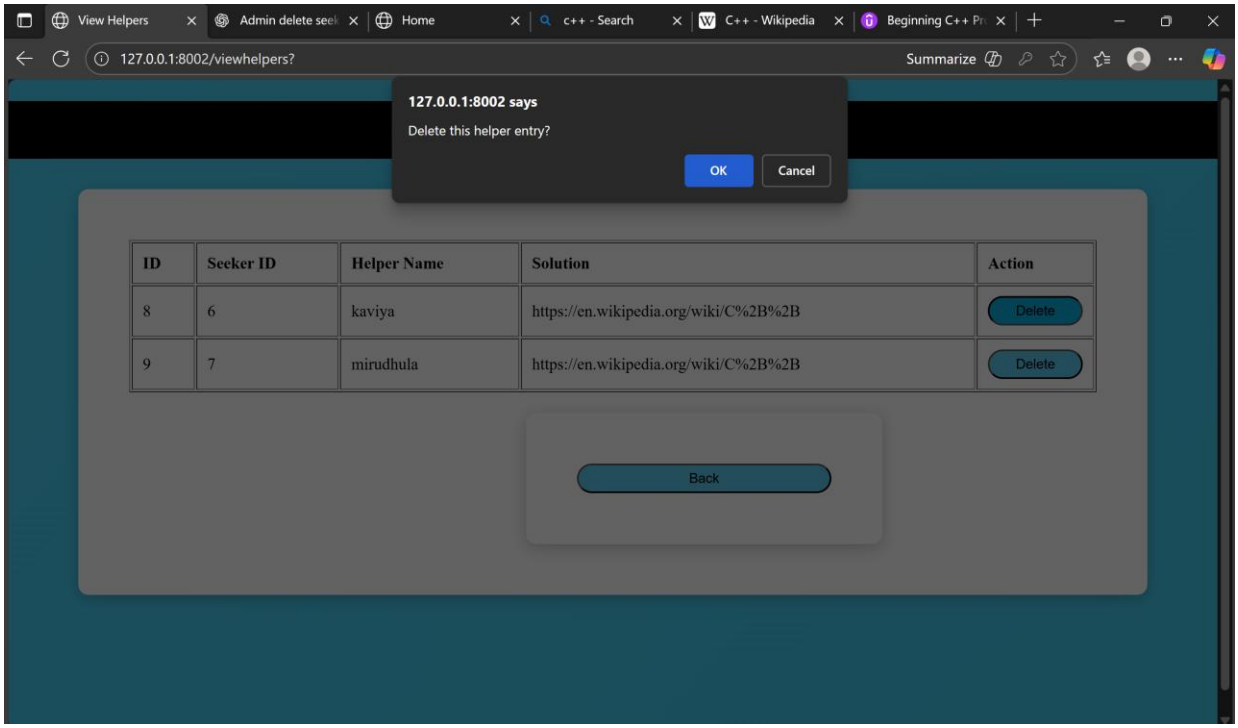




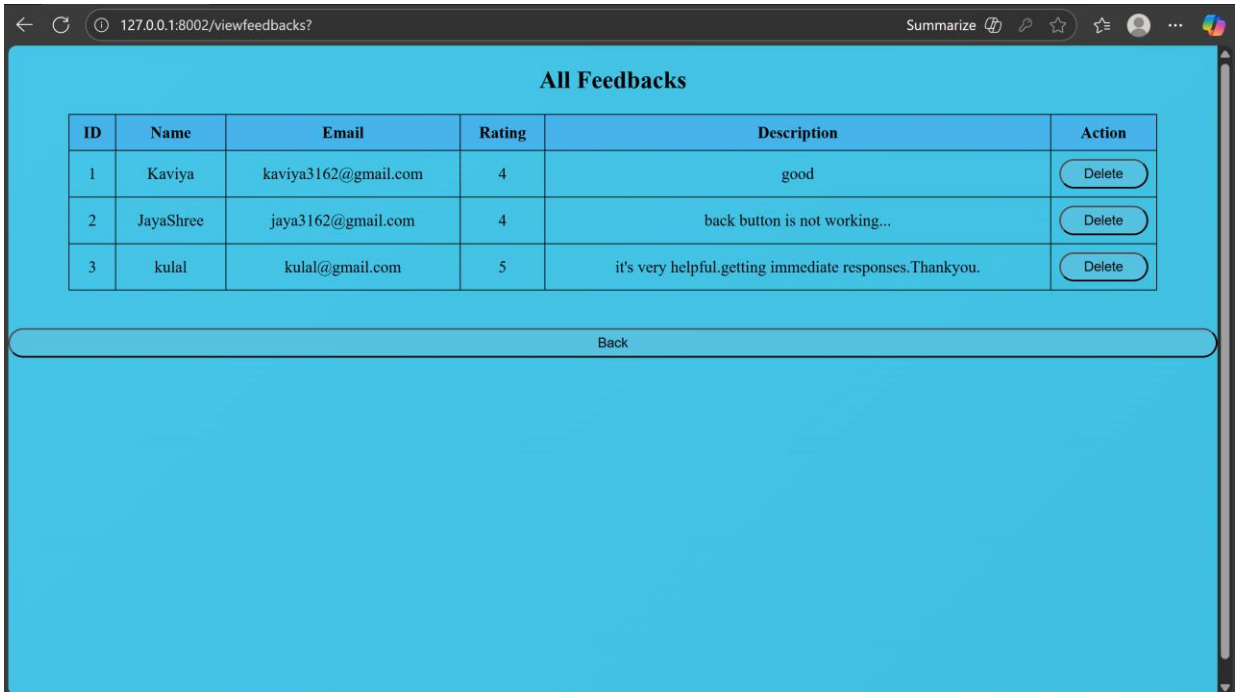


6.6.view helpers

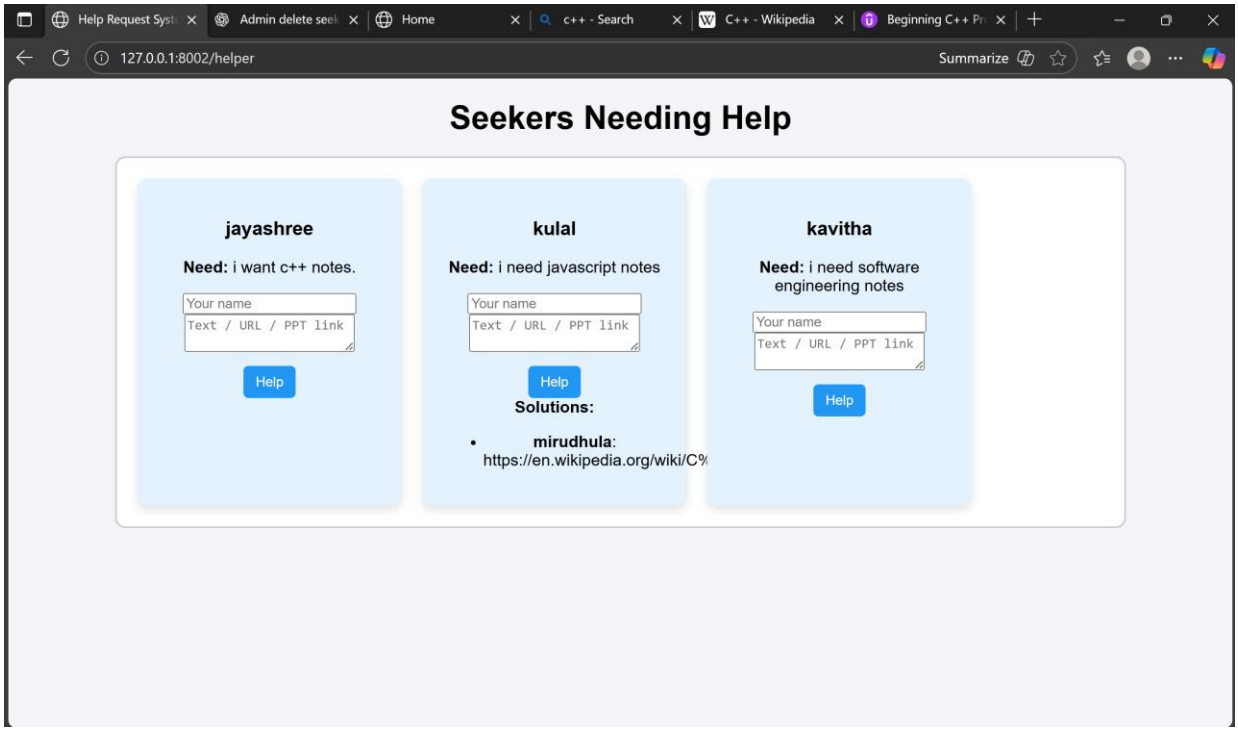




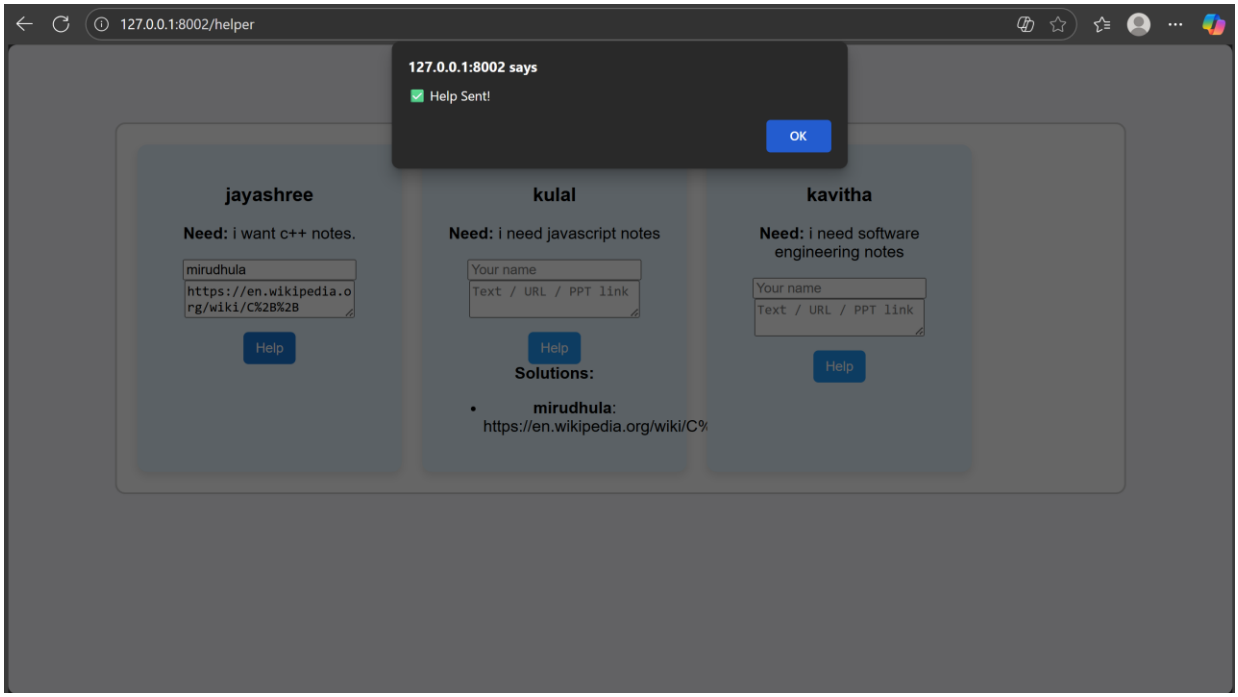
6.7.View FeedBacks



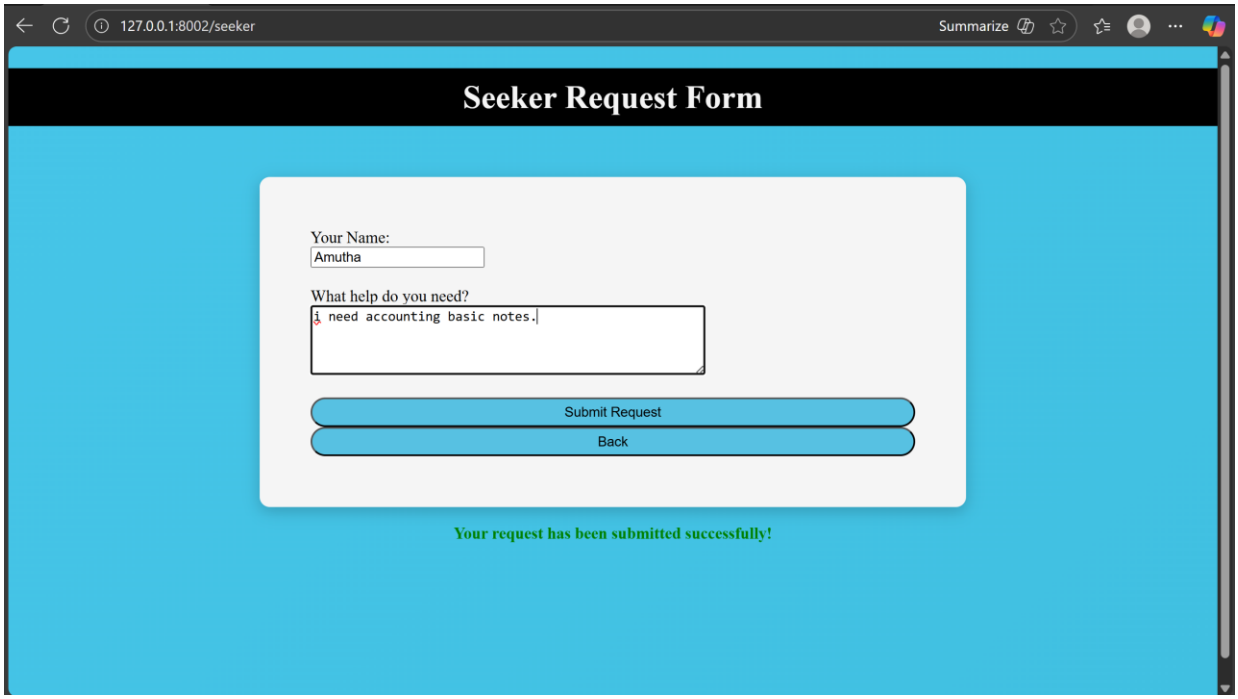
6.8.Helper



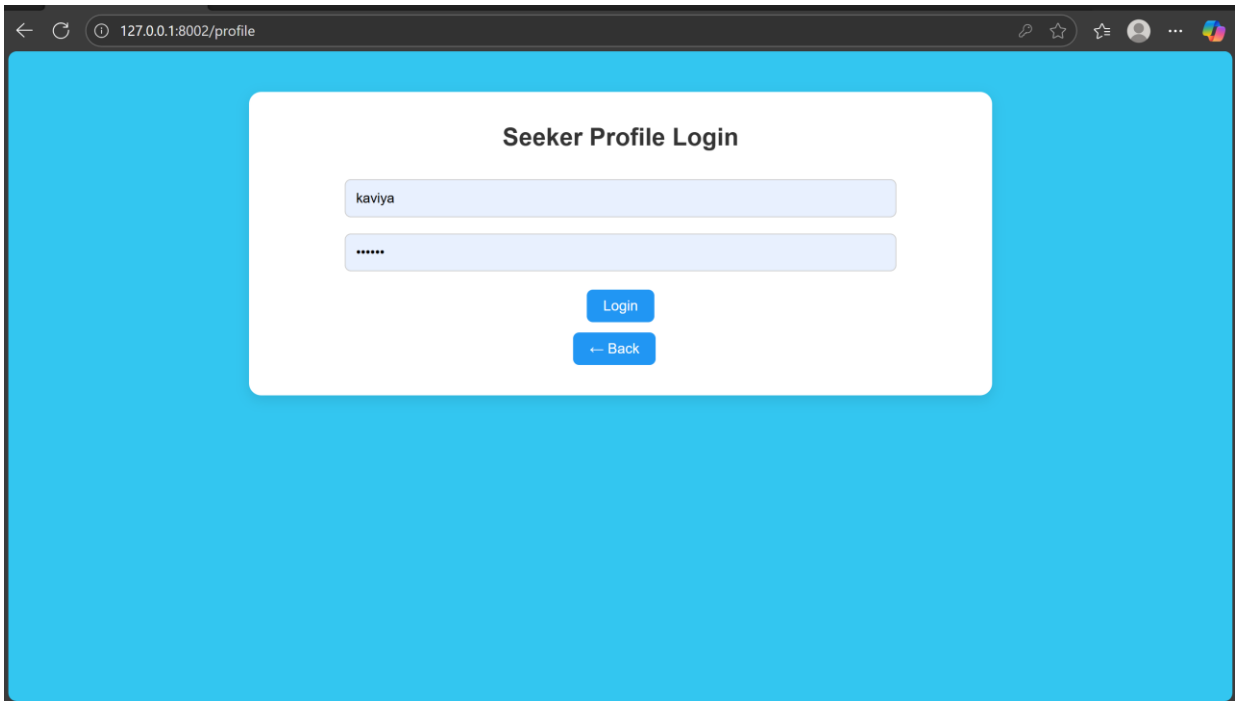
6.9.Helper sent help

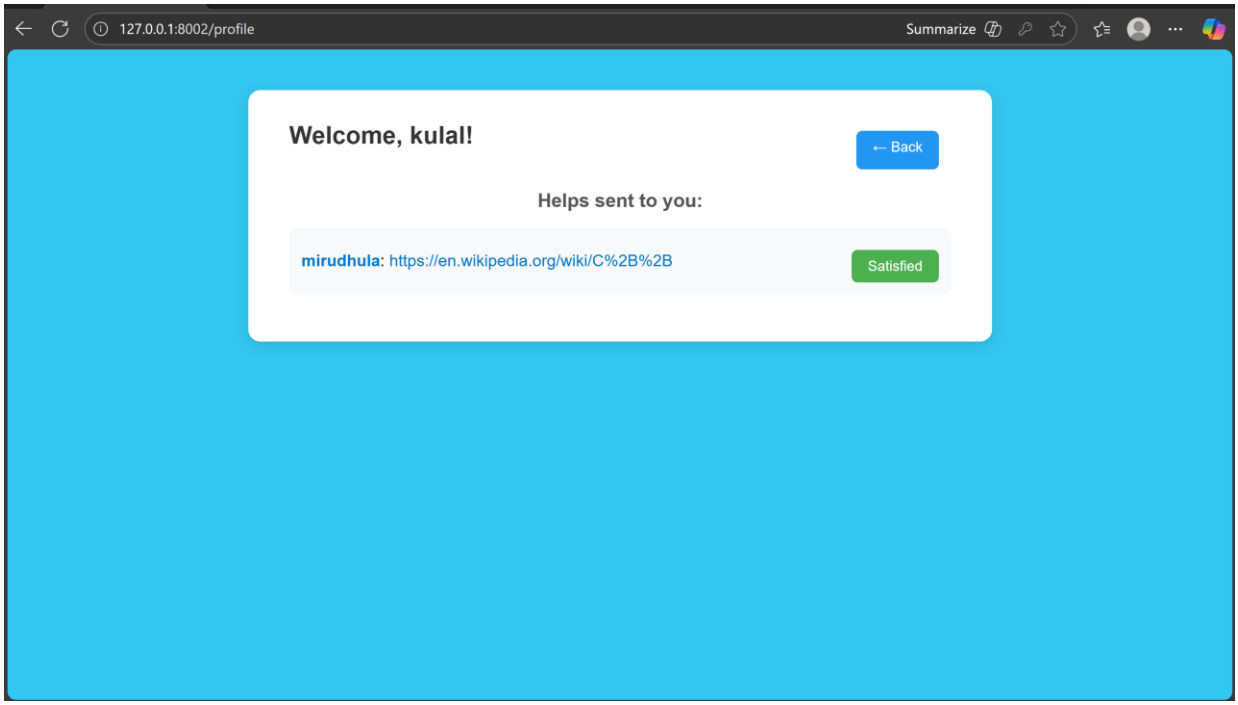


6.10.Seeker



6.11.Profile





6.12.Feedback