### **4. Interface Design (Higher Level)**

#### 4.1 System Architecture Overview

CULater follows a **three-tier client-server architecture**:

1. **Frontend (Client)**
   * Built with **React.js**.
   * Communicates with the backend via **RESTful APIs**.
   * Provides a **Graphical User Interface (GUI)** for user interactions.
2. **Middleware (Backend API)**
   * Developed using **Flask (Python)**.
   * Implements core business logic.
   * Manages **user authentication, task handling, group management**.
   * Interacts with the database via **SQLAlchemy ORM**.
3. **Database (PostgreSQL)**
   * Stores **users, tasks, groups, notifications**.
   * Ensures data **integrity, indexing, and security**.

#### 4.2 Communication Between Components

| **Component** | **Communication Partner** | **Protocol** | **Expected Input** | **Expected Output** | **Error Handling** |
| --- | --- | --- | --- | --- | --- |
| **Frontend (React.js)** | Middleware (Flask) | REST API (HTTPS, JSON) | API requests (Login, Task Creation, Group Join) | JSON responses (Success/Error messages, Data) | Displays error messages, redirects user |
| **Middleware (Flask)** | Database (PostgreSQL) | SQL Queries via SQLAlchemy | Queries for user auth, task CRUD, group management | Query results (User data, Tasks, Groups) | Rollback transactions, return HTTP 400/500 |
| **Middleware (Flask)** | External Authentication (OAuth, CUHK Login API) | OAuth2.0 / JWT | User credentials | JWT token | Invalid tokens return HTTP 401 |
| **Middleware (Flask)** | CI/CD Pipeline (GitHub Actions) | Webhooks | Code push events | Deployment status | Logs errors in CI/CD dashboard |

#### 

#### 4.3 API Endpoints

| **Endpoint** | **Method** | **Description** | **Request Body** | **Response** |
| --- | --- | --- | --- | --- |
| **User Management** | | | | |
| /api/auth/signup | POST | Register new user | { email, password } | 201 Created / 400 Bad Request |
| /api/auth/login | POST | Authenticate user | { email, password } | { token } / 401 Unauthorized |
| **Task Management** | | | | |
| /api/tasks | GET | Fetch all tasks | Authorization: Bearer <token> | [ { task\_id, title, description } ] |
| /api/tasks | POST | Create new task | { title, description, date, location, group\_id } | 201 Created / 400 Bad Request |
| **Group Management** | | | | |
| /api/groups | GET | Fetch user groups | Authorization: Bearer <token> | [ { group\_id, name } ] |
| /api/groups/join | POST | Join a group | { group\_id } | 200 OK / 400 Bad Request |
| **Invitations & Memberships** | | | | |
| /api/invites/send | POST | Send an invite to a user | { member\_email, group\_id, role\_name } | 201 Created / 400 Bad Request |
| /api/invites | GET | Fetch pending invites | Authorization: Bearer <token> | [ { member\_email, group\_id, inviter\_email, invited\_at } ] |
| /api/invites/accept | POST | Accept an invite | { group\_id } | 200 OK / 400 Bad Request |
| /api/invites/decline | POST | Decline an invite | { group\_id } | 200 OK / 400 Bad Request |
| /api/memberships | GET | Fetch group memberships | Authorization: Bearer <token> | [ { group\_id, role\_name, joined\_at } ] |

#### 4.4 Expected Exceptions & Handling

| **Scenario** | **Exception** | **Handling Strategy** |
| --- | --- | --- |
| User enters wrong credentials | 401 Unauthorized | Frontend displays error message |
| Task creation with missing title | 400 Bad Request | Backend returns validation error |
| Unauthorized task deletion | 403 Forbidden | Backend denies request |
| Database connection failure | 500 Internal Error | Retry mechanism, logs error |
| User tries to accept an expired invite | 400 Bad Request | Backend returns error message |

#### 4.5 Security Measures

1. **Authentication & Authorization**
   * **JWT Tokens** for session management.
   * **Role-based access control (RBAC)** (Admin, Contributor, Reader).
2. **Data Security**
   * Password hashing via **bcrypt**.
   * **HTTPS encryption** for all API requests.
3. **Input Validation & Protection**
   * Prevent **SQL Injection** (ORM with parameterized queries).
   * Validate all user inputs.

### **5. Component Design (Low Level)**

#### 5.1 Frontend Components (React)

| **Component** | **Responsibilities** | **Input** | **Output** |
| --- | --- | --- | --- |
| LoginPage.js | Handles user login | { email, password } | JWT Token / Error |
| TaskList.js | Displays tasks | User ID | [{ task\_id, title, date }] |
| TaskForm.js | Create/Edit tasks | { title, date, group\_id } | 201 Created / Error |
| GroupList.js | View user groups | User ID | [{ group\_id, name }] |

#### 5.2 Backend Components (Flask)

| **Module** | **Responsibilities** | **Input** | **Output** |
| --- | --- | --- | --- |
| auth.py | Manages user login/signup | { email, password } | { token } |
| tasks.py | Handles task CRUD operations | { title, date, group\_id } | { task\_id } |
| groups.py | Manages group creation/joining | { name } | { group\_id } |
| invites.py | Handles group invitations | { member\_email, group\_id, role\_name } | 201 Created / Error |
| memberships.py | Manages memberships | { group\_id } | { membership\_id } |

##### 

##### **Example: User Authentication (Flask)**

| from flask import Flask, request, jsonify from flask\_jwt\_extended import create\_access\_token from werkzeug.security import check\_password\_hash from models import User  app = Flask(\_\_name\_\_)  @app.route('/api/auth/login', methods=['POST']) def login():  data = request.json  user = User.query.filter\_by(email=data['email']).first()  if user and check\_password\_hash(user.password, data['password']):  token = create\_access\_token(identity=user.id)  return jsonify({"token": token}), 200  return jsonify({"error": "Invalid credentials"}), 401 |
| --- |

#### 

#### 

#### **5.3 Database Schema (PostgreSQL)**

| **CREATE TABLE users (  email VARCHAR(255) PRIMARY KEY,  password\_hash TEXT NOT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP );  CREATE TABLE tasks (  id SERIAL PRIMARY KEY,  title VARCHAR(255) NOT NULL,  description TEXT,  due\_date DATE,  location VARCHAR(255),  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  group\_id INTEGER REFERENCES groups(id) );  CREATE TABLE groups (  id SERIAL PRIMARY KEY,  name VARCHAR(255) NOT NULL,  description TEXT,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP );  CREATE TABLE roles (  name VARCHAR(50) PRIMARY KEY,  description TEXT );  CREATE TABLE memberships (  member\_email VARCHAR(255) REFERENCES users(email),  group\_id INTEGER REFERENCES groups(id),  accepted BOOLEAN DEFAULT FALSE,  role\_name VARCHAR(50) REFERENCES roles(name),  inviter\_email VARCHAR(255) REFERENCES users(email),  invited\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  joined\_at TIMESTAMP,  PRIMARY KEY (member\_email, group\_id) );  CREATE TABLE software\_licenses (  key VARCHAR(255) PRIMARY KEY,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  used\_status BOOLEAN DEFAULT FALSE );** |
| --- |

**5.4 Algorithms**

**Task Filtering by Group**

| def get\_tasks\_by\_group(user\_id, group\_id):  tasks = Task.query.filter\_by(user\_id=user\_id, group\_id=group\_id).all()  return tasks |
| --- |

**User Authentication**

| def authenticate(email, password):  user = User.query.filter\_by(email=email).first()  if user and check\_password\_hash(user.password, password):  return create\_access\_token(identity=user.id)  return None |
| --- |

**Handling Invitation**

| def send\_invite(member\_email, group\_id, role\_name, inviter\_email):  new\_invite = Membership(  member\_email=member\_email,  group\_id=group\_id,  role\_name=role\_name,  inviter\_email=inviter\_email,  accepted=False  )  db.session.add(new\_invite)  db.session.commit()  return new\_invite |
| --- |

**Accepting Invitation**

| **def accept\_invite(member\_email, group\_id):  invite = Membership.query.filter\_by(member\_email=member\_email, group\_id=group\_id, accepted=False).first()  if invite:  invite.accepted = True  invite.joined\_at = datetime.utcnow()  db.session.commit()  return invite  return None** |
| --- |