### **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** |
| --- | --- | --- | --- | --- |
| /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 |
| /api/tasks | GET | Fetch all tasks | Authorization: Bearer <token> | [{ task\_id, title, description }] |
| /api/tasks | POST | Create new task | { title, description, date } | 201 Created / 400 Bad Request |
| /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 |

#### **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 |

#### **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 } |

##### 

##### **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 (  id SERIAL PRIMARY KEY,  email VARCHAR(255) UNIQUE NOT NULL,  password\_hash TEXT NOT NULL );  CREATE TABLE tasks (  id SERIAL PRIMARY KEY,  title VARCHAR(255) NOT NULL,  description TEXT,  due\_date DATE,  user\_id INTEGER REFERENCES users(id) );  CREATE TABLE groups (  id SERIAL PRIMARY KEY,  name VARCHAR(255) NOT NULL,  admin\_id INTEGER REFERENCES users(id) );** |
| --- |

**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 |
| --- |

**Reasoning:**

The choices I made are based on:

1. Our project specification
2. Biking2 example
   1. 5.1. Whitebox biking APIs
   2. 5.2. Building blocks - Level 2
   3. 9. Design decisions

**Summary:**

**High-Level Interface Design**

* **Three-tier architecture** (Frontend, Middleware, Database).
* **RESTful API communication** with HTTP methods.
* **Security** via JWT, HTTPS, and RBAC.

**Component Design**

* **Frontend (React)**: Task list, group management.
* **Backend (Flask)**: Auth, task handling, group management.
* **Database (PostgreSQL)**: Stores users, tasks, groups.