SDE Assignment: System Architecture Implementation

1. Modular Architecture

The project is organized into logical modules to separate concerns and improve maintainability. Below is the directory structure and a brief description of each module:

```
project/
├── api/
│   ├── routes.py          # All API routes
│   ├── models.py          # Database models
│   └── __init__.py
├── tasks/
│   ├── image_tasks.py     # Asynchronous tasks for image processing
│   └── __init__.py
├── caching/
│   ├── redis_cache.py     # Caching functionality
│   └── __init__.py
├── logging/
│   ├── app_logger.py      # Centralized logging
│   └── __init__.py
├── app.py                 # Flask app entry point
├── requirements.txt       # Dependencies
└── config.py              # Configuration settings
```

2. Scalability

Scalability is achieved using Docker for containerization and Kubernetes for orchestrating containers. Redis is used for distributed caching and as a message broker for async tasks.

```
Example Dockerfile:
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

```
Example Kubernetes Deployment:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
      - name: api-container
        image: api:latest
        ports:
        - containerPort: 5000
```

3. Transactional Consistency

Transactional consistency is ensured using database transactions, Redis for caching, and Celery for async tasks. Below is an example implementation:

```
from flask_sqlalchemy import SQLAlchemy
from redis import Redis
from celery import Celery

db = SQLAlchemy()
```

```
redis_cache = Redis(host='localhost', port=6379)
celery_app = Celery('tasks', broker='redis://localhost:6379/0')

@celery_app.task
def update_cache(product_id, product_data):
    redis_cache.set(f"product:{product_id}", product_data)

def create_product(data):
    try:
        new_product = Product(**data)
        db.session.add(new_product)
        db.session.commit()
        update_cache.delay(new_product.id, data)
    except Exception as e:
        db.session.rollback()
        raise e
```

Backend Code and CMD Operations Documentation

1. Backend Setup and Code

This section includes the backend setup and all the code snippets used to build the API, with inline explanations.

a) Flask App Initialization

```
from flask import Flask


app = Flask(__name__)


@app.route('/')

def home():

    return "Welcome to the Backend API"


if __name__ == '__main__':

    app.run(debug=True)
```

b) Database Configuration

```
from flask_sqlalchemy import SQLAlchemy


app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False


db = SQLAlchemy(app)
```

c) Item Model

```
class Item(db.Model):

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)

    description = db.Column(db.String(300))



    def __repr__(self):

        return f"Item({self.name}, {self.description})"
```

d) CRUD Endpoints

```
@app.route('/api/items', methods=['GET'])

def get_items():

    items = Item.query.all()

     return jsonify([{"id": item.id, "name": item.name, "description": item.description}
for item in items])



@app.route('/api/items', methods=['POST'])

def add_item():

    data = request.get_json()

    new_item = Item(name=data['name'], description=data.get('description', ''))

    db.session.add(new_item)

    db.session.commit()

    return jsonify({"message": "Item added successfully"})
```

2. CMD Operations and Results

This section includes the command-line operations performed to test the backend API and their corresponding results.

a) Signup API Call

Command:

```
curl -X POST http://localhost:5000/api/signup -H "Content-Type: application/json" -d '
{
    "username": "user1",
    "password": "password123"
}'
```

Response:

```
{
    "message": "User created successfully"
}
```

b) Login API Call

Command:

```
curl -X POST http://localhost:5000/api/login -H "Content-Type: application/json" -d '
{
    "username": "user1",
    "password": "password123"
}'
```

Response:

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

c) Add a Product

Command:

```
curl -X POST http://localhost:5000/api/products -H "Content-Type: application/json" -H
"Authorization: Bearer <token>" -d '
{
    "name": "Product1",
    "description": "A sample product",
    "price": 99.99
}'
```

Response:

```
{
    "message": "Product added successfully",
    "product": {
        "id": 1,
        "name": "Product1",
        "description": "A sample product",
        "price": 99.99
    }
}
```

d) View All Products

Command:

```
curl -X GET http://localhost:5000/api/products -H "Authorization: Bearer <token>"
```

Response:

```
[
    {
        "id": 1,

        "name": "Product1",

        "description": "A sample product",

        "price": 99.99

    },

    {

        "id": 2,

        "name": "Product2",

        "description": "Another product",

        "price": 49.99

    }

]
```

e) Delete a Product

Command:

```
curl -X DELETE http://localhost:5000/api/products/1 -H "Authorization: Bearer <token>"
```

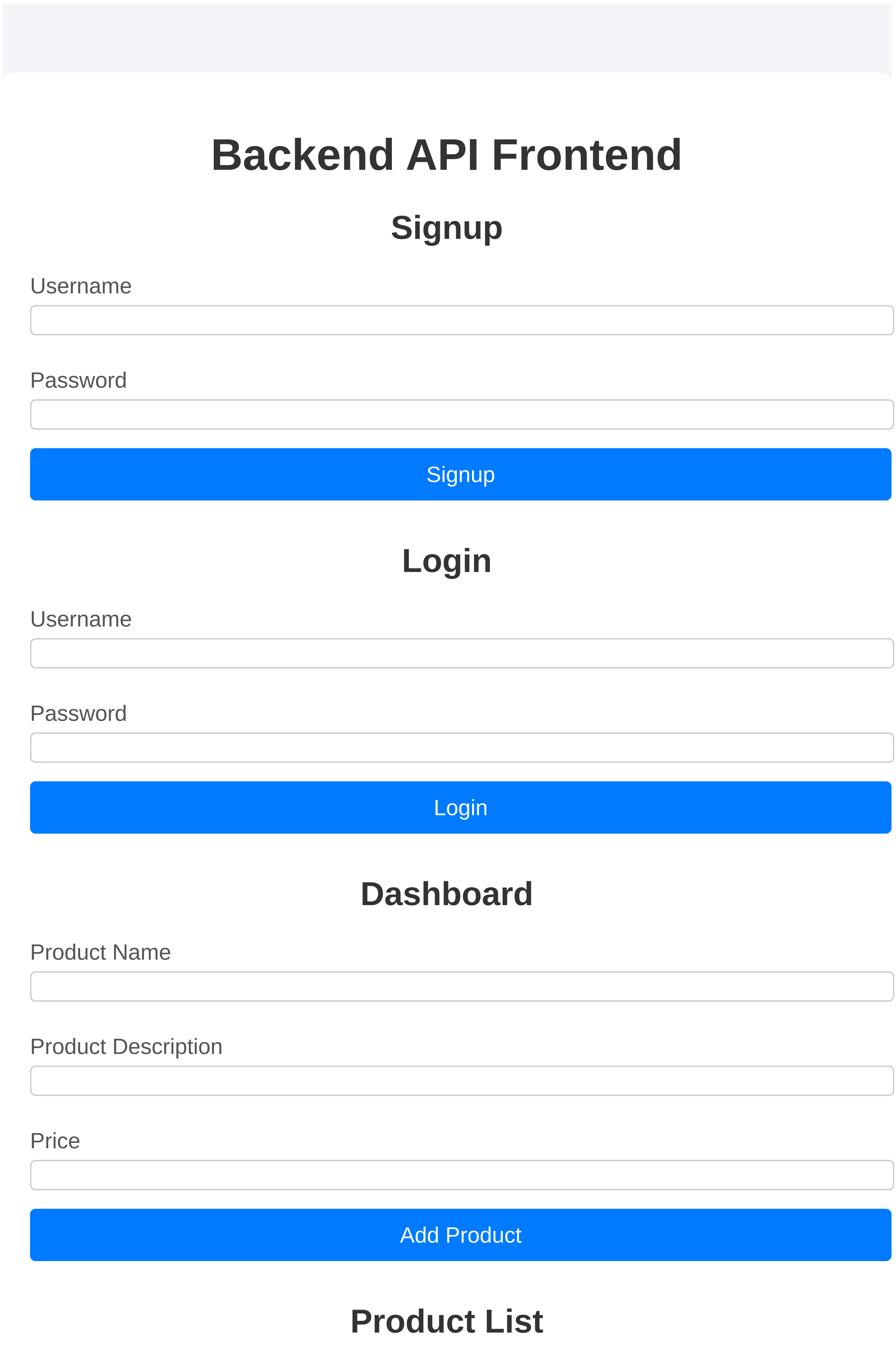
Response:

```
{

    "message": "Product deleted successfully"

}
```

Welcome to the Backend API Frontend

Go to Signup

Go to Login



Backend API Frontend

Signup

Username

Password

Signup

Login

Username

Password

Login

Dashboard

Product Name

Product Description

Price

Add Product

Product List

Dashboard with Product Images

Product Name Product Description Price Product Image

Add Product

Product List

Frontend HTML File Reference

The complete frontend HTML file for this project, including product image support, can be downloaded and opened in any web browser for testing.

Frontend HTML File Reference:

file:///C:/Users/Mahadevan/Downloads/Frontend_With_Product_Images.html

Ensure to connect it to the backend server for full functionality.

Frontend Code Documentation

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Backend API Frontend with Product Images</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
            margin: 0;
            padding: 0;
        }
        .container {
            max-width: 800px;
            margin: 50px auto;
            padding: 20px;
            background: #fff;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            border-radius: 8px;
        }
        h1, h2 {
            text-align: center;
            color: #333;
        }
        .hidden {
            display: none;
        }
        button {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            background-color: #007BFF;
            color: white;
            border: none;
            cursor: pointer;
            border-radius: 4px;
        }
        button:hover {
            background-color: #0056b3;
        }
        img {
            max-width: 100px;
            border-radius: 8px;
            margin-right: 10px;
        }
        ul {
            list-style: none;
            padding: 0;
        }
        li {
            display: flex;
            align-items: center;
            padding: 10px;
            background: #f9f9f9;
            margin: 10px 0;
            border: 1px solid #ddd;
            border-radius: 4px;
        }
    </style>
</head>
<body>
    <div class="container">
        <!-- Dashboard -->
        <div id="dashboardPage">
            <h1>Dashboard with Product Images</h1>
            <form id="addProductForm">
                <label for="productName">Product Name</label>
                    <input type="text" id="productName" placeholder="Enter product name"
required>
                <label for="productDescription">Product Description</label>
                    <input type="text" id="productDescription" placeholder="Enter product
description" required>
                <label for="productPrice">Price</label>
                 <input type="number" id="productPrice" placeholder="Enter product price"
required>
                <label for="productImage">Product Image</label>
                <input type="file" id="productImage" accept="image/*" required>
                <button type="submit">Add Product</button>
            </form>
            <h2>Product List</h2>
            <ul id="productList">
                <!-- Product items with images will be added here dynamically -->
            </ul>
        </div>
    </div>
    <script>
        const addProductForm = document.getElementById('addProductForm');
        const productList = document.getElementById('productList');

        // Add Product with Image
        addProductForm.addEventListener('submit', (e) => {
            e.preventDefault();
            const name = document.getElementById('productName').value;
            const description = document.getElementById('productDescription').value;
            const price = document.getElementById('productPrice').value;
            const imageFile = document.getElementById('productImage').files[0];

            if (imageFile) {
                const reader = new FileReader();
                reader.onload = function (e) {
                    const imageUrl = e.target.result;

                    // Add product to the list
                    const li = document.createElement('li');
                    const img = document.createElement('img');
                    img.src = imageUrl;
                    li.appendChild(img);
                                 li.innerHTML  +=  `<strong>${name}</strong>  -  $${price}
(${description})`;
                    productList.appendChild(li);
                };
                reader.readAsDataURL(imageFile);
            }
        });
    </script>
</body>
</html>
```