**CHAPTER 1**

**INTRODUCTION**

## Introduction & Motivation

## The ****Inventory Management Web App**** is a user-friendly tool designed to streamline expense tracking. It enables users to effortlessly record, monitor, and analyse their daily expenditures, offering features such as easy expense entry, convenient viewing options by date or category, and precise computation of total expenditure. Additionally, users can set budgets for various expense categories, receive alerts for budget limits, and visualize spending patterns through intuitive charts and graphs. With robust security measures ensuring data privacy, this app provides a comprehensive solution for effective financial Management.

The motivation behind the project was to make a convenient system for small to medium scale businesses to properly manage item/stock in their inventory by giving them role-based access to many privileges such as managing stock, managing orders, and analysing the sales based on many factors. Using Inventory management web app, businesses can have a one-stop application to manage their inventory.

.

## 1.2 Existing system

Inventory management systems and apps are helpful tools for businesses to organize their stock effectively. These tools let you keep track of what products you have, sort them into categories, and set alerts when supplies are running low. They often connect with your suppliers to keep everything up to date without manual effort. Plus, many of these systems give easy-to-understand reports so you can see how your stock is doing and where you might need to make changes. Overall, using an inventory management system makes it easier for businesses to stay organized, save money, and make smart decisions about their stock. The drawback of many of such applications is that issues of privacy, data leaks and cost-of-entry. Most of these inventory management systems are online applications which operate in databases on server side which may cause issues with privacy and data leaks. Apart from this, software from many famous companies have very high cost-of-entry meaning the cost of purchasing the software is very high and user specific features have high cost for implementation as custom features. Along with this, many companies have implemented subscription-based model which has its own issues.

## 1.3 Problem statement

In today's bustling business environment, effectively managing inventory and overseeing stock movements is essential. Manual inventory tracking methods are not only laborious but also prone to errors, leading to significant operational hurdles. Introducing the "Inventory Management Web App" project, a user-friendly digital solution aimed at empowering businesses to efficiently manage their inventory, minimize errors, and streamline operations.

## 1.4 Objectives

1. **Efficient Inventory Tracking:** Develop a system that enables businesses to track their inventory levels accurately in real-time, reducing the risk of stockouts and overstocking.

2**. Streamlined Order Management:** Create a platform that simplifies the process of creating, managing, and fulfilling orders, improving order accuracy and fulfilment speed.

3. **Comprehensive Reporting and Analytics:** Provide users with robust reporting and analytics capabilities, allowing them to generate actionable insights into inventory performance, sales trends, and stock movement.

4**. User-Friendly Interface:** Design an intuitive and user-friendly interface that enables users of all levels of technical expertise to navigate the web app easily and perform tasks efficiently.

5**. Integration with External Systems:** Enable seamless integration with accounting software, ecommerce platforms, and other external systems to synchronize data and streamline workflows across different business functions.

**6. Data Security and Compliance:** Implement robust security measures to protect sensitive inventory data from unauthorized access, ensuring compliance with data privacy regulations and industry standards.

**7. Scalability and Flexibility:** Build a scalable and flexible system that can accommodate the growing needs of businesses, whether they operate on a small scale or have complex inventory management requirements.

**8. Mobile Accessibility:** Develop a mobile-responsive design or companion mobile app that allows users to access the inventory management system from anywhere, anytime, using their smartphones or tablets.

**9. Cost Efficiency:**  Help businesses optimize inventory costs by providing tools and insights to minimize excess inventory, reduce carrying costs, and improve inventory turnover rates.

**10. Continuous Improvement:** Foster a culture of continuous improvement by gathering user feedback, monitoring system performance, and implementing regular updates and enhancements to meet evolving business needs and technology trends.

**1.5 Technology Implemented**

**1. Flask**: Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks. Flask offers suggestions but doesn’t enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy. We used Flask to handle the routing and rendering of your web application. For example, you might have come across routes like /inventory to show all items, /item/<item\_id> to show a specific item, and /additem to add a new item. Each of these routes would correspond to a function in your Flask app that would render an HTML template.

**2. Flask-SQL Alchemy:** Flask-SQL Alchemy is an extension for Flask that adds support for SQL Alchemy to your application. It simplifies using SQL Alchemy with Flask by setting up common objects and patterns for using those objects, such as a session tied to each web request, models, and engines. Flask-SQL Alchemy does not change how SQL Alchemy works or is used. This extension is used to interact with our database. We defined our inventory items as a class in Python, and Flask-SQL Alchemy would handle the details of storing and retrieving these items from the database.

**3. Flask-Login**: Flask-Login provides user session management for Flask. It handles the common tasks of logging in, logging out, and remembering your users' sessions over extended periods of time. Flask-Login is not bound to any particular database system or permissions model. The only requirement is that your user objects implement a few methods, and that you provide a callback to the extension capable of loading users from their ID. This extension is used to handle user sessions, so we can control who has access to our inventory management web Application. We use it to log users in and out, and to decorate routes that require a user to be logged in. Here, the @login\_required decorator prevents unauthenticated users from accessing the route.

## 1.6 Scope

1. **User Authentication:** The system provides user registration and login functionality. It uses Flask-Login for handling user sessions.

**2. Product Management:** The system allows users to add new products to the inventory. Each product has a name, part number, quantity, and category. Users can also update the details of an existing product.

**3. Category Management:** The system allows users to add new categories. Each category has a name and description. Users can also update the details of an existing category.

**4. Inventory Management:** The system allows users to stock up on an item or sell an item, updating the quantity of the item in the inventory accordingly.

**5. Request Management:** Users can request items from the inventory. These requests can be accepted or rejected.

**6. Cart Functionality:** Users can add items to a cart. They can update the quantity of an item in the cart or remove an item from the cart. There is also a checkout function, although the code for this function seems to be incomplete.

**7. Display Functionality:** The system can display all items, all requests, and all categories. It can also display the details of a specific category.

**1.7 Proposed System**

The proposed inventory management system aims to provide a robust and user-friendly platform for efficiently managing inventory operations within an organization. Leveraging the Flask framework along with Flask SQLAlchemy and Flask Login extensions, the system offers comprehensive functionalities tailored to meet the diverse needs of both employees and managers.

**System Architecture:**

* The system follows a client-server architecture, with the Flask application serving as the server-side component and providing RESTful APIs for communication with the client-side interface.
* Database connectivity is facilitated by Flask SQLAlchemy, which manages interactions with the underlying relational database management system (RDBMS).
* User Authentication and Authorization:
* User authentication and authorization mechanisms are implemented to ensure secure access to the system.
* The User model defines user attributes such as username, email, password, role, and active status. Role-based access control enables differentiation between employee and manager roles, allowing for granular control over system permissions.

**Inventory Operations:**

* The system enables employees to view product listings, process customer orders, request stock replenishment, and record sales transactions.
* Managers have access to advanced functionalities, including CRUD operations on inventory items, importing requested stock, and viewing analytics on sales performance.

**Database Model:**

* The database model includes entities such as User, Category, Item, ItemRequest, Cart, CartItem, and Order, defining the relationships and attributes necessary for managing inventory data effectively.
* Relationships between entities, such as item requests associated with users and items, carts containing cart items, and orders linked to carts, facilitate seamless data management and retrieval.

### CHAPTER 2

# REVIEW OF LITERATURE

**Title:** Game-Technical Management Of The Educational Environment  
**Authors:** Marina Aleksandrovna Romanova, Yulia A. Serebrennikova, Tatiana Fedorenko  
**Source:** SHS Web of Conferences  
**Year:** 2020

The research presented by Romanova et al. investigates the theoretical and practical implementation of game-technical management technologies in pedagogical universities to enhance the educational environment. Modern education faces significant challenges, driven by an evolving labour market that requires teachers to be adept at interactive technologies. The goal of the research is to develop a model for enriching education content using game-technical methods, emphasizing the teacher's role in mastering, implementing, and managing these technologies.

The study draws on international research and models, particularly those by Renzulli, Savenkov, and Heller, which focus on enriching education both horizontally and vertically. The horizontal enrichment deals with broadening a student’s knowledge, while vertical enrichment involves more in-depth engagement with specific content. The authors aim to adapt these approaches within the context of a contemporary pedagogical university, where teachers play a central role in managing and guiding students through interactive, game-based learning processes.

**Methodology**

The researchers designed a theoretical model of game-technical management to enhance the educational environment. The model is based on a systematic approach, integrating psychological and pedagogical principles. It consists of three interrelated components:

1. Target-oriented: focuses on mastering, implementing, and managing game technologies.

2. Content-oriented: involves identifying suitable interactive technologies and incorporating search and research games.

3. Estimation component: measures both cognitive and non-cognitive indicators to assess the model's effectiveness.

The model was tested on students from two universities-Moscow City Pedagogical University and Sakhalin State University-focusing on those in "Pedagogical Education" and "Psychological and Pedagogical Education" programs. The study engaged a mix of vertical (structuring teaching methods) and horizontal (enriching content) approaches.

**Key Findings**

The implementation of the game-technical management model led to several positive changes in both students and teachers, particularly in non-cognitive aspects like perseverance, creativity, and self-confidence. A significant shift was noted in the role of teachers, as students took a more active role in controlling the learning process, thereby fostering their development and creative problem-solving abilities.

The study's results were compared across experimental and control groups. The experimental groups, which applied the game-technical management model, showed better performance in solving pedagogical cases and more interaction-oriented behaviour. The model also led to an increased development of specific qualities such as resourcefulness, originality, and pedagogical thinking.

Role-playing games, pedagogical cases, and interactive simulations proved particularly effective in training future teachers. These tools provided practical experience and better prepared students for real-world teaching challenges. Additionally, the model helped shift the traditional focus from teacher-led instruction to more student-driven, game-based learning environments, which enhanced motivation and engagement.

**Statistical Results**

The study utilized Pearson’s Chi-square test to analyze data and evaluate the effectiveness of the game-technical model. For example, experimental group A performed significantly more correct operations than the control group B, indicating that the game-technical model promoted better problem-solving skills. Similarly, expert evaluations of students' pedagogical performance showed higher interaction-level indicators in the experimental group.

The model also had a long-term impact. A follow-up survey of young professionals (graduates who had been working for one year) indicated that those who had been trained with the game-technical management model held leadership positions and expressed higher job satisfaction. Notably, 224 out of 254 graduates surveyed were in leadership roles, and many demonstrated exceptional performance in their teaching careers.

**Discussion and Implications**

The study highlighted the significant qualitative improvements in student and teacher performance due to the game-technical management model. While the non-cognitive predictors such as motivation and engagement increased, cognitive predictors showed varying degrees of improvement, correlating with previous studies on the motivational aspect of intellectual activity. The research emphasizes that interactive technologies foster a dynamic educational environment where both students and teachers engage in creative, hands-on learning.

Game-technical management also shifted traditional teaching roles. By placing students at the center of the learning process, the model encouraged a more collaborative and flexible educational environment. The integration of role-playing, game design, and project-based learning helped future teachers develop skills in leadership, communication, and problem-solving.

The research demonstrated that implementing game-technical management technologies significantly enriched the educational environment in pedagogical universities. This model, when applied in the training of future teachers, led to the development of essential professional skills such as self-management, creativity, and interactive learning. The study underscores the importance of game-based, interactive methods in modern education, as they equip future educators with the tools necessary to adapt to an ever-changing educational landscape.

Further research is suggested to assess the long-term potential and risks of game-technical management in various educational contexts, with an emphasis on optimizing its integration into traditional teaching models. The authors advocate for continued exploration into the role of game-technical management in enhancing the quality of education and improving teacher training programs.

**Title:** Development of Game Management System Suitable for Mobile Games  
**Authors:** Dong-Seong Lee, Seuc-Ho Ryu, Byung-Pyo Kyung, Dong-Lyeor Lee  
**Source:** KoreaScience  
**Year:** 2018

This paper from the Journal of Digital Convergence discusses the development of a mobile game management system, particularly focusing on mobile games like Monster Master. The key point highlighted is the growing importance of a comprehensive game operation system, driven by the rapid expansion of the mobile game market. The authors explain that traditional mobile games are evolving into more service-oriented platforms requiring continuous updates and user engagement based on data-driven insights.

The paper proposes a mobile game management system that combines a user support system, content analysis system, and content operation system. This system aims to enhance game services, enabling developers to respond quickly to user queries, handle refunds, and improve game content based on user behaviour data. Additionally, the system allows for better planning of future updates, contributing to higher player satisfaction and retention.

Through the analysis of Monster Master, a turn-based RPG game, the authors illustrate how this system works in practice. They explain how the user support system facilitates real-time assistance, while the content analysis system helps optimize in-game content and balance. The content management system allows for flexible updates without the need for client modifications.

This paper highlights the need for operational tools that can analyze user data and enable developers to implement dynamic in-game changes quickly, which is critical for maintaining long-term user engagement in an increasingly competitive mobile game market.

**Key components of the system include:**

1. User Support System: This system is used to manage user inquiries, refunds, and in-game item recovery efficiently without requiring extensive operator intervention.

2. Content Analysis System: The system provides real-time user data, such as payment history and in-game behaviour, to help adjust game mechanics and plan future updates based on user trends and behaviour.

3. Content Management System: This allows for live updates to in-game content, the running of events, and real-time modifications without requiring game downtime or client-side updates.

The study uses the game "Monster Master" as a case study to illustrate the effectiveness of this management system. It demonstrates that through better user support and data-driven content management, companies can improve user retention and engagement, streamline operational processes, and plan content updates more effectively.

The paper concludes that such management systems are crucial for mobile games, particularly as the market expands and becomes more competitive. It highlights the importance of efficient game operation systems in enhancing the player experience and ensuring sustainable game development.

**Title:** Research Trends of Game-Based Language Learning in K-12 Education  
**Authors:** Lu Yang, Rui Li, Yu Zhou  
**Source:** Journal of Computer Assisted Learning (JCAL)  
**Year:** 2022

The paper titled "Research Trends of Game-Based Language Learning in K-12 Education: A Systematic Review of SSCI Articles during 2009–2022" by Lu Yang, Rui Li, and Yu Zhou, offers a comprehensive analysis of the research conducted in game-based language learning (GBLL) for K-12 students. It focuses on theoretical frameworks, instructional practices, and outcomes based on 83 Social Science Citation Index (SSCI) articles published between 2009 and 2022.

**Key Findings:**

1. **Theoretical Frameworks:**

Most studies utilized cognitive and social learning theories such as flow theory, situated learning, and socio-cultural theory. These frameworks emphasize how immersive and context-based learning can enhance language acquisition by engaging students in dynamic, interactive environments.

2. **Participants and Languages:**

The majority of the studies focused on primary school students, particularly those learning English as a foreign language (EFL). This indicates a strong interest in improving early language learning, especially in non-native English-speaking regions.

3. **Research Methods:**

Quantitative research methods dominated, with many studies employing experimental designs, questionnaires, and tests. This aligns with the goal of assessing measurable outcomes such as language proficiency improvements, student engagement, and cognitive load during gameplay.

4. **Game Genres:**

Tutorial-based games were the most frequently used genre. These games typically provide structured learning experiences where players follow guided instructions and engage in activities designed to improve specific language skills.

5. **Instructional Activities:**

Pre-game activities, like gameplay training and lectures, were commonly implemented to help students understand how to use the games effectively. This preparation was critical for ensuring that students could navigate the games and maximize their learning experience.

6. **Outcomes:**

Positive outcomes included enhanced communicative competence, improved attitudes toward language learning, and progress in the four language skills (listening, speaking, reading, and writing). However, technical issues, uncooperative peers, and complex tasks sometimes led to increased cognitive load and frustration among learners.

**Implications:**

The findings of this review highlight the need for:

* Pedagogical Improvements: Teachers and designers should continue developing GBLL experiences that balance engagement with cognitive load, ensuring games are neither too easy nor overly challenging.
* Technology Enhancements: Addressing technical issues is essential to providing a smooth gaming experience for learners.
* Customizing Games for Younger Learners: Since primary school students were the main participants, future research should explore the potential of GBLL in secondary education, as well as in different language learning contexts.

This systematic review offers valuable insights for educators, game designers, and researchers seeking to understand the evolving role of games in language education for young learners. It advocates for continued exploration of how games can effectively support language acquisition, while also identifying areas for improvement in game design and instructional integration.

### CHAPTER 3

### REQUIREMENT ANALYSIS

## 3.1 Software Requirements:

**1. Backend Framework:**

* Python: Use Python for coding the logic behind the web app.
* Flask or Django: Choose Flask or Django to handle web requests and interact with the database.
* SQLAlchemy or Django ORM: Employ SQLAlchemy or Django's built-in ORM to communicate with the database easily.

**2. Database Management System (DBMS):**

* + SQLite, PostgreSQL, or MySQL: Pick one of these databases to store and manage inventory data.

**3. Frontend Technologies:**

* + HTML5, CSS3, JavaScript: Use these to create the look, feel, and interactivity of the web app.
  + Bootstrap or Bulma: These frameworks make it easier to design responsive web pages.
  + JavaScript libraries (e.g., jQuery): Optional tools to simplify coding tasks.

**4. Additional Libraries and Tools:**

* + Flask-Login or Django's authentication: Manage user login and sessions.
  + Flask-WTF or Django forms: Build and validate web forms.
  + Barcode scanner library (if needed): Integrate barcode scanning functionality.
  + Reporting and charting libraries: Generate reports and visualizations.

## 3.2 Hardware Requirements:

**Server Hardware Requirements:**

1. **Processor**: A modern multi-core processor (e.g., Intel Core i5 or AMD Ryzen 5).
2. **Memory (RAM)**: At least 4GB, though more may be needed for larger datasets or heavy concurrent usage.
3. **Storage**: Solid-state drive (SSD) preferred for faster read/write speeds, but a traditional hard disk drive (HDD) can suffice.
4. **Network Interface**: Stable internet connection with sufficient bandwidth to handle incoming requests from clients.

**Client Hardware Requirements:**

1. **Processor**: Any modern processor capable of running web browsers efficiently (e.g., Intel Core i3 or equivalent).
2. **Memory (RAM)**: Minimum 4GB, though more may be beneficial for multitasking.
3. **Storage**: Adequate storage space for browser cache and local files (SSD or HDD).
4. **Network Interface**: Internet connection with sufficient bandwidth for accessing the web application.

## 3.3 System Requirements:

**3.3.1 Functional Requirements:**

**1. Two types of users:** a) Employee b) Manager. Employee will serve as user that views and sells items to customer and manager will be administrator for inventory.

**2. Ability to classify items according to category:** Every item should belong to some category which can be edited and accessed to view all products in that category.

**3. Ability to view all items:** The system should be able to show all the items in inventory along with relative information such as: Part no., price, quantity, category, a picture which can be edited by manager-admin.

**4. Managing items and categories:** Manager-admin should be able to manage- create, view, update, delete (CRUD operations) on items and categories.

**5. Sell items to user:** Manager and Employee should be able to sell single or multiple items to user and be able to see the order details. Manager should be able to see all the orders made by all employees.

**6. Request for restock:** Employee should be able to request for restock of item. Manager should be able to grant the request for restock.

**7. Analysis of sales:** Employee and Manager should be able to see the statistics of sales according to item category and sales made by each user.

**8. API integration:** Item information can be accessed via command prompt i.e. command line interface. CRUD operations must be able to be performed in item.

**9. User interface:** The website must have clean and easy to use interface.

**3.3.2 Non-Functional Requirements:**

**1. Login framework:** Flask-Login must be used for login according to the role-based functionality of the application.

**2. Database:** SQLite database can be used for storing the data as application can be kept lightweight and easy to use. To implement this, Flask-SQLAlchemy module is to be used.

**3. API framework:** Flask-Restful can be used to implement API in the application to perform CRUD operations in item.

**4. User Interface:** HTML pages styled using CSS and Bootstrap are to be used for making the interface neat and clean.

**5. Scalability:** Web framework – Flask can be scaled using modules such as Flask-SQLAlchemy for database in SQLite, Flask-Restful for API integration for scaling the frontend using frameworks like Vue.js, Angular, React, etc.

**6. Statistics using Matplotlib:** Matplotlib library can be used for generating graphs and charts dynamically when user requests for the analysis.

**7. Security:** Manager features must not be accessed to regular employees.

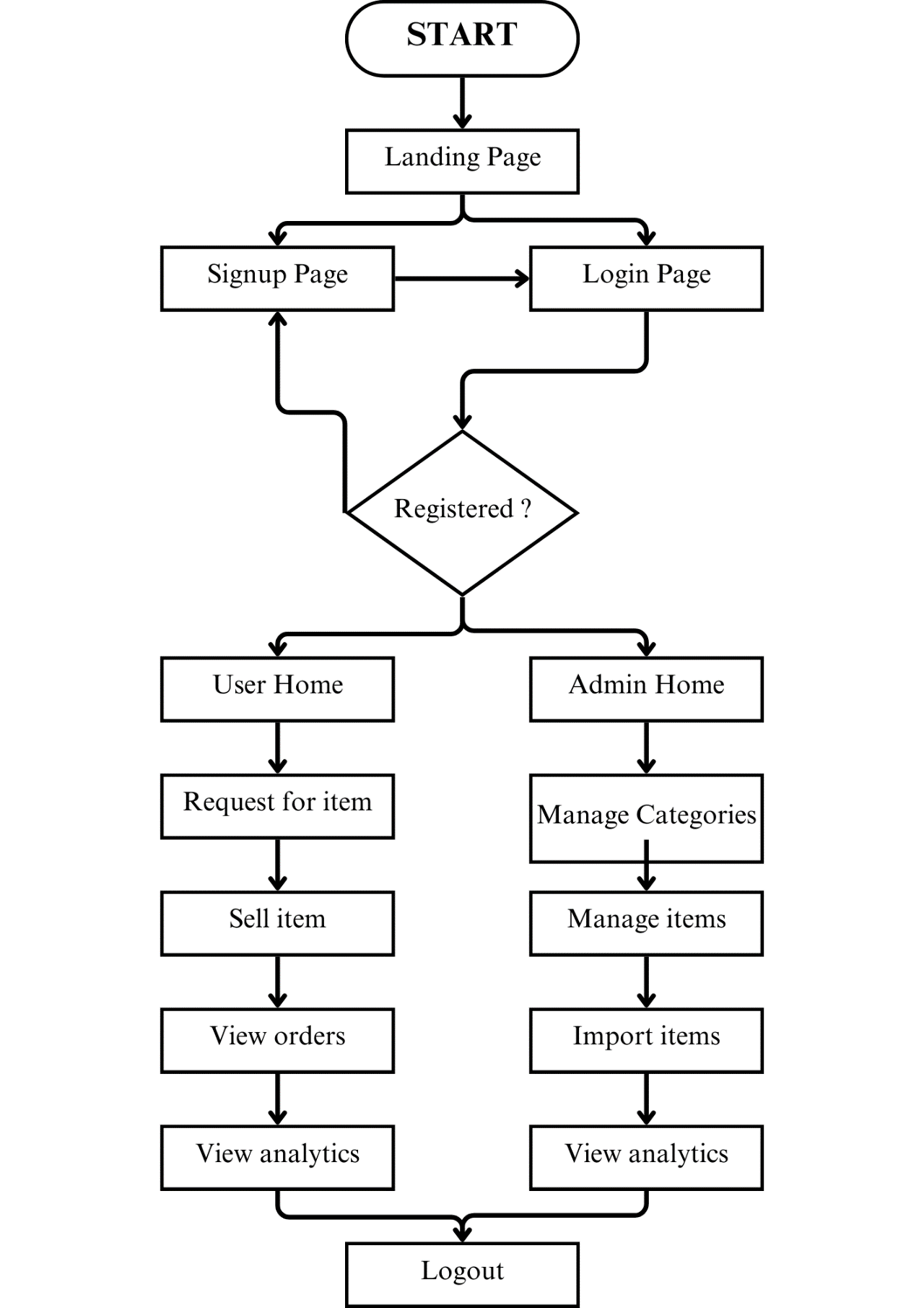
### CHAPTER 4

### DESIGN AND PLANNING

In the design and planning section of the project report for the inventory management system utilizing Flask web app, the focus is on outlining the comprehensive architecture and integration of various technologies. The project leverages Flask, a micro web framework in Python, along with Flask-SQLAlchemy for seamless database connectivity, ensuring efficient handling of inventory data. Flask-RESTful is employed to facilitate API implementation, enabling smooth communication between the frontend and backend components. The utilization of Flask-Login for user authentication ensures secure access control, distinguishing between employees and managers with distinct privileges.

Furthermore, the project incorporates data visualization techniques using Matplotlib and NumPy within the Flask framework. Matplotlib is utilized to generate dynamic pie charts and bar graphs, providing insightful visual representations of inventory statistics. HTML and CSS, supplemented by Bootstrap for styling, constitute the frontend, offering an intuitive user interface for seamless navigation and interaction. Additionally, comprehensive documentation, including workflow diagrams, use case diagrams, database schema, and data flow diagrams, alongside YAML documentation for Flask-RESTful API interactions, ensures clarity in system functionalities and interactions. This meticulous design and planning approach lay the foundation for the development of a robust inventory management system with a user-friendly interface and efficient backend processes.

**4.1 Flowchart (Workflow Diagram)**

****

(Fig. 4.1)

**4.2 Use Case Diagram**

**A diagram of a web application

Description automatically generated**

(Fig. 4.2)

**4.3 Data Flow Diagrams**

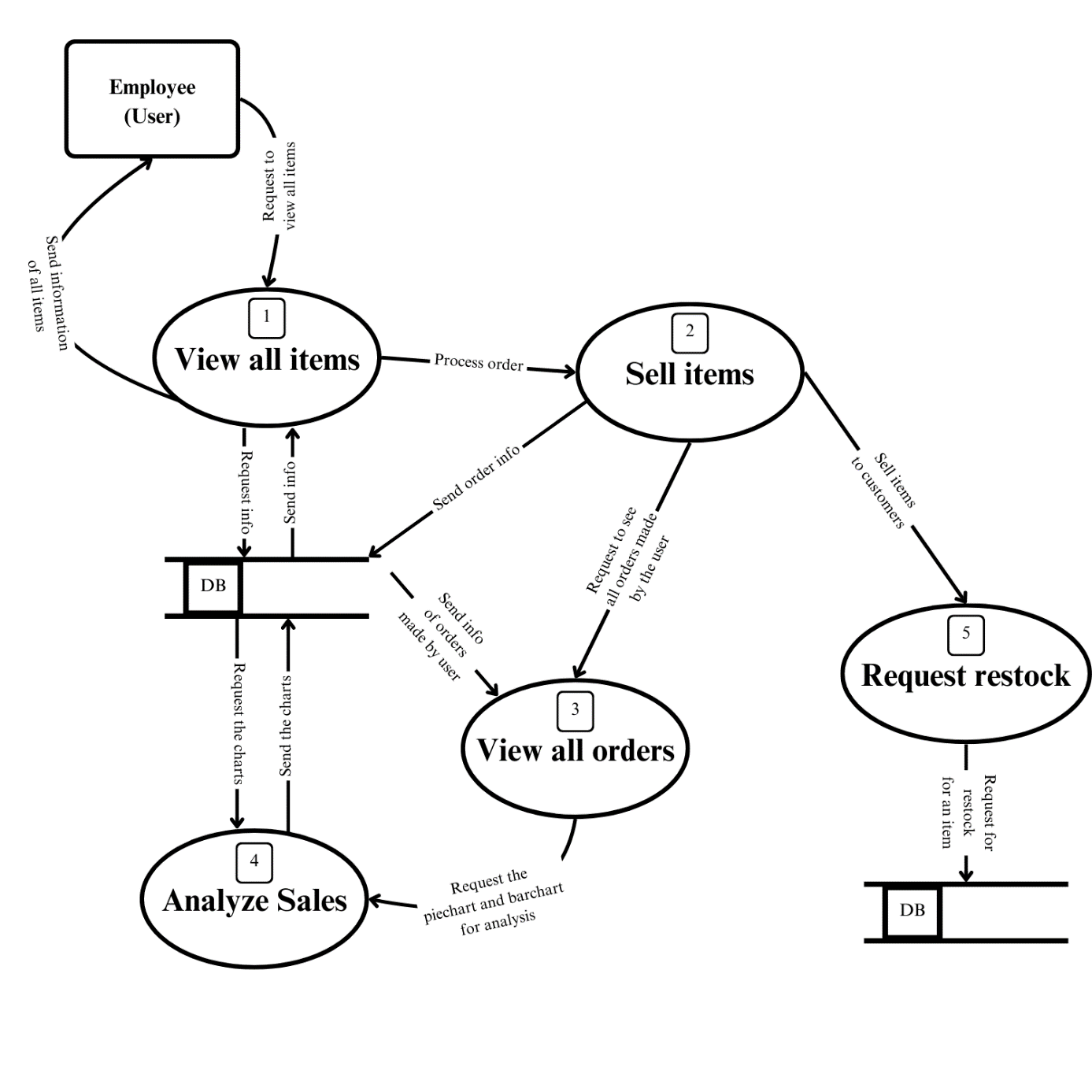
**(DFD Level 0)**

**A diagram of inventory management

Description automatically generated**

(Fig. 4.3.1)

**(DFD Level 1 – Employee (User))**

****(Fig. 4.3.2)

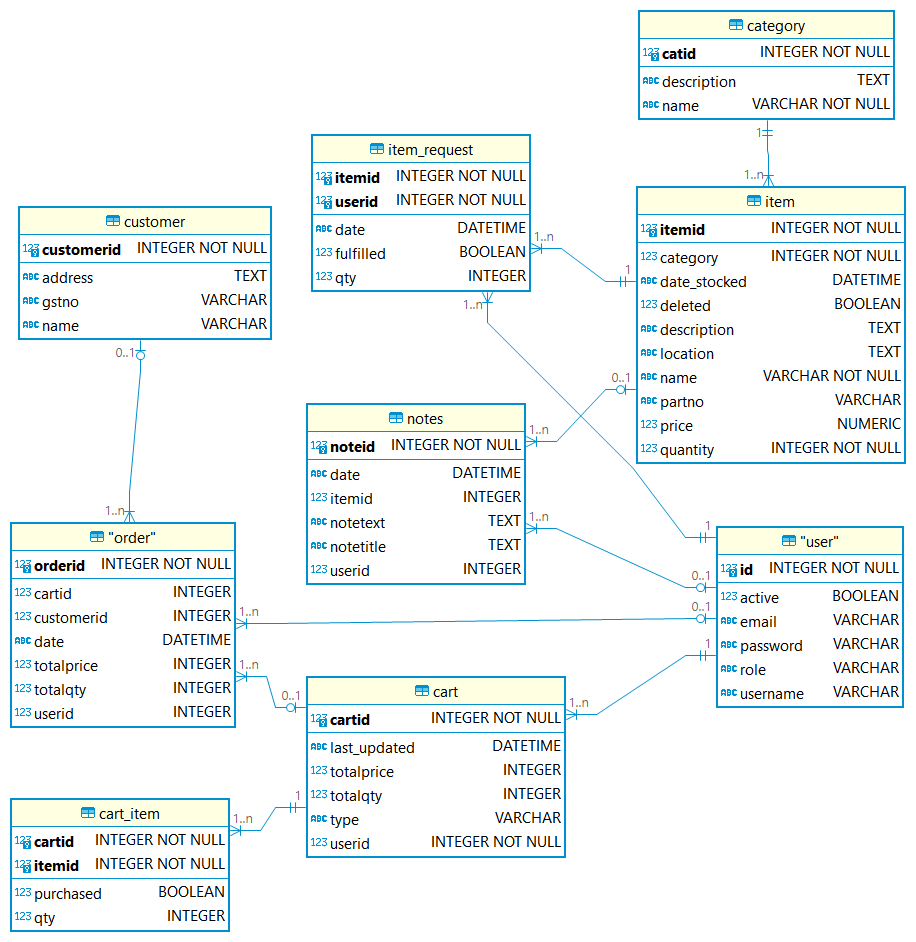
**(DFD Level 1 – Manager (Admin))**

**A diagram of a product

Description automatically generated**

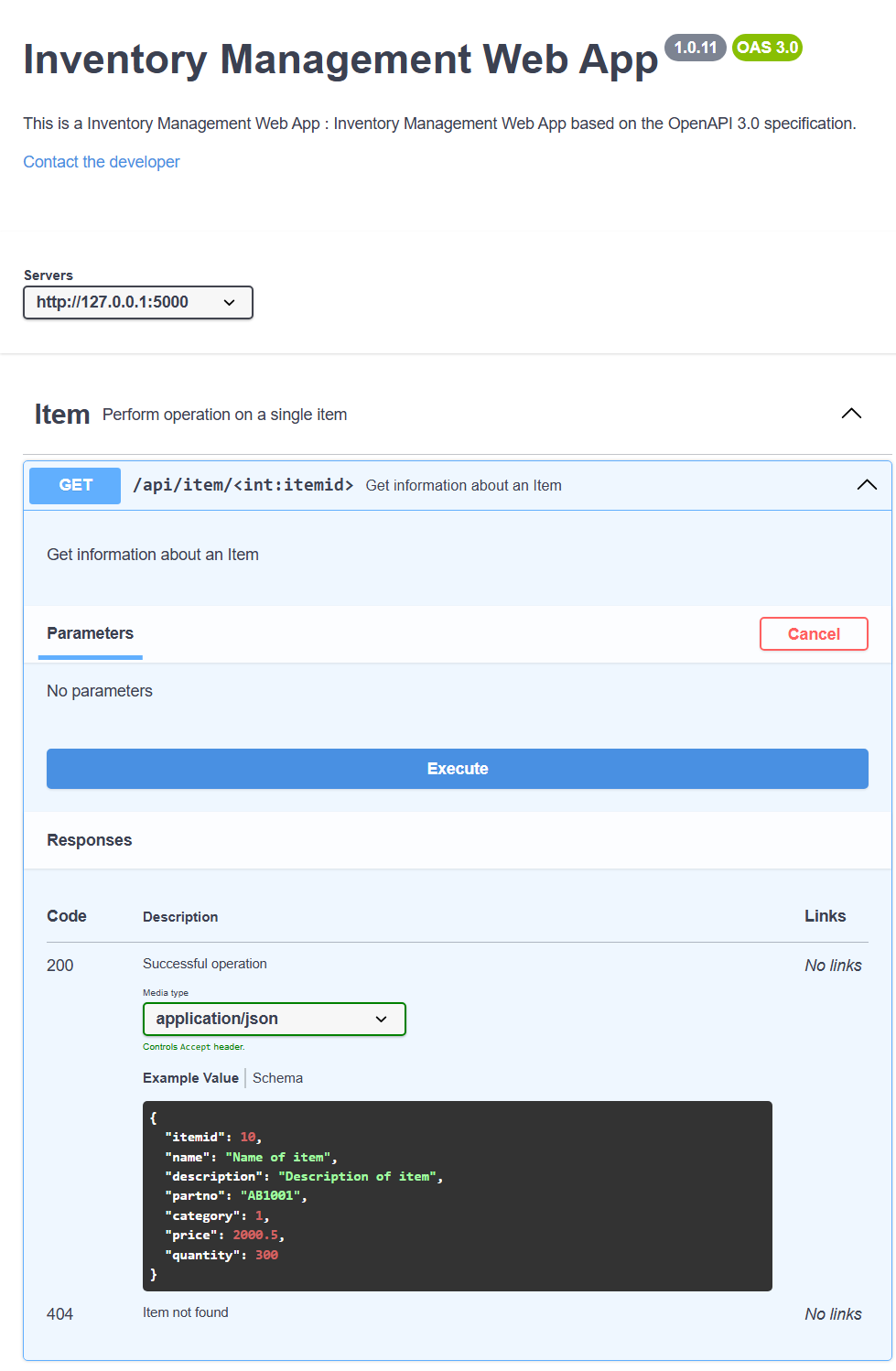
(Fig. 4.3.3)

**4.4 Database Schema**

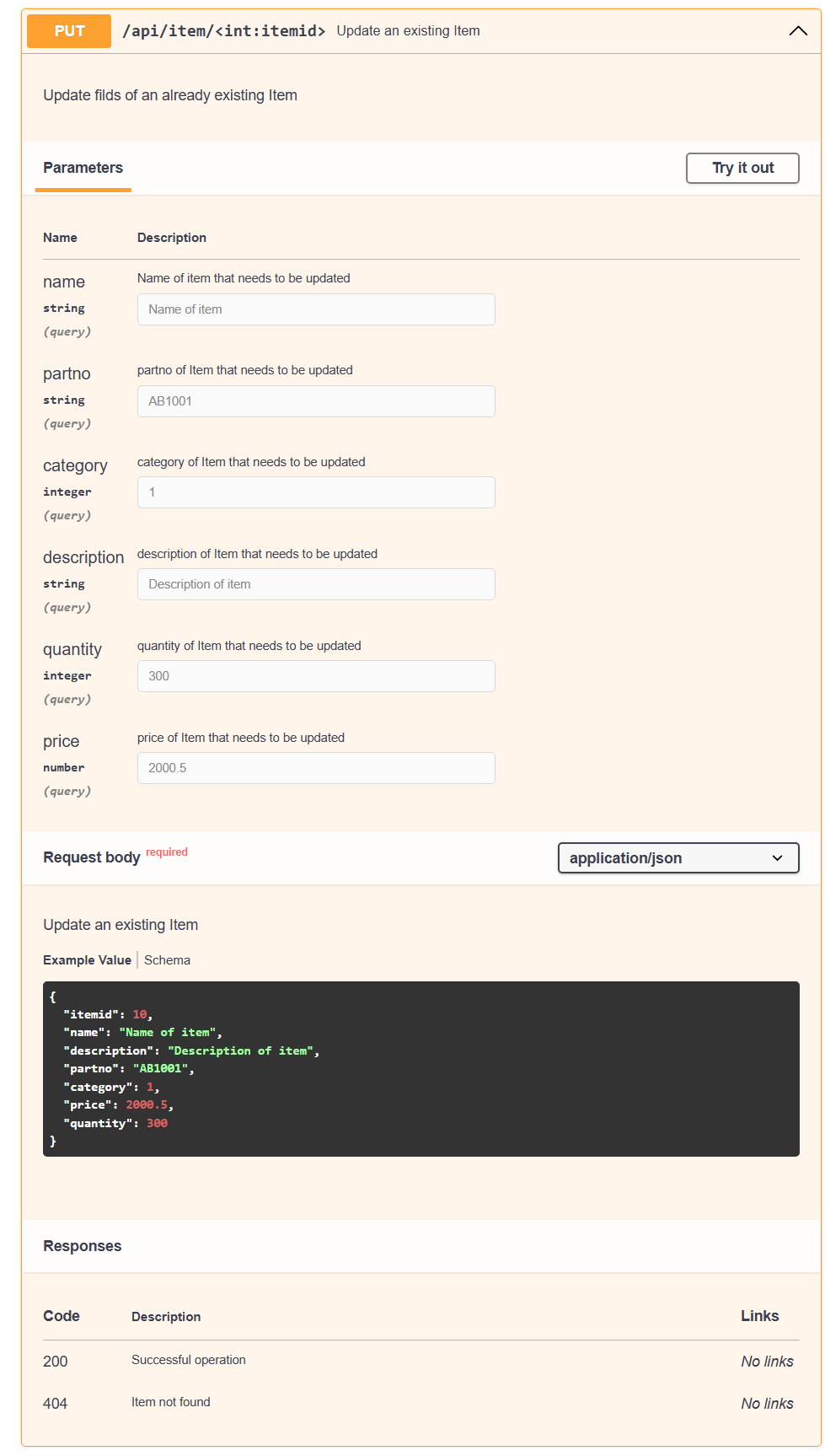
****

(Fig. 4.4)

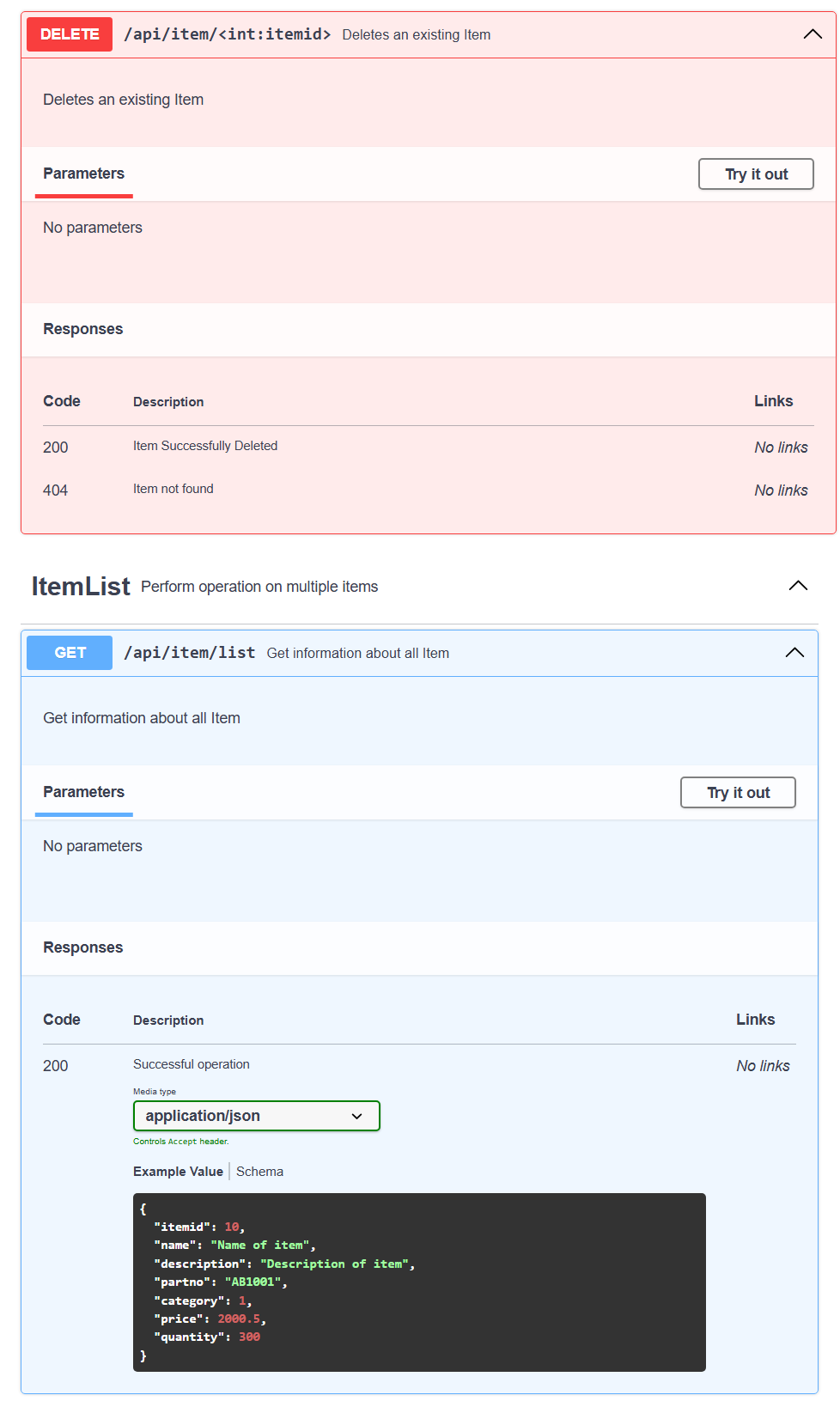
**4.5 SWAGGER Documentation (openAPI specification in YAML)**

****

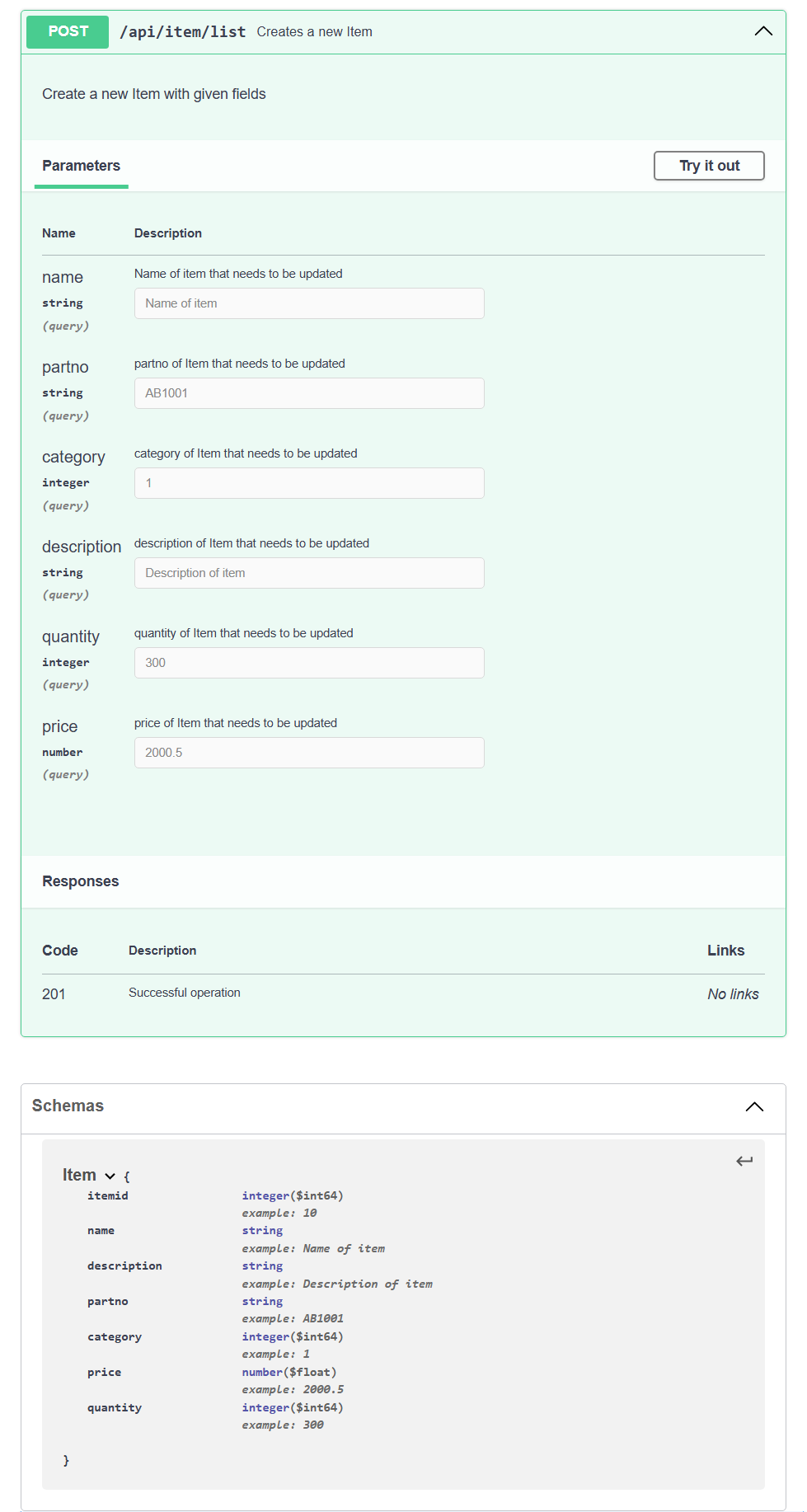
(Fig. 4.5.1)

****

(Fig. 4.5.2)

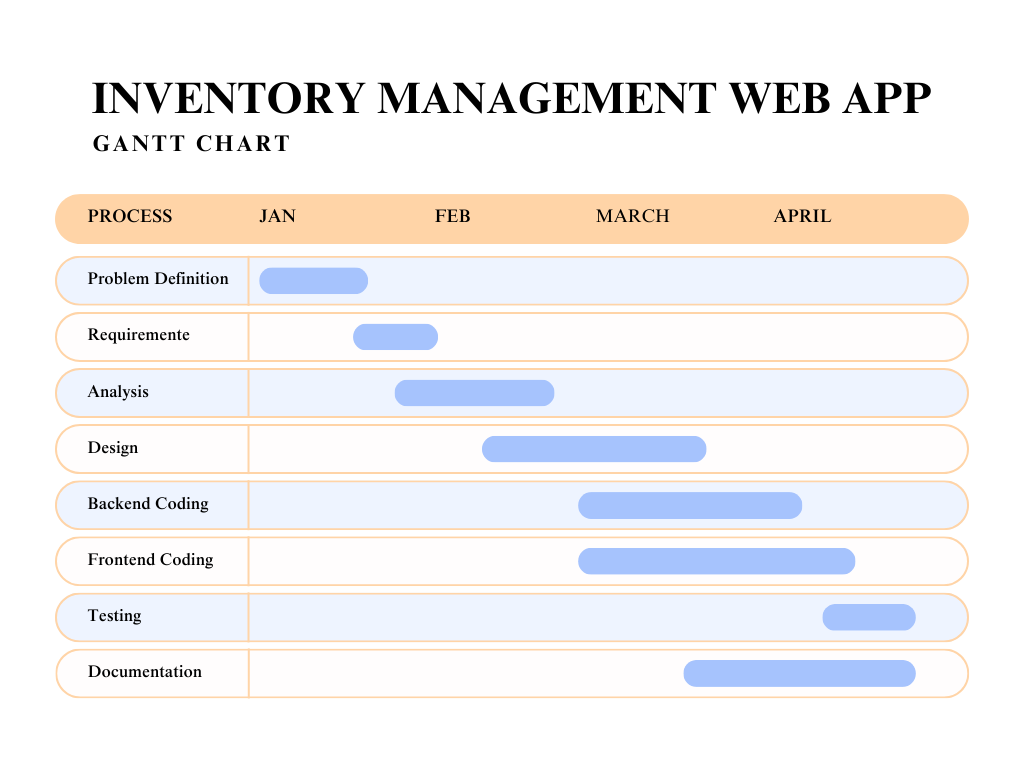
****

(Fig. 4.5.3)

****

(Fig. 4.5.4)

## 4.6 Gantt Chart



(Fig. 4.6)

**CHAPTER 5**

# IMPLEMENTATION

**Program Code :**

*from* flask *import* Flask, render\_template, url\_for, request *as* flask\_request, flash

*from* flask *import* redirect

*from* flask\_sqlalchemy *import* SQLAlchemy

*from* datetime *import* datetime, timezone

*from* model *import* db, User *as* user, Category *as* category\_model

*from* model *import* Item *as* item\_model, ItemRequest *as* item\_request

*from* model *import* Cart *as* cart\_model, CartItem *as* cart\_item, Order *as* order\_model

*from* model *import* Notes *as* note\_model, Customer *as* customer\_model

*from* flask\_login *import* LoginManager, login\_user, login\_required, logout\_user, current\_user

*import* csv

*import* os

*import* numpy *as* np

*from* flask\_restful *import* Resource, Api

*from* matplotlib *import* pyplot *as* plt

*import* matplotlib

matplotlib.use('Agg')

lm = LoginManager()

lm.login\_view = 'login'

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///database.db'

app.config['SECRET\_KEY'] = 'inventory'

lm.init\_app(app)

db.init\_app(app)

app.app\_context().push()

api = Api(app)

*with* app.app\_context():

    db.create\_all()

@lm.user\_loader

*def* load\_user(id):

*return* user.query.get(int(id))

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

*def* index():

*return* render\_template('index.html')

@app.route('/signup', methods=['GET', 'POST'])

*def* signup():

*if* flask\_request.method == 'GET':

*return* render\_template('signup.html')

*elif* flask\_request.method == 'POST':

        name = flask\_request.form.get('name')

        email = flask\_request.form.get('email')

        pw = flask\_request.form.get('password')

        u = user.query.filter\_by(email=email).first()

*if* not u:

            u\_ser = user(username=name,

                         email=email, password=pw, role='user')

            db.session.add(u\_ser)

            db.session.commit()

*return* redirect(url\_for('login'))

*else*:

*return* redirect(url\_for('signup'))

*return* render\_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])

*def* login():

*if* flask\_request.method == 'GET':

*return* render\_template('login.html')

*elif* flask\_request.method == 'POST':

        email = flask\_request.form.get('email')

        pw = flask\_request.form.get('password')

        u = user.query.filter\_by(email=email).first()

*if* u:

*if* u.password == pw:

                login\_user(u)

                flash(*f*'Welcome {current\_user.username}!',

                      category='success')

*return* redirect(url\_for('home'))

*else*:

                flash(*f*'Wrong password !',

                      category='danger')

*return* redirect(url\_for('login'))

*else*:

            flash(*f*'Please sign up first !',

                  category='warning')

*return* redirect(url\_for('index'))

*return* render\_template('login.html')

@app.route('/home', methods=['GET', 'POST'])

@login\_required

*def* home():

*if* current\_user.is\_authenticated == *True*:

*if* flask\_request.method == 'GET':

            items = item\_model.query.all()

            categories = category\_model.query.all()

*if* current\_user.role == 'user':

*return* render\_template('userhome.html',

                                       items=items, cats=categories)

*elif* current\_user.role == 'admin':

*return* render\_template('adminhome.html')

*elif* flask\_request.method == 'POST':

*return* redirect(url\_for('home'))

*else*:

*return* redirect(url\_for('login'))

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

@login\_required

*def* show\_all\_items():

    items = item\_model.query.all()

    cats = category\_model.query.all()

*return* render\_template('showallitems.html',

                           items=items, cats=cats, currentuser=current\_user)

@app.route('/item/<int:itemid>')

@login\_required

*def* showitem(itemid):

    item = item\_model.query.filter\_by(itemid=itemid).first()

    cats = category\_model.query.all()

*if* current\_user.role == 'user':

        notes = note\_model.query.filter\_by(userid=current\_user.id).all()

*elif* current\_user.role == 'admin':

        notes = note\_model.query.all()

*return* render\_template('item.html',

                           currentuser=current\_user, item=item, cats=cats, notes=notes)

@app.route('/requests', methods=['GET'])

@login\_required

*def* show\_all\_requests():

    users = user.query.all()

    irs = item\_request.query.all()

    items = item\_model.query.all()

    cats = category\_model.query.all()

*return* render\_template('showallrequests.html',

                           users=users, items=items, cats=cats, irs=irs)

@app.route('/add\_new\_product', methods=['GET', 'POST'])

@login\_required

*def* add\_new\_product():

    categories = category\_model.query.all()

*if* flask\_request.method == 'POST':

        name = flask\_request.form.get('name')

        partno = flask\_request.form.get('partno')

        qty = flask\_request.form.get('qty')

        category = flask\_request.form.get('category')

        already = item\_model.query.filter\_by(name=name, partno=partno).first()

*if* already:

*return* render\_template('addnewproduct.html', categories=categories)

*else*:

            current\_time = datetime.now(timezone.utc)

            it = item\_model(name=name, partno=partno, quantity=qty,

                            category=category, date\_stocked=current\_time)

            db.session.add(it)

            db.session.commit()

            flash(*f*'You have added {it.name} item !', category='success')

*return* redirect(url\_for('home'))

*return* render\_template('addnewproduct.html', categories=categories)

@app.route('/add\_new\_category', methods=['GET', 'POST'])

@login\_required

*def* add\_new\_category():

*if* flask\_request.method == 'POST':

        category = flask\_request.form.get('category')

        desc = flask\_request.form.get('description')

        already = category\_model.query.filter\_by(name=category).first()

*if* already:

*return* redirect(url\_for('home'))

*else*:

            c = category\_model(name=category, description=desc)

            db.session.add(c)

            db.session.commit()

            flash(*f*'You have added {c.name} category !', category='success')

*return* redirect(url\_for('home'))

*return* render\_template('addnewcategory.html')

@app.route('/<int:id>/update/item', methods=['GET', 'POST'])

@login\_required

*def* update(id):

    it = item\_model.query.filter\_by(itemid=id).first()

    categories = category\_model.query.all()

*if* flask\_request.method == 'GET':

*return* render\_template('updateitem.html', item=it, categories=categories)

*elif* flask\_request.method == 'POST':

        name = flask\_request.form.get('name')

        partno = flask\_request.form.get('partno')

        qty = flask\_request.form.get('qty')

        category = flask\_request.form.get('category')

        current\_time = datetime.now(timezone.utc)

        it = item\_model.query.filter\_by(itemid=id)

        it.update({item\_model.name: name, item\_model.partno: partno,

                   item\_model.quantity: qty, item\_model.category: category,

                   item\_model.date\_stocked: current\_time})

        db.session.commit()

        flash(*f*'You have updated {it[0].name} item !',

              category='success')

*return* redirect(url\_for('home'))

*return* render\_template('updateitem.html', item=it, categories=categories)

@app.route('/<int:catid>/update/category', methods=['GET', 'POST'])

@login\_required

*def* updatecategory(catid):

    c = category\_model.query.filter\_by(catid=catid).first()

*if* flask\_request.method == 'GET':

*return* render\_template('updatecategory.html', cat=c)

*elif* flask\_request.method == 'POST':

        category = flask\_request.form.get('category')

        description = flask\_request.form.get('description')

        c = category\_model.query.filter\_by(catid=catid)

        c.update({category\_model.name: category,

                 category\_model.description: description})

        db.session.commit()

        flash(*f*'You have updated {c[0].name} !',

              category='success')

*return* redirect(url\_for('home'))

*return* render\_template('updatecategory.html', cat=c)

@app.route('/categories', methods=['GET'])

@login\_required

*def* show\_all\_categories():

    cats = category\_model.query.all()

*return* render\_template('showallcategories.html', cats=cats, currentuser=current\_user)

@app.route('/category/<name>', methods=['GET'])

@login\_required

*def* opencategory(name):

    currentcat = category\_model.query.filter\_by(name=name).first()

    items = item\_model.query.filter\_by(

        category=currentcat.catid).all()

*if* current\_user.role == 'admin':

*return* render\_template('category.html', cat=currentcat, items=items)

*else*:

*return* render\_template('usercategory.html', cat=currentcat, items=items)

@app.route('/<int:userid>/request/<int:itemid>/accept')

@login\_required

*def* acceptrequest(userid, itemid):

    ir = item\_request.query.filter\_by(userid=userid, itemid=itemid).first()

    ir.fulfilled = *True*

    it = item\_model.query.filter\_by(itemid=itemid).first()

    q = it.quantity

    q += int(ir.qty)

    it.quantity = q

    db.session.commit()

    flash(*f*'You have accepted request of {ir.userid} of {it.name} item !',

          category='success')

*return* redirect(url\_for('show\_all\_requests'))

@app.route('/<int:userid>/request/<int:itemid>/reject')

@login\_required

*def* rejectrequest(userid, itemid):

    ir = item\_request.query.filter\_by(userid=userid, itemid=itemid).first()

    us, it = ir.userid, ir.itemid

    db.session.delete(ir)

    db.session.commit()

    flash(*f*'You have rejected request of {us} of {it} item !',

          category='warning')

*return* redirect(url\_for('show\_all\_requests'))

@app.route('/<int:id>/stock', methods=['GET', 'POST'])

@login\_required

*def* stock(id):

    it = item\_model.query.filter\_by(itemid=id).first()

    categories = category\_model.query.filter\_by(catid=it.category)

*if* flask\_request.method == 'POST':

        stqty = flask\_request.form.get('stqty')

        current\_time = datetime.now(timezone.utc)

        it = item\_model.query.filter\_by(itemid=id)

        new\_qty = int(it[0].quantity)+int(stqty)

        it.update({item\_model.quantity: new\_qty,

                  item\_model.date\_stocked: current\_time})

        db.session.commit()

        flash(*f*'You have added {stqty} Qty of{it[0].name} item !',

              category='success')

*return* redirect(url\_for('home'))

*return* render\_template('stockitem.html',

                           item=it, categories=categories)

@app.route('/<int:id>/sell', methods=['GET', 'POST'])

@login\_required

*def* sell(id):

    it = item\_model.query.filter\_by(itemid=id).first()

    categories = category\_model.query.filter\_by(catid=it.category)

*if* flask\_request.method == 'POST':

        stqty = flask\_request.form.get('sellqty')

        it = item\_model.query.filter\_by(itemid=id)

        new\_qty = int(it[0].quantity)-int(stqty)

        it.update({item\_model.quantity: new\_qty})

        db.session.commit()

        name = it.name

        flash(*f*'You have sold {stqty} Qty of{name} item !',

              category='success')

*return* redirect(url\_for('home'))

*return* render\_template('sellitem.html',

                           item=it, categories=categories)

@app.route('/<int:id>/delete')

@login\_required

*def* delete(id):

    it = item\_model.query.get\_or\_404(id)

    name = it.name

    db.session.query(cart\_item).filter\_by(itemid=it.itemid).delete()

    db.session.delete(it)

    db.session.commit()

    flash(*f*'You have deleted {name} item !', category='danger')

*return* redirect(url\_for('home'))

@app.route('/<int:id>/request', methods=['GET', 'POST'])

@login\_required

*def* request(id):

*if* flask\_request.method == 'POST':

        ir = item\_request.query.filter\_by(

            userid=current\_user.id, itemid=id).first()

*if* ir:

            q = ir.qty

*if* q == *None* or q == '':

                q = 0

            qty = flask\_request.form.get('qty')

*if* qty == *None* or qty == '':

                qty = 0

            q += int(qty)

            ir.qty = q

            db.session.commit()

*return* redirect(url\_for('home'))

*else*:

            qty = flask\_request.form.get('qty')

*if* qty == *None* or qty == '':

                qty = 0

            itr = item\_request(userid=current\_user.id, itemid=id, qty=qty)

            db.session.add(itr)

            db.session.commit()

            flash('You have requested the item ! Please wait for response !',

                  category='success')

*return* redirect(url\_for('home'))

*return* redirect(url\_for('home'))

@app.route('/cart/add/<int:item\_id>', methods=['GET', 'POST'])

@login\_required

*def* add\_item\_cart(item\_id):

    usercart = cart\_model.query.filter\_by(

        userid=current\_user.id, type='current').first()

*if* not usercart:

        newcart = cart\_model(userid=current\_user.id, type='current')

        db.session.add(newcart)

        db.session.commit()

    usercart = cart\_model.query.filter\_by(

        userid=current\_user.id, type='current').first()

*if* flask\_request.method == 'GET':

*return* redirect(url\_for('cart'))

*elif* flask\_request.method == 'POST':

        already = cart\_item.query.filter\_by(

            cartid=usercart.cartid, itemid=item\_id).first()

*if* already:

            flash("You already have the item in your cart !", category='warning')

*else*:

            additem = cart\_item(cartid=usercart.cartid, itemid=item\_id)

            db.session.add(additem)

            db.session.commit()

            flash("You added the item in your cart !", category='success')

*return* redirect(url\_for('cart'))

*return* render\_template

@app.route('/cart', methods=['GET', 'POST'])

@login\_required

*def* cart():

    usercart = cart\_model.query.filter\_by(

        userid=current\_user.id, type='current').first()

*if* not usercart:

        newcart = cart\_model(userid=current\_user.id, type='current')

        db.session.add(newcart)

        db.session.commit()

    usercart = cart\_model.query.filter\_by(

        userid=current\_user.id, type='current').first()

    cartitems = cart\_item.query.filter\_by(cartid=usercart.cartid).all()

    items = item\_model.query.all()

*if* flask\_request.method == 'GET':

*return* render\_template('cart.html',

                               usercart=usercart, cartitems=cartitems, items=items)

*elif* flask\_request.method == 'POST':

        incqty = flask\_request.form.get('qty')

*return* render\_template('cart.html',

                               usercart=usercart, cartitems=cartitems, items=items)

*return* render\_template('cart.html',

                           usercart=usercart, cartitems=cartitems, items=items)

@app.route('/cart/update/<int:itemid>', methods=['GET', 'POST'])

@login\_required

*def* updatecart(itemid):

    currentcart = cart\_model.query.filter\_by(

        userid=current\_user.id, type='current').first()

    cartitem = cart\_item.query.filter\_by(

        cartid=currentcart.cartid, itemid=itemid).first()

*if* flask\_request.method == 'POST':

        incqty = flask\_request.form.get('incqty')

        cartitem.qty = incqty

        db.session.commit()

*return* redirect(url\_for('cart'))

*return* redirect(url\_for('cart'))

@app.route('/cart/delete/<int:itemid>')

@login\_required

*def* deletecartitem(itemid):

    currentcart = cart\_model.query.filter\_by(

        userid=current\_user.id, type='current').first()

    cartitem = cart\_item.query.filter\_by(

        cartid=currentcart.cartid, itemid=itemid).first()

    db.session.delete(cartitem)

    db.session.commit()

*return* redirect(url\_for('cart'))

@app.route('/checkout/<int:cartid>', methods=['GET', 'POST'])

@login\_required

*def* checkout(cartid):

    cartitems = cart\_item.query.filter\_by(cartid=cartid).all()

    totalqty, totalprice = 0, 0.0

*for* i *in* cartitems:

        item = item\_model.query.filter\_by(itemid=i.itemid).first()

        totalqty += i.qty

        totalprice += float(i.qtyitem.price)

    items = item\_model.query.all()

*for* ci *in* cartitems:

        item = item\_model.query.filter\_by(itemid=ci.itemid).first()

*if* ci.qty > item.quantity:

            flash(*f*'The quantity of {item.name} is unavailable ! Please reduce the quantity ! Available quantity : {item.quantity}...',

                  category='danger')

*return* redirect(url\_for('cart'))

*if* flask\_request.method == 'GET':

*return* render\_template('checkout.html', cartid=cartid,

                               cartitems=cartitems, items=items,

                               totalqty=totalqty, totalprice=totalprice)

*elif* flask\_request.method == 'POST':

*try*:

            cname = flask\_request.form.get('customername')

*except*:

            cname = ''

*try*:

            caddress = flask\_request.form.get('customeraddress')

*except*:

            caddress = ''

*try*:

            cgstin = flask\_request.form.get('customergstin')

*except*:

            cgstin = ''

        existingc = customer\_model.query.filter\_by(name=cname).first()

*if* not existingc:

            existingc = customer\_model(

                name=cname, address=caddress, gstno=cgstin)

            db.session.add(existingc)

            db.session.commit()

*return* redirect(url\_for('checkoutcomplete', cartid=cartid,

                                customerid=existingc.customerid))

@app.route('/checkout/<int:cartid>/complete/<int:customerid>')

@login\_required

*def* checkoutcomplete(cartid, customerid):

    cart = cart\_model.query.filter\_by(cartid=cartid).first()

    cart.type = 'ordered'

    cartitems = cart\_item.query.filter\_by(cartid=cartid).all()

    totalprice = 0.0

*for* i *in* cartitems:

        item = item\_model.query.filter\_by(itemid=i.itemid).first()

        totalprice += float(item.price)int(i.qty)

        item.quantity -= i.qty

        i.purchased = *True*

        db.session.commit()

    order = order\_model(cartid=cartid, totalqty=len(cartitems),

                        totalprice=totalprice, userid=current\_user.id,

                        customerid=customerid)

    db.session.add(order)

    db.session.commit()

*return* redirect(url\_for('orders'))

@app.route('/orders')

@login\_required

*def* orders():

    cartitems = cart\_item.query.all()

    items = item\_model.query.all()

    customers = customer\_model.query.all()

*if* current\_user.role == 'user':

        userorders = order\_model.query.filter\_by(userid=current\_user.id).all()

*return* render\_template('order.html',

                               userorders=userorders, cartitems=cartitems,

                               items=items, currentuser=current\_user,

                               customers=customers)

*elif* current\_user.role == 'admin':

        userorders = order\_model.query.all()

        users = user.query.all()

*return* render\_template('order.html',

                               userorders=userorders, cartitems=cartitems,

                               items=items, currentuser=current\_user,

                               users=users, customers=customers)

@app.route('/addnote/<int:itemid>', methods=['GET', 'POST'])

*def* addnote(itemid):

*if* flask\_request.method == 'POST':

        title = flask\_request.form.get('notetitle')

        text = flask\_request.form.get('notetext')

        newnote = note\_model(notetitle=title, notetext=text,

                             userid=current\_user.id, itemid=itemid)

        db.session.add(newnote)

        db.session.commit()

*return* redirect(url\_for('showitem', itemid=itemid))

@app.route('/deletenote/<int:noteid>')

*def* deletenote(noteid):

    note = note\_model.query.get\_or\_404(noteid)

    itemid = note.itemid

    db.session.delete(note)

    db.session.commit()

*return* redirect(url\_for('showitem', itemid=itemid))

@app.route('/search', methods=['GET', 'POST'])

@login\_required

*def* search():

*if* flask\_request.method == 'GET':

*return* render\_template('search.html', result='', searchterm='')

*elif* flask\_request.method == 'POST':

        searchterm = flask\_request.form.get('searchterm')

        print(searchterm)

        searchitems = item\_model.query.filter(

            item\_model.name.like(*f*'%{searchterm}%')).all()

        cats = category\_model.query.all()

        print(searchitems)

*return* render\_template('search.html',

                               results=searchitems, searchterm=searchterm,

                               cats=cats, currentuser=current\_user)

@app.route('/analyze')

@login\_required

*def* analyze():

*if* current\_user.role == 'admin':

        barchart = adminbarchart()

        piechart = adminpiechart()

*return* render\_template('analysis.html',

                               barchart=barchart, piechart=piechart)

*else*:

*return* redirect(url\_for('home'))

@app.route('/logout')

*def* logout():

    logout\_user()

*return* redirect(url\_for('index'))

*def* adminbarchart():

    categories = category\_model.query.all()

    cartitems = cart\_item.query.filter\_by(purchased=*True*).all()

    category\_counts = {}

    categorynames = []

    item\_counts = []

*for* category *in* categories:

        categorynames.append(category.name)

        item\_counts.append(0)

*if* category not in category\_counts:

            category\_counts[category.catid] = 0

*for* i *in* cartitems:

        item = item\_model.query.filter\_by(itemid=i.itemid).first()

*if* item.category not in category\_counts:

            category\_counts[item.category] = 0

        category\_counts[item.category] += 1

*for* category *in* categories:

        item\_counts[category.catid-1] = category\_counts[category.catid]

    print(item\_counts)

    print(item\_counts)

    plt.figure(figsize=(10, 6))

    plt.bar(categorynames, item\_counts, color='skyblue')

    plt.xlabel('Items Sold by category')

    plt.ylabel('Frequency')

    plt.title('Category Wise Analysis')

    plt.xticks(rotation=45, ha='right')

    plt.yticks(np.arange(0, max(item\_counts) + 1, 1))

    plt.tight\_layout()

    chart\_filename = 'admin\_bargraph.jpg'

    chart\_path = os.path.join(app.static\_folder, 'charts', chart\_filename)

    plt.savefig(chart\_path)

    plt.close()

*return* chart\_path

*def* adminpiechart():

    allorders = order\_model.query.all()

    user\_orders = {}

*for* order *in* allorders:

*if* order.userid not in user\_orders:

            user\_orders[order.userid] = 0

        user\_orders[order.userid] += order.totalprice

    users = user.query.all()

    user\_list = []

    user\_total = []

*for* u *in* users:

*if* u.id not in user\_orders:

            user\_orders[u.id] = 0

        user\_list.append(u.username)

        user\_total.append(user\_orders[u.id])

    colors = ['lightskyblue', 'lightcoral',

              'lightpink', 'lightyellow', 'lightgreen']

    plt.figure(figsize=(8, 8))

    plt.pie(user\_total, labels=user\_list, colors=colors,

            autopct='%1.1f%%', startangle=140)

    plt.axis('equal')

    plt.title('Total sales by users')

    chart\_filename = 'admin\_piechart.jpg'

    chart\_path = os.path.join(app.static\_folder, 'charts', chart\_filename)

    plt.savefig(chart\_path)

    plt.close()

*return* chart\_path

*class* *ItemResources*(Resource):

*def* get(self, itemid):

        item = item\_model.query.get\_or\_404(int(itemid))

*return* {

            'itemid': item.itemid,

            'item\_name': item.name,

            'part\_no': item.partno,

            'category': item.category,

            'description': item.description,

            'quantity': item.quantity,

            'price': float(item.price),

        }

*def* delete(self, itemid):

        item = item\_model.query.get\_or\_404(int(itemid))

        db.session.query(cart\_item).filter\_by(itemid=itemid).delete()

        db.session.delete(item)

        db.session.commit()

*return* {

            'message': 'item successfully deleted !',

        }

*def* put(self, itemid):

        data = flask\_request.get\_json()

        item = item\_model.query.filter\_by(itemid=int(itemid))

        item.update({

            item\_model.name: data['name'],

            item\_model.partno: data['partno'],

            item\_model.category: data['category'],

            item\_model.description: data['description'],

            item\_model.quantity: data['quantity'],

            item\_model.price: float(data['price']), })

        db.session.commit()

*return* {'message': 'item successfully updated !'}

*class* *ItemListResources*(Resource):

*def* get(self):

        items = item\_model.query.all()

        result = []

*for* item *in* items:

            result.append({

                'itemid': item.itemid,

                'item\_name': item.name,

                'part\_no': item.partno,

                'category': item.category,

                'quantity': item.quantity,

                'price': float(item.price),

            })

*return* result

*def* post(self):

        data = flask\_request.get\_json()

        newitem = item\_model(

            name=data['name'],

            partno=data['partno'],

            category=data['category'],

            description=data['description'],

            quantity=data['quantity'],

            price=float(data['price'])

        )

        db.session.add(newitem)

        db.session.commit()

*return* {'message': 'new item added !'}

api.add\_resource(ItemResources, '/api/item/<itemid>')

api.add\_resource(ItemListResources, '/api/item/list')

*def* datagen():

    u = user.query.all()

*if* not u:

        us1 = user(username='admin', email="admin@gmail.com",

                   password='password', role='admin')

        db.session.add(us1)

        us2 = user(username='user', email="user1@gmail.com",

                   password='password', role='user')

        db.session.add(us2)

        db.session.commit()

    c = category\_model.query.all()

*if* not c:

*with* open('./static/category.csv', newline='') *as* csvfile:

            reader = csv.reader(csvfile)

*for* row *in* reader:

                des = category\_model(

                    name=row[0], description=row[1])

                db.session.add(des)

            db.session.commit()

    i = item\_model.query.all()

*if* not i:

*with* open('./static/item.csv', newline='') *as* csvfile:

            reader = csv.reader(csvfile)

*for* row *in* reader:

                des = item\_model(

                    name=row[0], partno=row[1],

                    description=row[2], quantity=row[3],

                    price=row[4], category=row[5])

                db.session.add(des)

            db.session.commit()

datagen()

*if* \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=*True*)

**Testing :**

A screenshot of a computer

Description automatically generated

(Landing page – ability to login as existing user or sign up as new user-employee)

A screenshot of a computer

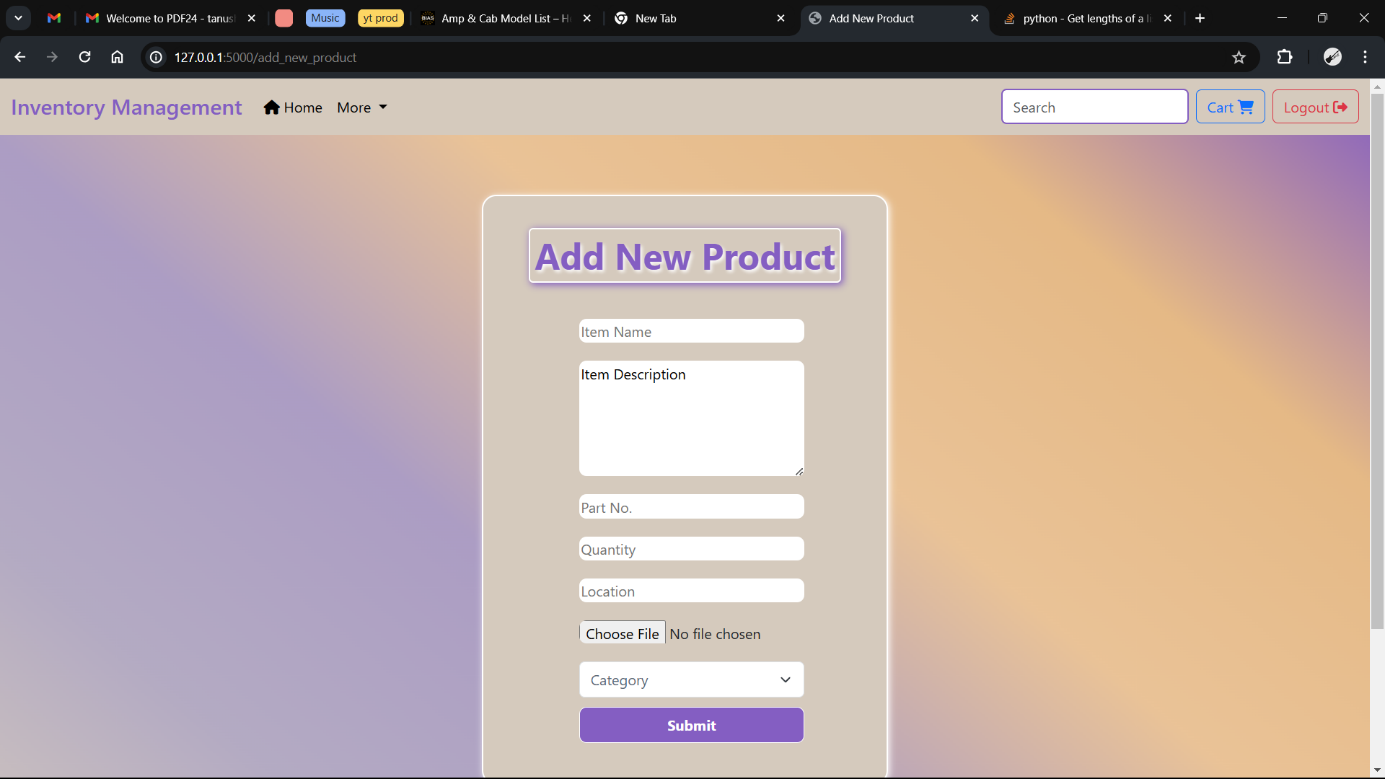
Description automatically generated

(Login using email id and password)

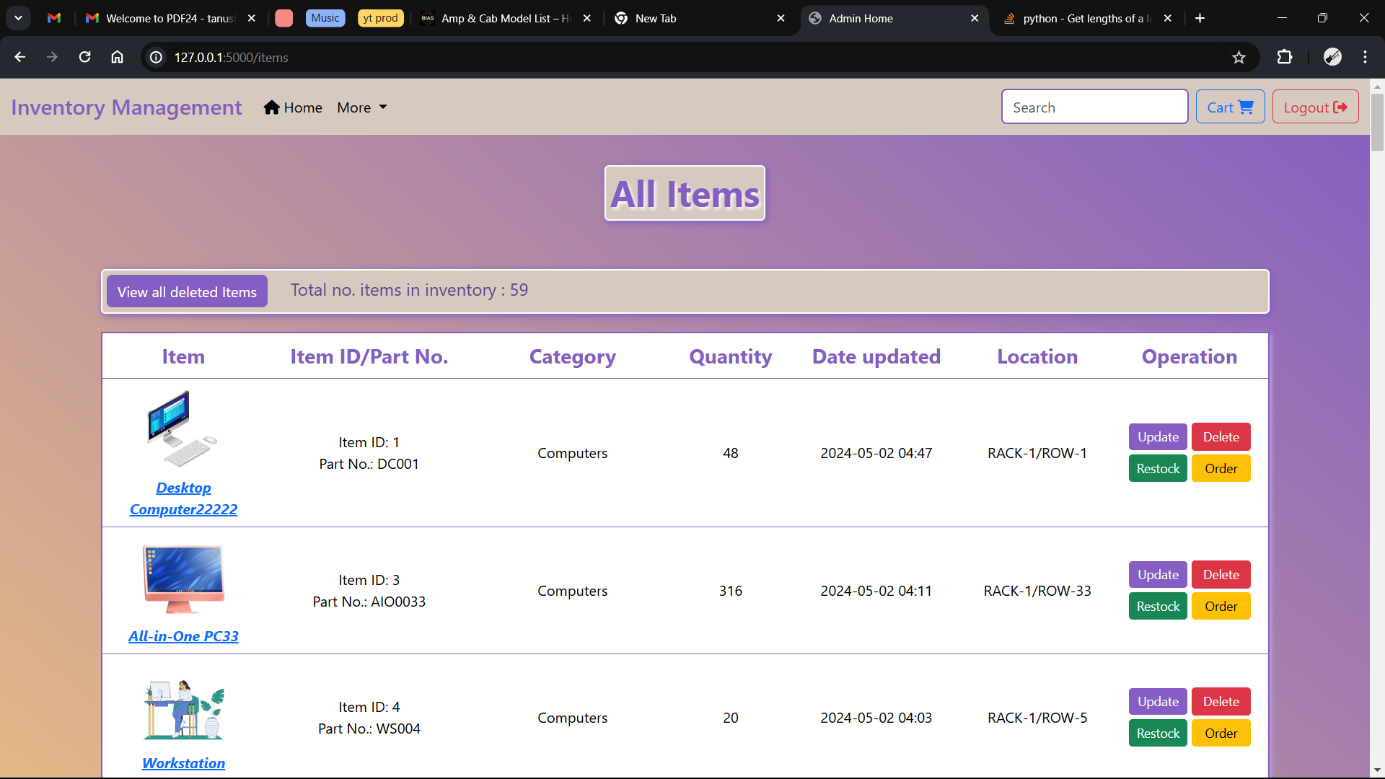
A screenshot of a computer

Description automatically generated

(Home page for admin/manager)



(Adding new item/products in the inventory – Admin feature)



(View all the items in the inventory)

A screenshot of a computer

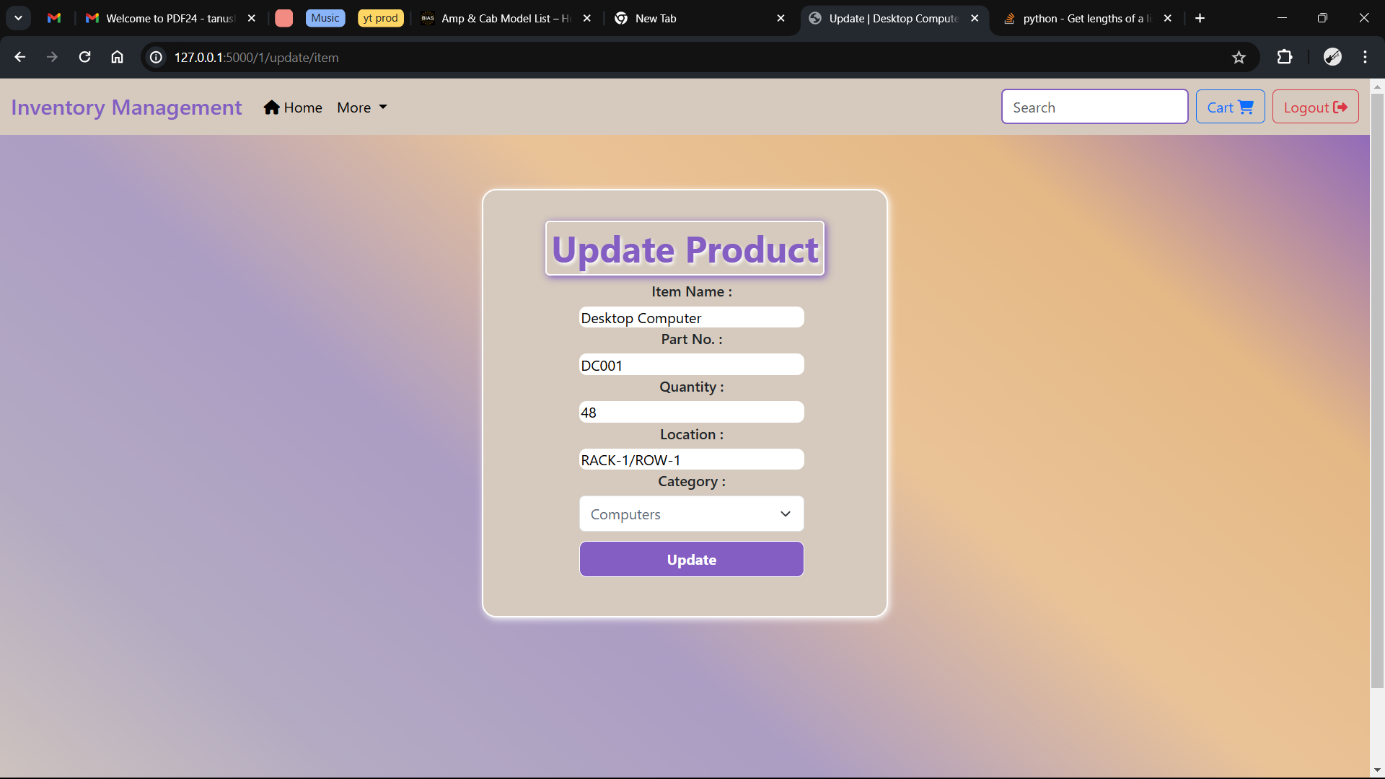
Description automatically generated

(Adding new category in the inventory – Admin feature)

A screenshot of a computer

Description automatically generated

(View all the categories in the inventory)



(Update the product details – Admin feature)

A screenshot of a computer

Description automatically generated

(Search for an item)

A screenshot of a computer

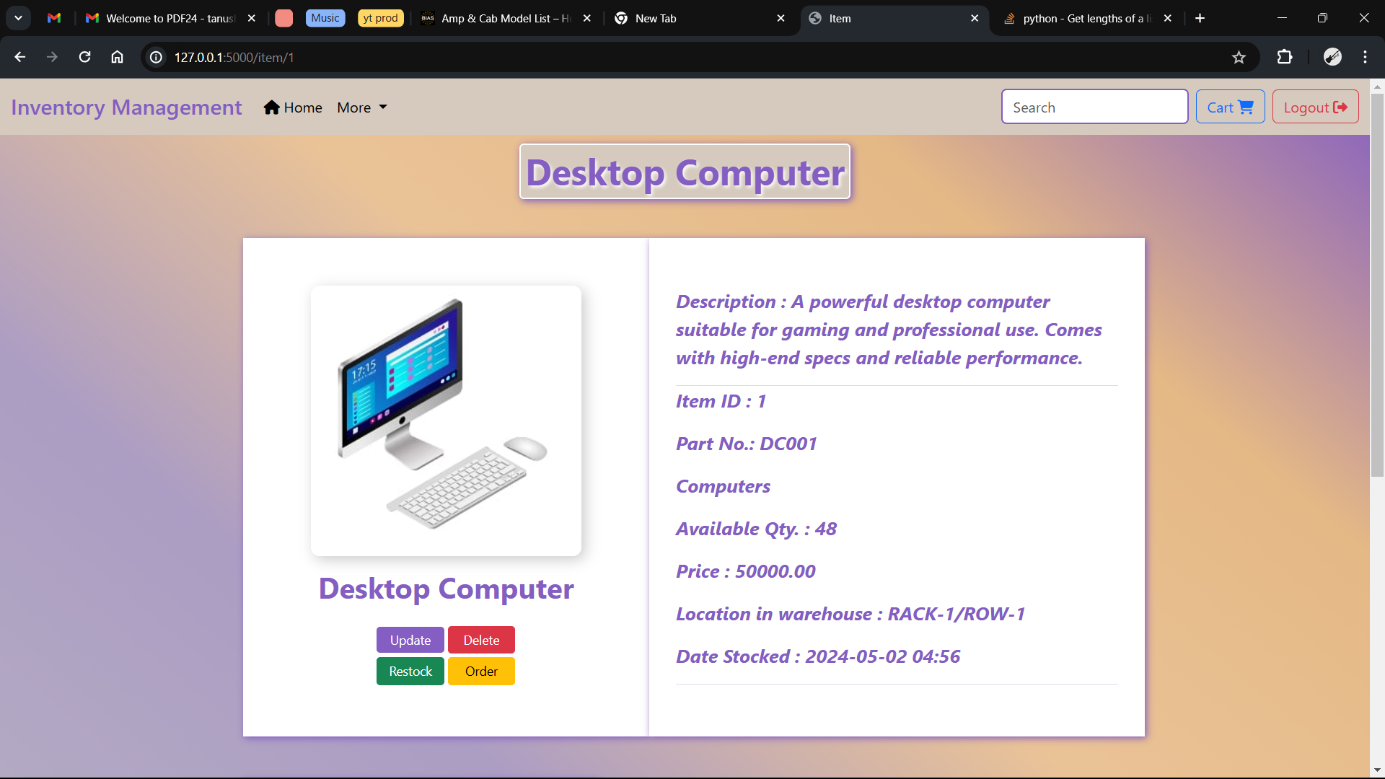
Description automatically generated

(Home page for Employee/user)

A screenshot of a computer

Description automatically generated

(View all items in the inventory)

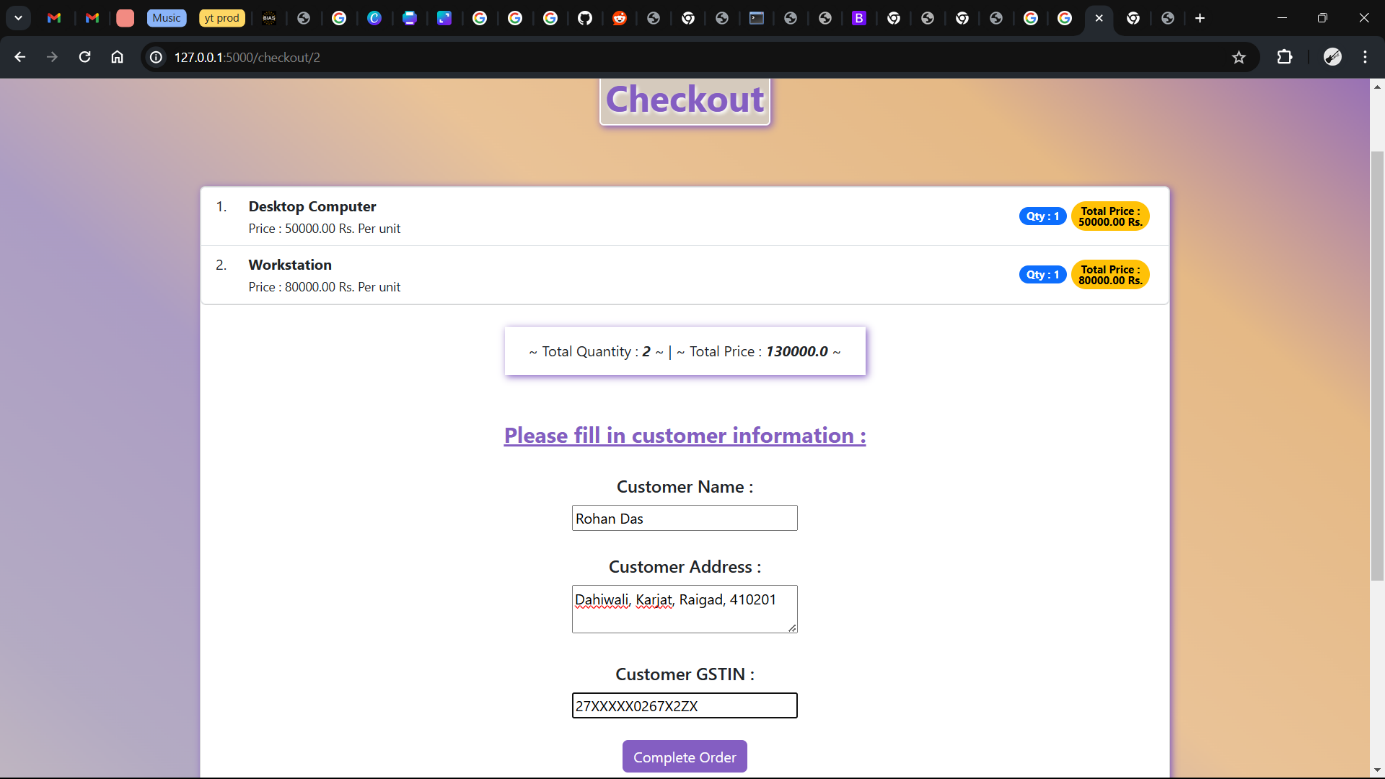


(View dedicated page for an item and add notes)

A screenshot of a computer

Description automatically generated

(Add items to cart for order processing)



(Checkout page for entering the customer details)

A screenshot of a computer

Description automatically generated

(View all the orders made by employee and their data)

A screenshot of a computer

Description automatically generated

(View the analytics for sales – user wise and category wise)

A screenshot of a computer

Description automatically generated

(API implementation – GET operation on item to get single item)

A screenshot of a computer

Description automatically generated

(API implementation – GET operation on item to get multiple item)

### CHAPTER 6

### FUTURE ENHANCEMENT

**1. Barcode Scanning Integration:** Enable users to scan barcodes using their smartphones or dedicated scanners to quickly add or update inventory items, reducing manual data entry and improving accuracy.

**2. Automatic Low Stock Alerts:** Implement automatic notifications to alert users when inventory levels fall below a certain threshold, prompting timely reordering and preventing stockouts.

**3. Customizable Dashboard Widgets:** Allow users to customize their dashboard by adding, removing, and rearranging widgets to display the most relevant inventory metrics and reports for their specific needs.

**4. Mobile-Friendly Design:** Optimize the web app for mobile devices to ensure seamless access and functionality on smartphones and tablets, enabling users to manage inventory on the go.

**5. Integration with Accounting Software:** Integrate the web app with popular accounting software like QuickBooks or Xero to synchronize inventory data and streamline financial reporting and reconciliation processes.

**6. Multi-Language Support:** Offer support for multiple languages to accommodate users from different regions and facilitate international expansion, enhancing accessibility and usability for a diverse user base.

**7. Bulk Import/Export Functionality:** Enable users to import/export inventory data in bulk using CSV or Excel files, simplifying the process of adding new products, updating quantities, or exporting reports.

**8. User Feedback Mechanism:** Introduce a feedback mechanism within the app to gather user suggestions, feature requests, and bug reports, allowing developers to prioritize enhancements based on user needs and preferences

**CHAPTER #**

**REFERENCES**

**Journal Paper,**

1. **Design and Optimization of An Inventory Management System for Central Stores**

By Nwafor Chidinma Anulika1 , Idoko Nnmadi A2 , Agbo Jonathan Chukwuwike3 , Ogbene Nnaemeka Emeka4

ISSN: 2456-236X

International Journal of Interdisciplinary Innovative Research &Development (IJIIRD)

1. **ONLINE STOCK AND INVENTORY MANAGEMENT SYSTEM**

by Sai Prasad K1, Sai Prasad R2, Sai Rahul J3, Sai Rithick K4, Vennela P5, Sai Shashank V6, Sai Srinish G7

e-ISSN: 2582-5208

International Research Journal of Modernization in Engineering Technology and Science

**Referral links,**

Bootstrap: https://getbootstrap.com/docs/5.3/getting-started/introduction​

Web Gradients: https://webgradients.com/​

Amazon Inventory Management: https://sell.amazon.com/learn/inventory-management/

Ship rocket: https://www.shiprocket.in/inventory-management/​