Table of Contents

1. Executive Summary

The Food Delivery System is a comprehensive web application built using Flask, Google Sheets, and JavaScript, designed to streamline and automate the food ordering and delivery process. This system serves as a centralized platform that connects customers, delivery workers, and administrators through an intuitive interface while leveraging cloud-based data storage for flexibility and accessibility.

The primary purpose of this application is to digitize the entire food delivery workflow, from order placement to delivery confirmation, eliminating manual processes and paperwork. The system's scope encompasses order management, real-time tracking, worker assignment, delivery verification, and administrative oversight. Key features include secure user authentication, role-based access control, real-time synchronization with Google Sheets as the database backend, and a responsive user interface that adapts to different devices and user roles.

For customers, the system provides a seamless ordering experience with real-time updates on order status. Delivery workers benefit from clear order assignment, navigation assistance, and digital verification processes. Administrators gain comprehensive oversight of the entire operation, with tools to manage users, monitor performance, and optimize delivery logistics. By integrating these components, the Food Delivery System transforms traditional food delivery operations into a modern, efficient digital service.

2. Problem Statement

The campus lacked a dedicated food delivery system tailored to the needs of students and on-campus vendors. As a result, both students and restaurant operators faced persistent challenges that hindered convenience, efficiency, and satisfaction.

Key Challenges:

- **No Existing Digital Platform:** Students had no centralized platform to browse menus, place orders, or receive deliveries within campus boundaries.
- **Time Management Issues:** Long queues and wait times during class breaks caused significant time loss for students, affecting academic schedules and increasing stress.
- **Limited Seating Capacity:** Campus restaurants were often overcrowded, lacking sufficient space to accommodate large volumes of students during peak hours.
- **Inconvenient Access:** Some food outlets were located at a distance from academic buildings, making it difficult for students to access them within short breaks.
- **Operational Difficulties for Vendors:**
  - ✓ Restaurant staff struggled with managing peak-time demand efficiently.

- ✓ There was no delivery coordination system, leading to missed or delayed orders.
- ✓ Restaurants had limited ability to satisfy customers quickly due to space and staff limitations.

Strategic Solution:

To address these issues, a web-based food delivery system backed by Google Sheets was proposed as a cost-effective, low-maintenance alternative to a traditional database-driven solution.

Rationale for Google Sheets Integration: Familiar Interface for Non-Technical Users: Restaurant staff and administrators can easily view and manage data without requiring database expertise.

Cloud-Based Accessibility: Enables anytime, anywhere access for delivery coordination and system monitoring, especially valuable in a distributed campus setting.

Low Implementation and Maintenance Overhead: Avoids the complexity of database hosting and maintenance, making it suitable for smaller operations.

Built-In Collaboration Tools: Supports real-time updates and transparency without additional software development.

This hybrid approach enables rapid digitization of campus food services, bridging the gap between manual operations and enterprise-level platforms. It provides a scalable, accessible solution that enhances student convenience while improving vendor operations all without demanding significant technical infrastructure or staffing.

3. Software Development Approach

The development of the Food Delivery Software System was guided by the need for rapid, user-focused delivery of functional components that directly addressed real-world challenges faced by students, vendors, and administrators on campus. The system was built using a structured evolutionary prototyping model combined with modular monolithic architecture, ensuring both speed and maintainability.

Methodology and Model Given the dynamic and feedback-driven nature of the environment, the team adopted a structured evolutionary prototyping approach. This allowed for the creation of early functional prototypes that were refined through real user input. The system was developed iteratively in multiple stages:

1. **Initial Prototype (MVP**) – Focused on core features such as login, menu browsing, order placement, and basic delivery workflows.

2. **Feedback and Iteration Phase** – Incorporated usability feedback from students and restaurant operators, improving UI flow and interaction design.

3. Enhancement Phase – Introduced delivery verification codes, post-delivery rating functionality, and administrative tools for monitoring operations.

This iterative model enabled the system to evolve naturally based on user needs and operational realities, reducing rework and aligning technical development with stakeholder expectations.

Architectural Design The system is built on a modular monolithic architecture, where all core features reside within a single deployable application but are internally organized into logical modules—such as user authentication, order processing, delivery management, and administrative functions. This structure offered the following advantages:

- Simplified deployment and management
- Easier integration with a lightweight backend (Google Sheets)
- Logical separation for future scalability or service decomposition if needed

Technology Stack The technology stack was selected based on ease of development, cost-effectiveness, and minimal maintenance overhead:

- **Backend Framework:** Python's Flask was used for its lightweight nature and rapid development capabilities.
- **Frontend Technologies:** HTML5, CSS3, JavaScript, and Bootstrap ensured responsive design across devices. jQuery was used to simplify asynchronous interactions.
- **Data Storage:** Google Sheets served as the backend datastore, accessed via the Google Sheets API. This provided familiarity for non-technical users, low cost, and cloud accessibility.
- **Authentication:** User sessions were managed using Flask-Login, with bcrypt used for secure password hashing. CSRF protection was enabled for form security.
- **Development Tools:** Git (version control), GitHub (repository hosting), Visual Studio Code (IDE), Postman (API testing), and GitHub Actions (CI/CD automation) supported efficient team collaboration and continuous deployment.

Implementation Challenges: The team faced several challenges during development, primarily due to the use of Google Sheets as a backend and the distributed nature of users:

- Latency and API Limits: Google Sheets API occasionally exhibited high latency and enforced rate limits. This was mitigated using local caching, background synchronization for non-critical tasks, and batching with exponential backoff.

- Concurrent Access Conflicts: To prevent data conflicts when multiple users accessed the same sheet, optimistic concurrency controls with version checking were implemented.
- Session and Security Handling: Session timeout and refresh management were tuned for both security and user experience.
- Cross-Browser Consistency: Extensive testing and progressive enhancements ensured the application functioned reliably across a range of devices and browsers.

Despite these constraints, the system was designed and implemented to be robust, user-friendly, and maintainable. The chosen development approach ensured that essential functionality was delivered quickly while providing a foundation for future enhancements.

4. Business Requirements Analysis

4.1 Business Goals: The primary objective of the Campus Food Delivery System is to provide a simple, fast, and reliable platform that enhances food accessibility for students while improving operational efficiency for vendors and delivery personnel. The goals include:

- Improve Student Convenience
- Digitize Vendor Operations
- Empower Delivery Personnel
- Increase Operational Transparency
- Minimize Technical Overhead

4.2 Existing Environment: Prior to the implementation of the system, food ordering and delivery were handled entirely offline and informally. Students and vendors faced limitations in speed, space, and scalability.

4.3 Operational Limitations:

1. No Central Ordering Platform
2. Time and Location Constraints
3. Inefficient Vendor Workflows
4. Lack of Delivery Verification
5. Disorganized Data and Reporting

4.4 Strategic Need for a System: A centralized, digital platform was needed to coordinate orders, ensure delivery validation, and allow lightweight management of operations without heavy IT demands.

4.5 Role of the Food Delivery System: The solution enables digital ordering, structured delivery, code-based confirmation, role-specific control, and future scalability within campus constraints.

5. Overview of the Proposed System

The Campus Food Delivery System is a purpose-built web application that digitizes the food ordering and delivery workflow for university environments. It serves as a centralized hub that connects students, food vendors, delivery personnel, and administrators through a responsive, role-based interface. The system leverages cloud-based tools most notably Google Sheets to eliminate traditional logistical bottlenecks and enable a seamless, transparent experience across all user roles.

Scope of the System:

- Enables students to place orders from anywhere on campus using a digital interface.
- Allows food vendors to update menus in real time using Google Sheets.
- Provides delivery workers with assigned order details and digital verification tools.
- Grants administrative users oversight tools to manage users, monitor deliveries, and generate reports.
- Will not support off-campus delivery or integration with third-party food delivery platforms in its initial deployment.
- Target Users and Environment:
- University students needing on-demand food service during class hours.
- On-campus restaurant vendors with limited physical seating and staff.
- Designated campus delivery workers (e.g., student part-timers).
- Campus administrators responsible for operations and service quality.

6. System Requirements Specification (SRS)

Includes Functional and Non-Functional Requirements, Use Case Analysis, Behavioral and Static Modeling.

6.1 Functional Requirements

- Students can view categorized menus, place orders, and receive verification codes.

- Admins can assign orders and monitor system performance.

- Vendors update menu items via Google Sheets.

- Delivery workers access assigned orders and verify deliveries via code.

- Students can rate delivered food.

6.2 Non-Functional Requirements

- *Usability*: Mobile and desktop-friendly UI.

- *Performance*: Fast response times.

- Security: Hashed passwords, verification codes.

- *Maintainability*: Modular codebase and Google Sheets integration.

- *Scalability:* Easy to expand to more users or vendors.

6.3 Use Case Analysis

Actors:

a. Admin

b. Database Manager (Vendor)

c. Customer (Student)

d. Worker (Delivery Personnel)

Use Cases:

i. Access Verification Code

ii. Monitor Orders

iii. Add workers

iv. Rate Food

v. Place Order

vi. View Menu

vii. Login

viii. Update Price

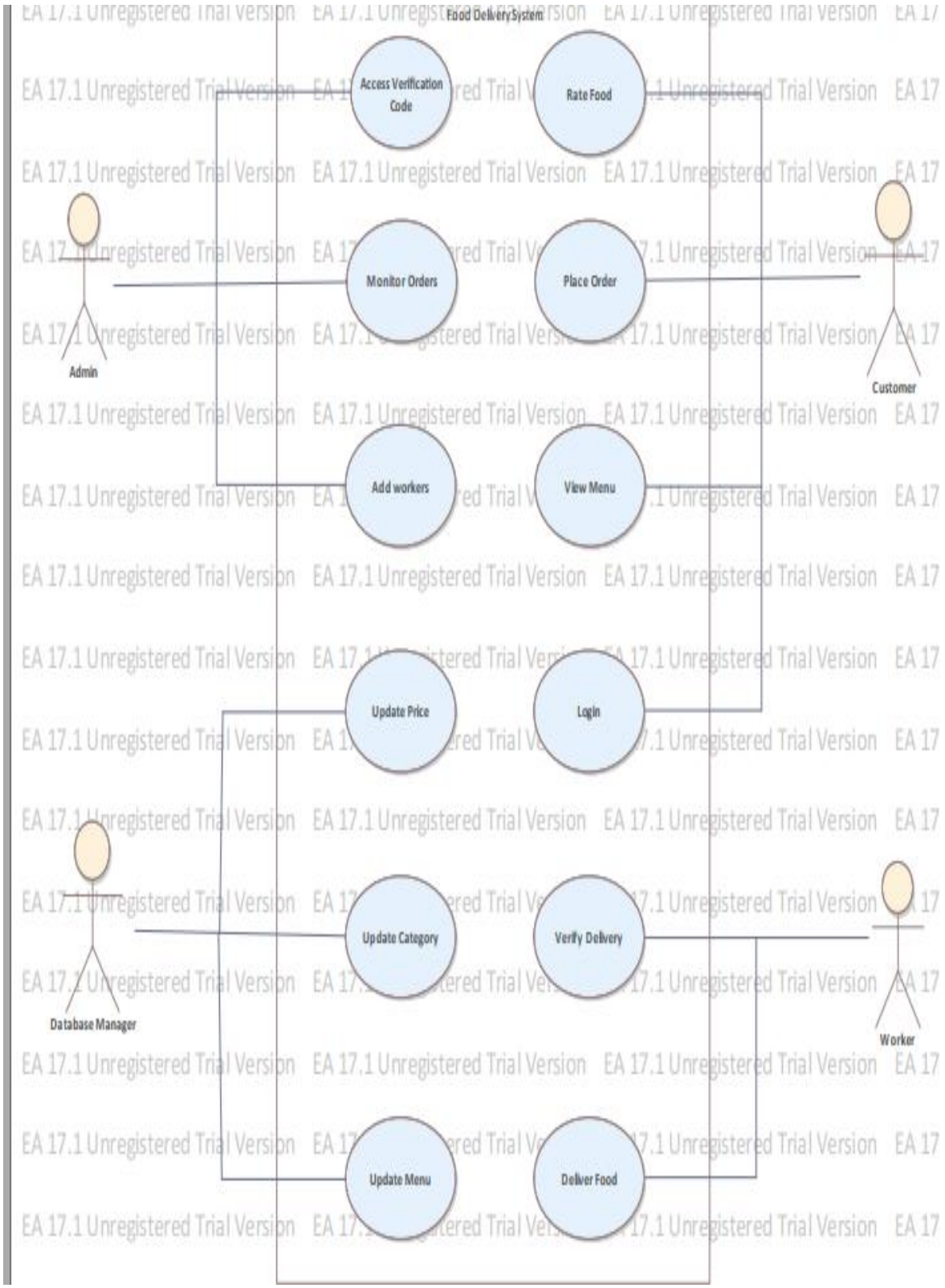
ix. Update Category

x. Update Menu

xi. Verify Delivery

xii. Deliver Food

6.3.1 Use Case Diagram



6.4 Behavioral Modeling

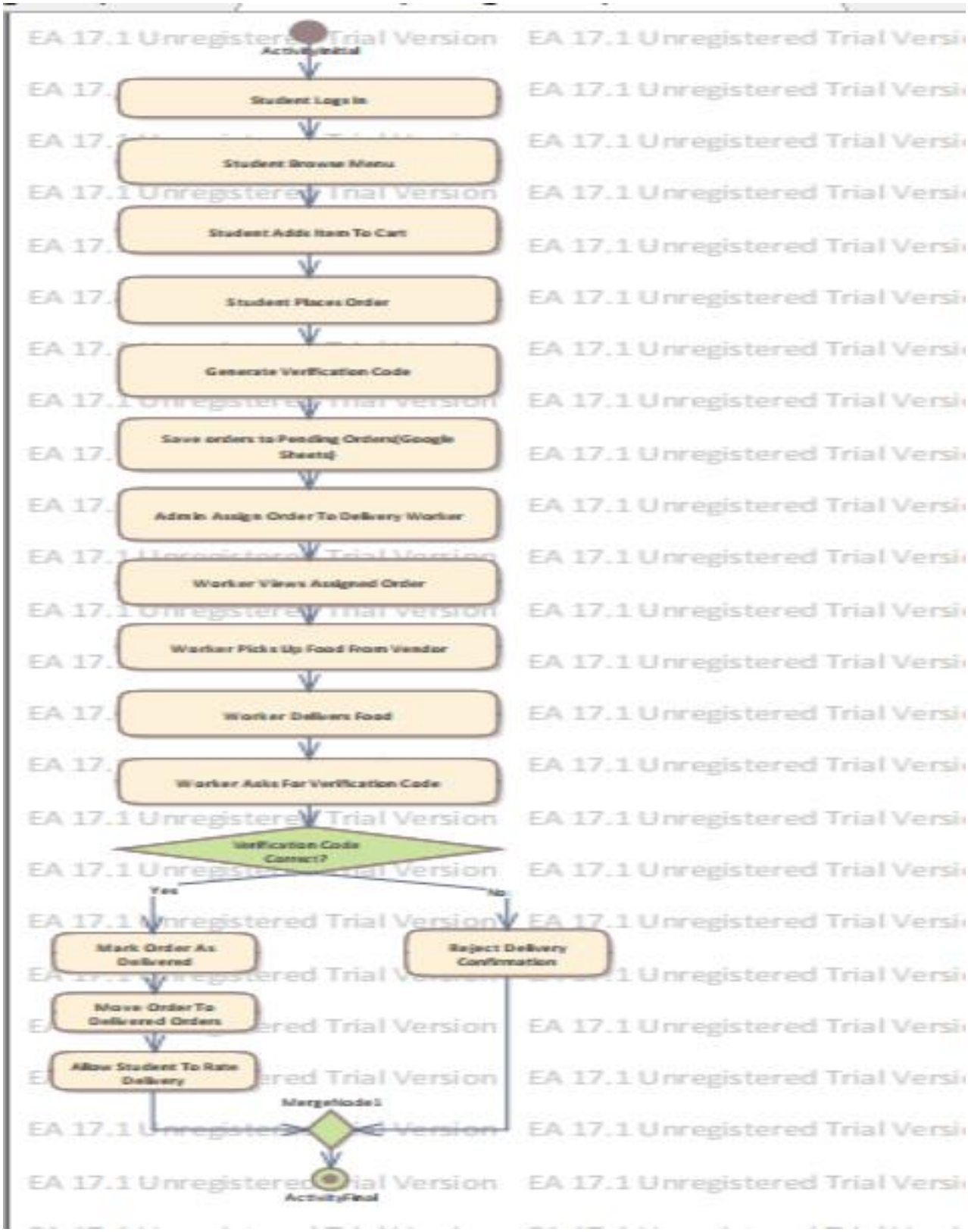
Activity Diagram Order Lifecycle:

1) Student logs in and browses menu.
2) Adds food items to cart and confirms order.
3) Student Places Order
4) Generate Verification Code
5) Order is saved to "PendingOrders".
6) Admin assigns order to a delivery worker.
7) Worker views order and location.
8) Delivers order, confirms code, marks delivered.
9) Customer submits feedback.

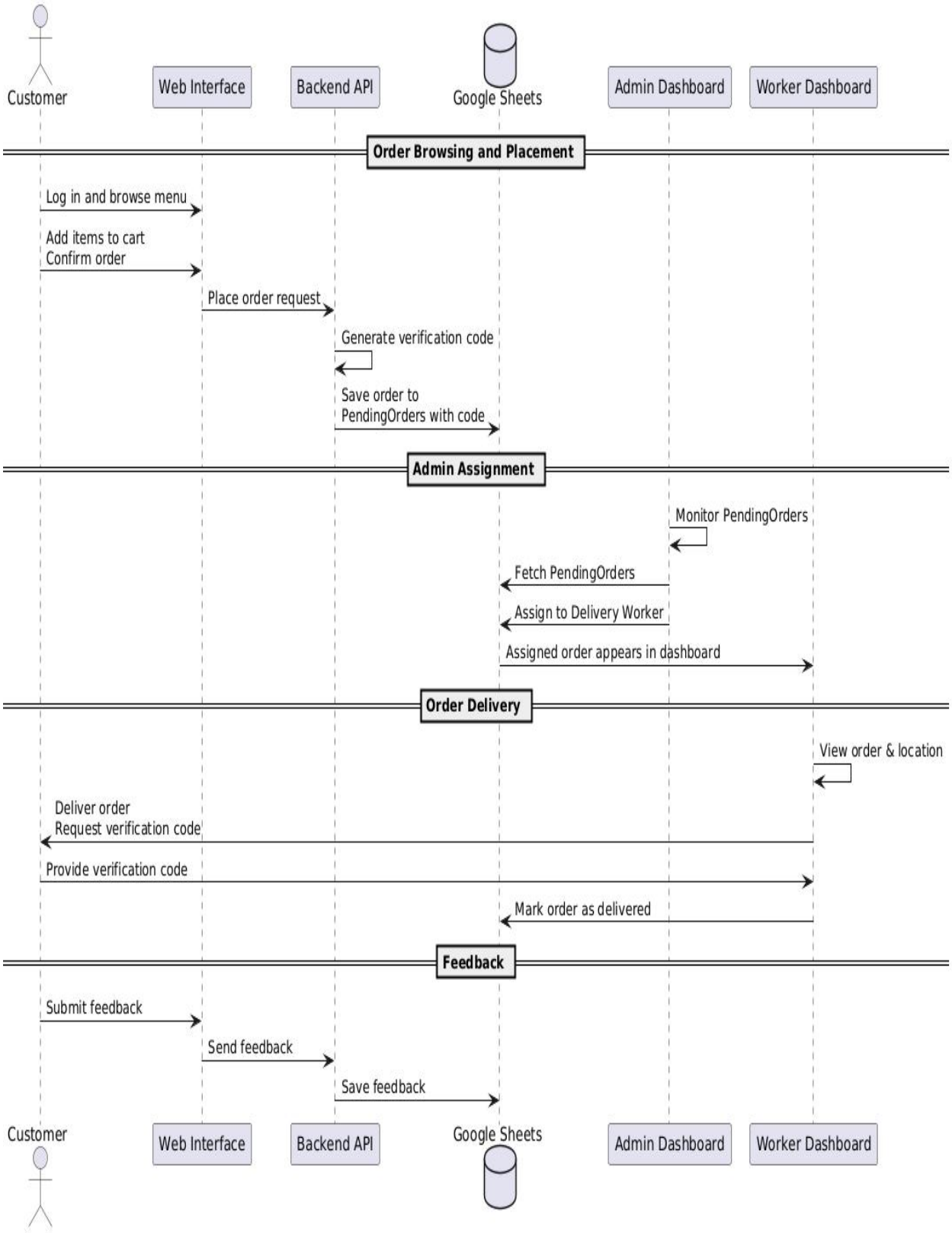
Sequence Diagram Order Confirmation:

Customer → Web Interface → Backend API → Google Sheets → Admin Dashboard → Google Sheets → Worker Dashboard → Customer → Google Sheets

6.4.1 Activity Diagram - Order Lifecycle



6.4.2 Sequence Diagram - Order Confirmation Process

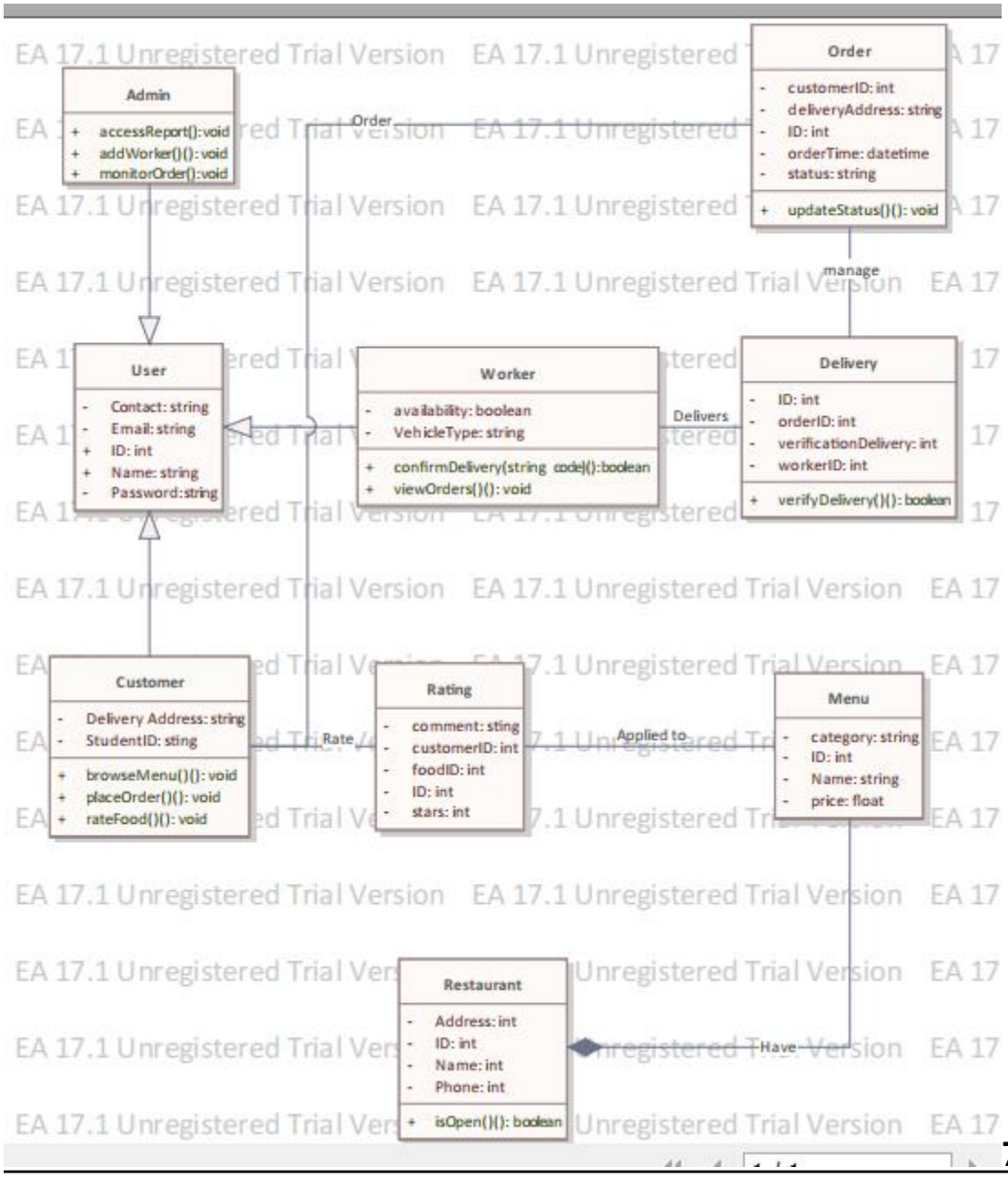


6.5 Static Modeling

Class Diagram:

- User (user_id, username, password, role)
- Order (order_id, customer_id, item_list, location, status, timestamp)
- MenuItem (item_id, name, price, vendor_id)
- Delivery (delivery_id, order_id, worker_id, verification_code, delivery_time)
- Restaurant(ID, Address, Name, Phone)

6.5.1 Class Diagram



7. System Architecture and Design

- Modular monolithic architecture with Flask backend and Google Sheets data layer.

- Uses HTML, CSS, JS, Bootstrap, and jQuery on the frontend.

- Authentication with Flask-Login and bcrypt.

- Version control with GitHub, CI/CD with GitHub Actions.

7.1 Component Design

- Auth Module

- Order Module

- Delivery Module

- Admin Panel

- Vendor Module

7.2 Database Design

- Users Sheet

- Menu Sheet

- PendingOrders Sheet

- DeliveredOrders Sheet

7.3 Interface Design

- Customer: Menu view, order history

- Worker: Assigned orders, delivery confirmation

- Admin: Order dashboard, worker stats

- Vendor: Google Sheet menu updates

8. Implementation Details

The Food Delivery System was implemented as a modular monolithic Flask application, with clear separation of concerns across authentication, order management, delivery workflows, and administrative control. Key implementation highlights include:

- **Session Management**: Flask-Login was used for session handling with user roles defined to direct logic (admin, vendor, delivery, customer).
- **Data Access Layer**: All interactions with Google Sheets are abstracted via wrapper functions using the Google Sheets API. Read and write operations are batched and queued where needed.
- **Frontend Integration**: Asynchronous requests powered by jQuery ensure dynamic page updates, particularly for menu loading, order status, and delivery confirmation.

- **Order Assignment**: Admins use a dashboard with filtered views of pending orders. Assignments update both the PendingOrders sheet and notify delivery workers through UI changes.
- **Verification and Feedback**: A unique 6-digit code is generated on order placement, used for delivery confirmation. Post-delivery feedback is saved in a separate sheet and viewable by admins.

9. Testing Strategy

The testing strategy focused on ensuring functional accuracy, security, and performance across devices and user roles.

- **Unit Testing**: Python unittest was used to validate core functions like order creation, delivery confirmation, and authentication.
- **Integration Testing**: End-to-end testing simulated a full order flow from customer to delivery using mocked Sheets API responses.
- **Cross-Browser Testing**: Manual testing was done across Chrome, Firefox, Safari, and Edge to validate consistent UI behavior.
- **Security Testing**: Forms were tested against CSRF vulnerabilities, password handling was validated via bcrypt hashing.
- **User Testing**: Real student and vendor users participated in pilot testing, providing feedback on UI/UX, load times, and workflow clarity.

10. Deployment Plan

The system was deployed on a university-hosted Linux server with HTTPS enabled via Let's Encrypt. Key deployment steps included:

- **Environment Setup**: Python virtual environment configured with dependencies via requirements.txt.
- **Secure Access**: Nginx used as a reverse proxy with SSL termination; only Flask app port exposed internally.
- **Sheet Credentials**: Google API credentials stored in environment variables and loaded at runtime for secure access.
- **Continuous Deployment**: GitHub Actions automates deployment on push to the main branch, including test runs and linter checks.

11. Maintenance and Support

A simple yet effective maintenance plan ensures the system remains functional and adaptable:

- **Bug Tracking**: GitHub Issues is used for tracking bugs, feature requests, and improvement tasks.
- **Sheet Health Checks**: A scheduled script pings and verifies access to key Sheets (Menu, Orders) and alerts the admin if errors are found.
- **User Support**: Basic user help resources are included on the platform; admin can reset passwords and reassign delivery roles.
- **System Updates**: Code updates are versioned and deployed via CI/CD pipelines. A staging server is used for pre-production testing.

12. Conclusion

The Campus Food Delivery System represents a tailored, scalable, and cost-effective solution to the unique logistical challenges of food delivery within a university environment. By integrating familiar tools like Google Sheets with robust web technologies, the platform delivers real-time efficiency, improved coordination, and ease of use for all stakeholders. The modular architecture ensures the system can evolve over time, supporting additional features, analytics, or even migration to a traditional database when needed.