Comprehensive Artifact Update Summary

As the E-Commerce project progressed through each development milestone, key artifacts were systematically refined to reflect both technical advancements and feedback from previous evaluations. The **Project Proposal**, **Requirements Specification**, and **System Architecture Plan** have all undergone thorough revisions to ensure they align with the final implementation.

These updated documents now encapsulate the project's **matured scope, validated user stories, and finalized technical structure**, integrating the latest enhancements introduced during the final sprint. Key improvements include:

- **Robust authentication and registration logic** with front-end validation and secure password hashing
- **Streamlined checkout workflow** integrating input validation, order review, and payment placeholder logic
- **Fully connected backend APIs** that support CRUD operations for users, products, and orders through secure role-based access control

Each artifact now provides a **complete and accurate reflection** of the system as deployed, supporting traceability from initial design through to functional deployment. The refined architecture diagrams illustrate current database relationships, security layers, and controller-service-repository flows, offering a clear high-level and low-level view of the project’s infrastructure.

This documentation not only captures the project's current capabilities but also ensures it is **well-positioned for future scalability, testing, and professional deployment.**

Document History

Version 1.0 – Initial Foundation

Title: *Laying the Groundwork for a Secure E-Commerce Web Application*

Overview
The first release of the project established the **vision and strategic foundation** of a dynamic E-Commerce platform designed to simulate an online shopping experience. This early version focused on conceptualizing system goals, establishing the technology stack, and drafting user experience flows.

Key Highlights

- **Project Proposal & Objective Statement**: Outlined a focused goal to build a secure, intuitive shopping portal allowing users to register, log in, browse a product catalog, and securely log out.

- **Preliminary Designs**: Developed initial **wireframes**, **flowcharts**, and **feature maps** to visualize user pathways for registration, login, and product browsing.
- **Core Technologies Selected**:
    - Frontend: *HTML*, *CSS*, *JavaScript*
    - Backend: *Node.js*, *Express.js*
    - Database: *MongoDB*
- **Defined Functional Specs**: Early feature list included secure user registration with hashed credentials, JWT-based login session tracking, and a product list view accessible without login.

Outcome
This version served as the project's blueprint, anchoring future decisions and providing a clear path for aligning functional requirements with instructor feedback.

🔁 Version 2.0 – Evolving the Design
Title: *Refinement and Standards Implementation*

Overview
Version 2.0 focused on transforming early concepts into scalable designs while **introducing formal software engineering practices** to the codebase. This version marked the evolution from planning to structured architecture.

Key Highlights

- **Wireframe and Flowchart Enhancements**: Added detailed decision points and edge case logic to the user journey (e.g., error handling for invalid credentials, session timeout logic).
- **REST API Design Beginnings**: Introduced modular API route planning and documentation using REST principles (GET/POST/PUT/DELETE).
- **Separation of Concerns**: Laid groundwork for layered architecture—distinct modules for user services, authentication middleware, and database schemas.
- **Security Policies Introduced**: Added basic security headers, CORS restrictions, and login rate limiting concepts to system planning.

Outcome
This release demonstrated the transition from idea to implementation by addressing design scalability and compliance with software development best practices, as reflected in the revised project architecture and aligned with Status Report Page 2: *Standards Followed*.

🛠️ Version 3.0 – Finalized Architecture & Readiness

Title: *Deployment-Ready Model with Full Functional Integration*

Overview
Version 3.0 is the capstone release—finalizing designs and specifications to reflect the **fully implemented system**, following instructor feedback and end-to-end integration testing.

Key Highlights

- **Architecture Plan Finalized**: Includes updated diagrams showcasing the MVC structure, user authentication flows, and data pipelines across services.
- **Checkout and Order System**: Introduced a working checkout cart, order summary review, and simulated payment module.
- **Security Enhancements**: Final implementation includes **bcrypt password hashing**, **JWT-based token validation**, and secured API endpoints with role-based access.
- **Backend Integration**: Fully connected database operations for managing user data, product listings, and order records using Mongoose.

Outcome
This final version consolidated all elements into a functional, testable product. The updated project documentation reflects not only what was built, but also the **evolution of thought, design, and execution** throughout the course.

🧱 Version 4.0 – User Interface Foundation

Title: *Establishing the Visual Identity and Navigation Flow*

Overview
Version 4.0 marked the transition from system planning into actual page development, focusing on building the foundational frontend components of the E-Commerce system. While minimal dynamic content existed at this stage, the **static structure and user journey** began to take shape visually.

Key Highlights

- **Frontend Pages Implemented**:
    - Home.html: Landing page with clean navigation layout
    - About.html: Static informational page about the platform's mission
    - Product.html: Empty placeholder representing future dynamic product listing
- **Navigation Setup**: Introduced basic anchor links between Home, About, Login, and Register pages to simulate a functional UI experience.
- **Focus Period**: Post-May, the development focus shifted to refining visual consistency, page layout, and user flow logic while deferring backend logic and product interaction.

Outcome
This phase served to provide stakeholders and instructors a **tangible prototype** for interface critique and usability input. While coding activity was limited, this phase was critical for mapping out the user experience blueprint.

🔩 Version 5.0 – Backend Integration Begins

Title: *First Steps Toward User Authentication and Data Persistence*

Overview
Version 5.0 introduced the project’s backend layer with a focus on **user registration and login functionality**, forming the backbone of secure access control. MongoDB was deployed via Atlas, and the basic API endpoints were developed and tested.

Key Highlights

- **Database Setup**:
    - MongoDB Atlas connection established

- “Users” collection created to store user credentials

- **Backend Development**:
  - Node.js + Express.js API routes created:
    - /register: Registers new users with password hashing
    - /login: Verifies user credentials and returns JWT token

- **JWT Implementation**: Tokens were used to create sessionless authentication, allowing the frontend to identify authenticated users securely.

- **Frontend Status**: Product page remained static, using hardcoded content as a placeholder for upcoming database-driven menus.

- **Security Measures**: Access to protected pages (e.g., checkout, order history) now required JWT token validation.

Outcome
This version was a major technical milestone, introducing **database-backed authentication** and laying the foundation for more secure, scalable user management and API interaction.

🛒 Version 6.0 – Dynamic Product Integration
Title: *Transition from Static to Dynamic: Product Database Live*

Overview
Version 6.0 signified a leap forward by replacing static product listings with **live data fetched from a backend collection**. The platform moved closer to functioning as a true E-Commerce system with dynamic content generation.

Key Highlights

- **Product Collection Added**:
  - MongoDB “Products” collection implemented
  - Includes fields such as name, description, image URL, category, and price

- **Admin Tools (WIP)**: Internal scripts or future dashboard to support adding/updating products in the database

- **Frontend Integration (Planned)**: Preliminary hooks were created for retrieving product data via /products API endpoint (to be connected in later versions)

Outcome
This milestone allowed the E-Commerce system to serve dynamic product content, paving the way for shopping cart functionality, search, and

inventory management in upcoming versions.

📦 Version 7.0 – Shopping Cart and Order Management Integration

Title: *Turning Transactions into Action: Full Cart and Order Backend Implementation*

Overview
Version 7.0 marked a pivotal expansion of the E-Commerce platform, transforming it from a display-only product interface into a fully functional transactional system. This update introduced the logic for cart persistence, order processing, and secure checkout, effectively bridging the frontend with backend data operations.

Key Highlights

- **Orders Collection**:
    - MongoDB “Orders” collection added to capture all customer purchases, with structure including customer ID, products ordered, timestamp, and shipping metadata.
    - A new backend model (Order.js) was created, encapsulating data validation and order schema management.
- **API Enhancements**:
    - POST /orders: Allows authenticated users to place orders securely and persistently.
    - POST /cart/add, PUT /cart/update, DELETE /cart/remove: Enabled backend shopping cart management and dynamic cart updates per user session.
- **Cart Model Added**:
    - CartItem.js introduced to manage individual product quantities and variations per user.
- **Checkout Flow Expansion**:
    - Introduced confirmation.html and a basic shipping information form, simulating a real-world checkout experience and establishing foundations for shipping modules.

Outcome
The platform evolved into a **true e-commerce engine**, where users could browse, add to cart, and complete a purchase. This version laid the groundwork for future enhancements in validation, shipping logic, and user-specific cart retention.

☑ Version 8.0 – Final Testing, UX Polishing & Feature Completion

Title: *From Prototype to Production: Final Touches and Full Testing*

Overview
Version 8.0 culminated the project with a thorough **user experience overhaul**, **security hardening**, and **end-to-end testing**. This release ensured that every component—from registration to checkout—functioned cohesively with visual feedback, responsive design, and secure API communication.

Key Highlights

- **Enhanced Login/Registration Validation**:
  - Enforced minimum 5-character input for usernames and passwords.
  - Real-time input feedback using color-coded validation cues.
  - Error messages displayed dynamically, reducing form submission confusion.
- **Checkout UI Refinements**:
  - Implemented a city dropdown with 20 major U.S. cities to streamline user input.
  - Phone number validation included smart formatting: (XXX) XXX-XXXX as user types.
  - Form validation and feedback matched modern UX standards, improving form accuracy.
- **Navigation and Port Standardization**:
  - All frontend API calls aligned to http://localhost:5000, ensuring consistent dev environment behavior.
- **Shopping Cart Improvements**:
  - Fully persistent carts tied to individual users.
  - Error handling through toast notifications, ensuring real-time updates when cart actions failed or succeeded.
- **Mobile-Responsive & Robust Design**:
  - Polished all frontend pages for mobile compatibility.
  - Enhanced error handling across all modules, ensuring graceful degradation and feedback.
- **Final System Testing**:

- Conducted complete walkthrough testing from registration to order placement.
- Verified all success paths and failure modes using dummy data to simulate user behavior.

Outcome
With Version 8.0, the system matured into a **polished, user-friendly, and fully functional e-commerce platform**. It successfully integrates modern design practices, intuitive user flows, and solid backend connectivity, passing all critical project milestones and fulfilling key academic and functional requirements.

Final Project Proposal

Project Title: Secure E-Commerce Web Application

Overview

This project centers on the development of a secure, full-featured **E-Commerce Web Application** designed to simulate the functionality of a modern online retail platform. The application empowers users to **register, log in securely, browse available products, manage a personalized shopping cart, and complete a full checkout process with order confirmation**.

With a focus on **secure backend architecture**, intuitive design, and end-to-end functionality, this system is built using the **MERN stack**—MongoDB, Express.js, React (substituted here with HTML/CSS/JS for frontend simplicity), and Node.js. From a technical and user experience perspective, the platform ensures **responsive design, protected user data, and a streamlined transaction flow**, making it ideal for demonstrating secure software engineering principles in a real-world context.

The final build also reflects the successful integration of various academic deliverables, including feedback-based refinements, milestone testing, and adherence to traceability via functional requirements.

Objectives

The primary objectives of the E-Commerce Capstone Project are to:

- ☑ **Deliver a Seamless Shopping Experience**
  Provide end-to-end functionality for a complete e-commerce journey—user registration, secure login, product display, cart management, checkout, and order confirmation.

- 🔒 **Implement Enterprise-Level Security Standards**
  Incorporate **JWT (JSON Web Token)** authentication and **bcrypt** hashing to secure user sessions and protect sensitive data such as passwords and order information.

- **Design a Responsive & User-Centered Interface**
  Utilize modern front-end practices to create a mobile-friendly, responsive design with **real-time input validation**, UI error feedback, and accessible navigation.

- **Ensure Persistent, Scalable Data Storage**
  Use **MongoDB Atlas** to persist user accounts, shopping carts, product inventory, and order history in a scalable, secure database.

- **Establish Full Backend Integration**
  Ensure that frontend forms and buttons are connected via RESTful APIs to backend logic written in Node.js/Express, completing the cycle of interaction between UI, business logic, and data storage.

- **Achieve Complete Feature Coverage**
  Attain 100% implementation of all core functional and nonfunctional requirements, ensuring that he final system is deployable, testable, and reflective of real-world commercial applications.

- ## 📌 Scope of the Project

  - The scope of this capstone project entails the full-stack development of a secure, responsive, and user-friendly **web-based e-commerce platform**. The system simulates a modern online retail environment, offering essential functionality for browsing products, managing shopping carts, and processing simulated checkouts.

  - ### ☑ In-Scope Features

  - The project includes the following critical components and functionalities:

  - ***🌐 Frontend Design***

- Developed using **HTML, CSS, and JavaScript** to support responsive and interactive UI across multiple devices.
- Seamless navigation with clearly defined routes for Home, About, Login, Register, Products, Cart, and Checkout pages.

- ***⚙ Backend Architecture***

- Implemented with **Node.js and Express.js** to serve as the RESTful API layer.
- **MongoDB Atlas** used for secure, cloud-based data storage of users, products, cart items, and orders.
- Modular architecture leveraging **MVC principles** and clean separation of concerns.

- ***🚀 Key Functional Features***

- **User Registration & Login:**
  - Minimum 5-character inputs for usernames and passwords.
  - Real-time validation feedback with color indicators and error messages.
  - Secured with **bcrypt** for password hashing and **JWT** for session authentication.
- **Dynamic Product Catalog:**

- Loaded directly from the MongoDB database.
- Allows users to add products to the cart dynamically.

- **Persistent Shopping Cart:**
  - Cart items stored per user session.
  - Supports quantity control, live updates, and item deletion.
- **Streamlined Checkout Process:**
  - Form validation for **city dropdown** (20 U.S. cities) and phone numbers formatted as (XXX) XXX-XXXX.
  - Order confirmation page displays transaction summary.
- **Access Control:**
  - JWT middleware ensures only authenticated users can access cart and checkout pages.
  - Unauthorized users are redirected with clear messaging.
- **Error Handling & UX Enhancements:**
  - Toast notifications, UI feedback, and friendly error messages to guide users.
  - ## ⃠ Out of Scope
  - While the current release includes a robust e-commerce core, several features have been **excluded from scope** to maintain focus and ensure a successful implementation within the course timeframe:

- Advanced product filtering or category-specific search
- Payment gateway integration (the checkout is simulated only)
- Full order history for returning users
- Inventory management or stock tracking
- Email verification or password reset functionality
-

🚧 Project Milestones & Development Timeline

The project has evolved through structured milestone phases, each reflecting progressive functionality, bug fixes, and design upgrades.

✔ Fully Implemented Features

The following major tasks and enhancements have been successfully completed:

- **Frontend-Backend Integration** using API endpoints for login, registration, product loading, cart updates, and order submission.
- **Enhanced User Authentication:** Real-time feedback, validation, and secure session management.
- **Shopping Cart Persistence:** Data retained across sessions and tailored to individual user accounts.
- **Streamlined Checkout:** Includes validation logic, dropdown selections, and simulated confirmation screens.
- **Order Management:** Integration of Orders and CartItem models in MongoDB, with full backend persistence.
- **Final Touches:** End-to-end testing, responsive design tuning, toast-based alerts, and complete error handling.

🚀 Potential Future Enhancements

While not implemented in the current version, these features have been identified as valuable extensions for future development:

- **Order History Module (FR-010):**
  Allow users to view past orders via their account dashboard.
- **Advanced Product Filtering:**
  Introduce category tags, price filters, or search options.
- **Enhanced Authentication Measures:**
  Support for more secure password policies and potential **email validation** features.
- **Expanded Product Diversity:**
  Additional product categories and UI enhancements for showcasing item types (e.g., digital vs. physical goods).

🔍 Final Traceability Overview

- The final traceability of the Secure E-Commerce Web Application ensures comprehensive alignment between defined functional requirements and their corresponding implementations. Each requirement was meticulously developed, tested, and validated to ensure it meets user expectations, technical constraints, and security standards. This traceability section highlights how each feature was integrated into the final system, showcasing the system's progression from conceptual planning to deployment-ready functionality.

- **FR-001: User Registration**

Purpose:
To allow users to create secure accounts while enforcing input validation and preventing duplicate entries.

Implementation Summary:

- A dynamic registration form was designed to capture usernames (5–30 characters) and passwords (5–100 characters) with real-time input validation.
- Only alphanumeric characters and underscores were permitted for usernames, promoting consistency and security.
- Feedback was color-coded: red for invalid inputs, green for valid inputs.
- Passwords were encrypted using **bcrypt** prior to being stored in the **MongoDB** database.
- The system checks for duplicate usernames and presents meaningful error messages to the user.

Outcome:
A fully secure and user-friendly registration module that prevents weak credentials and duplicate accounts, improving onboarding security.

- **FR-002: User Authentication**

Purpose:
To validate user credentials and securely initiate a session.

Implementation Summary:

- Login forms enforced a minimum of 5 characters for both usernames and passwords.
- Incorrect or missing credentials triggered real-time alerts, eliminating guesswork for users.
- Upon successful authentication, a **JWT (JSON Web Token)** was generated and stored in the client's localStorage.
- Users were then automatically redirected to shop.html, signaling successful login.

Outcome:
A robust login system that guarantees only valid users gain access, with strong session protection and minimal user friction.

- **FR-003: User Session Control and Lifecycle Handling**

Purpose:
To restrict access to sensitive pages and actions based on authentication status.

Implementation Summary:

- Pages such as shop.html, checkout.html, and confirmation.html implemented middleware logic to detect valid JWT tokens.
- If absent or expired, users were redirected to login.html with informative prompts.
- Backend endpoints (e.g., for cart and orders) were also secured with middleware, ensuring users could not access resources without valid credentials.

Outcome:
Effective enforcement of session control, ensuring system integrity and security across protected user pathways.

- **FR-004:** Logout Functionality

Purpose:
To enable users to explicitly end their session and clear sensitive data.

Implementation Summary:

- A logout trigger on shop.html removed the JWT token from local storage.
- Users were provided a success message and redirected to login.html for re-authentication.
- This process ensured no session residue remained, even on shared devices.

Outcome:
A clean and user-aware logout feature that enhances account safety and trust.

- **FR-005:** Real-Time Cart Overview Interface

Purpose:
To present a dynamic, visually appealing product catalog pulled from the backend.

Implementation Summary:

- Product details were retrieved via the GET /products API from the **Products** collection in **MongoDB**.
- Each product was presented in a grid layout with name, price, category, and description.

- CSS-based hover effects added interactive polish and usability.

Outcome:
A dynamic and scalable product presentation layer that adapts as product data evolves.

- **FR-006:** Cart Operations and Order Preparation System

Purpose:
To provide users with full control over their shopping experience.

Implementation Summary:

- Items could be added via POST /cart/add, with live updates reflected on-screen.
- Cart item quantities could be adjusted (PUT /cart/update) and removed (DELETE /cart/remove) with immediate feedback.
- Each user's cart was stored persistently in **MongoDB**, tied to their JWT-authenticated identity.

Outcome:
An interactive and persistent shopping cart experience that ensures usability across sessions and devices.

- **FR-007: Real-Time Cart Overview Interface**

Purpose:
To allow users to visually access and review their selected items before checkout.

Implementation Summary:

- The cart sidebar dynamically loads via the GET /cart API, displaying all relevant cart data: item names, quantities, individual prices, and a running subtotal.
- Designed for responsiveness, the sidebar adjusts to mobile and desktop screens while gracefully handling empty-cart states with friendly prompts.
- Real-time updates ensure that cart modifications are instantly reflected without requiring page reloads.

Outcome:
An intuitive and efficient shopping cart preview system, essential for informed purchasing decisions.

- **FR-008: Interactive Cart Update and Item Adjustment**

Purpose:
To enable users to make changes to their cart contents with real-time visual and backend synchronization.

Implementation Summary:

- Users can increase or decrease item quantities or remove items entirely.
- Updates are handled via the PUT /cart/update and DELETE /cart/remove APIs.
- If an item’s quantity is set to zero, it is automatically removed from the cart.
- Each change recalculates the subtotal and reflects changes immediately in the interface.

Outcome:
A highly interactive and user-friendly cart management system that empowers users to curate their orders in real time.

- **FR-010:** Transaction Records and Purchase History Tracking (Planned Feature)

Purpose:
To provide users the ability to view past purchase records and enhance transparency.

Future Implementation Plan:

- Users will be able to access a dedicated order history page displaying:
    - Order ID
    - Order date and timestamp
    - Itemized breakdown
    - Total order amount
- Designed for future milestone beyond current project scope (marked as FR-010 in planning).

Planned Outcome:
A value-added feature to improve account management and customer trust by enabling post-order tracking and review.

- **FR-011: Intuitive Interface Navigation and Page Routing**

Purpose:
To offer a consistent and seamless user experience across all pages in the application.

Implementation Summary:

- A persistent navigation bar is implemented across all core pages: Home.html, Login.html, Register.html, Shop.html, Checkout.html, and Confirmation.html.
- Navigation links dynamically adjust based on authentication status (e.g., hiding login/register when authenticated).
- Page transitions maintain state using the JWT stored in localStorage, ensuring continuity and session awareness.

Outcome:
A unified navigation experience that simplifies usability and reinforces application consistency.

- **FR-012: Database Integration and Storage Management**

Purpose:

To ensure secure and reliable storage of all critical user and transactional data.

Implementation Summary:

- All core datasets—**user credentials**, **cart contents**, and **orders**—are stored in **MongoDB Atlas**, the cloud-hosted NoSQL database.
- Cart data persists across browser sessions and reloads, tied to the authenticated user's ID.
- Orders are stored indefinitely, ensuring reliable historical data for future enhancements.

Outcome:
A persistent, secure backend architecture that supports long-term data reliability and scalability.

Non-Functional Requirements (Updated and Detailed)

The success of this e-commerce web application depends not only on its core features but also on how effectively it performs, scales, secures, and interacts with users. The following non-functional requirements define key operational qualities that support long-term reliability, usability, and maintainability of the system.

Security

Security is paramount for any system handling user credentials, shopping activity, and order transactions. The application incorporates multiple layers of protection to ensure data integrity and confidentiality:

- **Token-Based Access Control:**
  The system utilizes **JSON Web Tokens (JWT)** to manage user authentication and secure access to protected routes. JWTs are stored securely in the browser's localStorage and validated on each request.
- **Encrypted Credential Storage:**
  Passwords are hashed using **bcrypt**, a widely accepted cryptographic algorithm that incorporates salting and adaptive complexity, making brute-force attacks highly ineffective.
- **Configuration Isolation:**
  Sensitive configuration values (e.g., database credentials, API keys) are stored in a **.env environment file**, which ensures separation from the source code and reduces the risk of exposure in version control systems like Git.

Usability

User experience is central to the success of the platform. The interface is designed to be modern, responsive, and intuitive, ensuring accessibility across devices and technical skill levels:

- **Responsive User Interface (UI):**
  The front end is built to adapt seamlessly to various screen sizes including mobile phones, tablets, and desktops, following modern responsive web design principles.

- **Real-Time Form Validation:**
  All input forms, including login, registration, and checkout, feature **live validation feedback**. Inputs are dynamically styled (e.g., red for errors, green for valid entries) and submit buttons are disabled until all required fields are correctly filled.

- **Interactive Feedback Mechanisms:**
  Toast notifications provide contextual messages during user actions such as adding items to the cart, logging in, or submitting orders. These improve clarity, engagement, and trust.

Performance

Efficiency in application performance enhances user satisfaction and ensures smooth operations even as user traffic scales.

- **Optimized Database Operations:**
  MongoDB queries are streamlined using indexed fields and efficient data structures. This ensures fast read/write access for product display, user authentication, and order processing.

- **Loading Indicators & UI Feedback:**
  While API calls are in progress (e.g., loading products, placing orders), the interface displays loading animations or placeholders to maintain engagement and inform the user of background activity.

- **Minimal Latency for Common Tasks:**
  Core actions like logging in, adding items to cart, and submitting orders are optimized to return responses quickly, reducing perceived lag and bounce rates.

Software Engineering Standards

Adherence to modern development conventions ensures the application is scalable, testable, and maintainable by other developers or future

enhancements.

- **RESTful Architecture:**
  All backend APIs follow RESTful principles with logical resource-based endpoints (/login, /register, /products, /cart, /orders). This makes the API easy to extend and integrate with other frontends or services.

- **Modular Code Structure:**
  The backend and frontend are separated based on **Separation of Concerns (SoC)** principles. Each module (authentication, product service, cart service, etc.) has a clear responsibility, improving testability and debugging.

- **Consistent Design Patterns:**
  The frontend uses standardized component patterns, while the backend follows a structured MVC (Model-View-Controller) approach. This promotes clean code and reduces technical debt over time.

- **Accessibility & Compliance:**
  Basic accessibility standards (ARIA labels, tab navigation, form focus) are observed to support inclusive usage.

⚙️ System Architecture Overview

The architecture of this e-commerce platform is structured for scalability, modularity, and security. Each component of the system works in tandem to provide a seamless shopping experience for the user, while maintaining data integrity and system efficiency. The architecture adopts a **client-server model**, leveraging RESTful APIs to facilitate communication between the frontend and backend layers.

🔧 4.1 Technology Stack Breakdown

Frontend Technologies

- **HTML5, CSS3, JavaScript (Vanilla)**
  These foundational web technologies are used to build a responsive and interactive user interface. All major views—including registration, login, product browsing, cart, and checkout—are rendered using clean and semantic HTML, styled with modern CSS, and brought to life with dynamic JavaScript.

- **User Experience Enhancements**
  Features such as **real-time form validation**, **toast notifications**, and **responsive layouts** ensure accessibility and usability across desktops, tablets, and mobile devices.

Development Tools

- **Live Server (VS Code Extension)**
  Enables real-time preview of HTML/CSS/JS changes during development for rapid iteration and debugging.

- **Thunder Client**
  A lightweight REST API client used to test backend endpoints such as login, product retrieval, and order creation directly from within the Visual Studio Code environment.

Backend Technologies

- **Node.js**
  Provides a fast, non-blocking runtime environment for the server-side application. Built with scalability and performance in mind.

- **Express.js**
  A minimal and flexible Node.js framework that simplifies API creation through routing, middleware support, and modular development.

API Architecture

- **RESTful Endpoints**
  The system exposes a suite of REST APIs that conform to standard HTTP methods and URIs for the following key operations:
    - POST /register – Handles user registration.
    - POST /login – Authenticates users and returns a JWT.
    - GET /products – Retrieves product catalog from the database.
    - POST /cart/add – Adds item to the user's cart.
    - PUT /cart/update – Modifies quantity of existing cart items.
    - DELETE /cart/remove – Removes items from the cart.
    - POST /orders – Processes a new order and stores it in the database.

Database & Models

- **MongoDB (via MongoDB Atlas)**
  A NoSQL cloud database used to persist all application data including users, products, carts, and orders. Collections are optimized using indexes for rapid queries.
- **Defined Mongoose Models:**
    - User.js: Handles schema for registration and login.
    - Product.js: Manages product listings and attributes.
    - CartItem.js: Tracks items added to each user's cart.
    - Order.js: Records processed orders and shipping info.

Security Implementation

- **JWT (JSON Web Tokens)**
  Used for stateless authentication. Tokens are generated on login and attached to protected API requests, ensuring session integrity and access control.

- **bcrypt**
  Implements secure password hashing with salting, making stored credentials resistant to brute-force and dictionary attacks.

- **Environment Configuration**
  Sensitive credentials (e.g., database URIs, JWT secret) are stored in a .env file and never hard-coded in source files, reducing vulnerability and improving portability.

☑ Updated Project Structure

```
E-Commerce/
 ├─── Front End/
 │    ├─── Home.html              # Landing page introducing the store
 │    ├─── Login.html             # Enhanced with real-time validation, JWT login
 │    ├─── Register.html          # Integrated with username/password rules, feedback
 │    ├─── Shop.html              # Displays product catalog, add-to-cart feature
 │    ├─── Checkout.html          # Features phone validation and city dropdown
 │    └─── Confirmation.html      # Order confirmation post-purchase
 │
 ├─── Server.js                   # Entry point for Express backend, routes integration
 ├─── .env                        # Secures MongoDB URI, JWT secret, and config vars
 ├─── package.json                # Node.js dependencies and metadata
 │
 ├─── Routes/
 │    └─── auth.js                # Handles user registration, login APIs (JWT auth)
 │    └─── products.js            # API to retrieve and manage product data
 │    └─── cart.js                # APIs for cart add/update/remove
 │    └─── orders.js              # Order processing and storage endpoints
 │
 └─── Models/
      ├─── User.js                # Mongoose schema for user credentials
      ├─── Product.js             # Schema for products displayed in shop.html
```

```
├── CartItem.js          # Handles cart persistence and structure
└── Order.js             # Schema for finalized orders and shipping info
```

Key Highlights

- **Separation of Concerns**: Frontend HTML files are kept distinct from backend logic and database models.
- **Security**: JWT, bcrypt, and .env practices are clearly embedded in both architecture and codebase layout.
- **Scalability**: Modular folders and RESTful routes allow easy expansion (e.g., adding order history, product filtering).
- **User Experience**: HTML pages incorporate validation, feedback, and progressive disclosure (like dropdown cities, toasts).
- **Real-World Readiness**: Structure aligns with modern full-stack standards, suitable for portfolio or industry deployment.

```
└── Order.js (Fully implemented)
```

🏗 4.3 System Architecture Overview

The **Secure E-Commerce Web Application** follows a **four-layer architecture**, ensuring modularity, scalability, and maintainability. Each layer fulfills a distinct role in delivering a responsive, secure, and data-driven user experience.

Client Layer – User Interaction and Presentation

- The **client layer** represents the browser-based interface that users directly interact with.

- Built using **HTML5**, **CSS3**, and **JavaScript**, it enables an intuitive shopping experience.
- Pages include user registration, login, product browsing, cart management, and checkout.
- This layer emphasizes **responsive design**, ensuring seamless access across desktops, tablets, and smartphones.
- Real-time validation and visual feedback guide users through every form and transaction.

Application Layer – Logic and Control

- The **application layer** acts as the intermediary between the user interface and the database.

- Implemented using **Node.js** and **Express.js**, it handles **API routing**, **input validation**, and **core business logic**.
- It manages session handling, request authentication, and cart/order workflows.
- All API endpoints conform to **RESTful design principles**, promoting scalability and modularity.
- This layer processes requests such as product retrieval, cart updates, and order submissions, returning structured JSON responses.

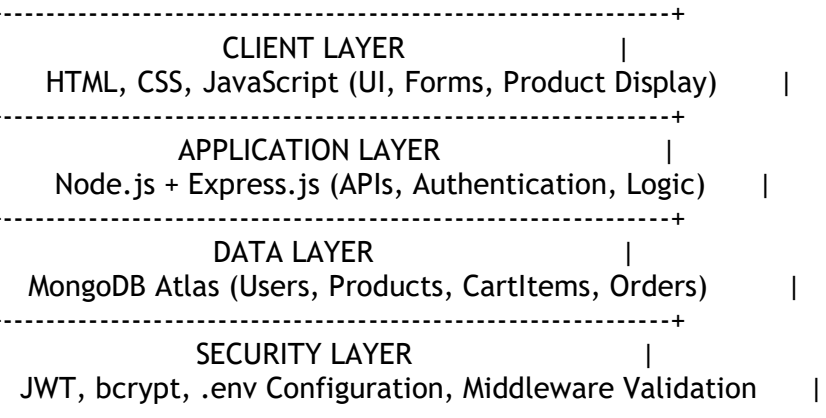
Data Layer – Storage and Retrieval

- The **data layer** ensures reliable and persistent data storage through **MongoDB Atlas**, a cloud-based NoSQL database.
- Collections include **Users**, **Products**, **CartItems**, and **Orders**.
- Data models (User.js, Product.js, CartItem.js, Order.js) define structure, relationships, and validation.
- The database supports high-speed queries, indexed lookups, and ACID-compliant transactions.
- CRUD operations (Create, Read, Update, Delete) are exposed through secure APIs, ensuring efficient data management.

Security Layer – Protection and Privacy

- Security is woven into every layer of the architecture.
- **JWT (JSON Web Tokens)** protect restricted routes and verify user sessions.
- **bcrypt** ensures passwords are hashed and salted before storage, preventing unauthorized disclosure.
- **Environment variables** in a .env file safeguard sensitive configurations, such as database URIs and JWT secrets.
- All API routes are validated through middleware, ensuring only authenticated users can access restricted operations.

Visual Summary



+--+

- This architecture ensures a clear separation of responsibilities while maintaining **security, performance, and scalability**across all components.

👥 User Stories

- The following **user stories** define the system's expected behavior from the perspective of different users, ensuring that all functional goals align with real-world usability.

🧍 Authentication & Account Management

- As a **new user**, I want to register an account easily so that I can shop and make purchases securely.
- As a **returning user**, I want to log in with my saved credentials to continue shopping where I left off.
- As a **logged-in user**, I want to log out securely to ensure my session is closed properly.
- As a **visitor**, I want to be automatically redirected to the login page if I attempt to access protected features without authentication.

🛒 Shopping & Cart Management

- As a **shopper**, I want to browse a catalog of products with images, descriptions, prices, and categories.
- As a **customer**, I want to add, update, or remove items from my cart and see the total update instantly.
- As a **customer**, I want my cart to persist between sessions so I don't lose my selections if I log out or refresh.
- As a **user**, I want real-time notifications or alerts when cart updates succeed or fail.

💳 Checkout & Order Processing

- As a **customer**, I want a simplified checkout form that validates my input before submission.
- As a **buyer**, I want to select my **city** from a dropdown and enter a phone number that formats automatically for accuracy.
- As a **customer**, I want to receive an **order confirmation** with a unique order ID and a summary of my purchase.
- As a **buyer**, I want the option to print or save my order confirmation after checkout.
  -

🧭 Navigation & User Experience

- As a **user**, I want to easily navigate between pages such as Home, Login, Shop, Checkout, and Confirmation.
- As a **customer**, I want a consistent color scheme, layout, and branding throughout the site to build trust.
- As a **shopper**, I want the website to adapt smoothly across all screen sizes and orientations.
- As a **user**, I want clear visual feedback (e.g., color highlights, animations, or toasts) when performing actions, so I know the system has responded.
    -

Data Persistence and Long-Term Storage

A vital pillar of the Secure Order E-Commerce platform is its robust **data persistence architecture**, ensuring all user interactions and transactional records are securely stored, retrievable, and consistent across sessions. From user registration to final order confirmation, the application is designed to maintain data integrity and security using industry-standard technologies and best practices.

Persistent User Accounts

All customer accounts are persistently stored in the **MongoDB Atlas** database. During registration, user credentials—especially passwords—undergo **bcrypt hashing**, offering cryptographic protection before being saved. This design ensures that even in the unlikely event of a database breach, sensitive credentials remain protected and unreadable.

Key attributes saved include:

- Username (validated for length and format)
- Bcrypt-hashed password
- Registration timestamp

The platform guarantees that each user's data is uniquely stored and accessible only via valid credentials authenticated using **JWT tokens**, enhancing session-level security and protecting data from unauthorized access.

Shopping Cart State Retention

The shopping cart, often one of the most dynamic elements in an e-commerce platform, is **persisted per user session**through the **CartItem.js model** in MongoDB. When users add, modify, or remove products from their cart:

- Each action updates the backend database in real-time via RESTful API calls.
- The cart's contents are associated with the authenticated user's ID, allowing for **cross-session continuity**—users can leave and return later without losing their selections.

This persistent cart state not only improves the user experience but also reduces cart abandonment by letting users resume their purchasing journey with ease.

Order Lifecycle and Historical Records

Each completed transaction triggers a **POST request** to the /orders endpoint, generating a new entry in the **Orders collection**. Every order is stamped with:

- A unique order ID
- Timestamp of purchase
- Associated user ID
- Line items (products, quantities, prices)
- Shipping details (validated at checkout)

Although the **Order History** feature is scoped for future implementation (FR-010), the system already retains this critical data in anticipation of expanded functionality. This design enables:

- Future dashboards for users to review previous purchases
- Administrative insights into ordering trends
- Auditing and debugging capabilities for developers and support teams

🧠 *Long-Term Value and Forward Compatibility*

The persistence strategy used in Secure Order ensures that all data—user accounts, carts, and orders—is stored in a **scalable and query-optimized structure**. As the platform evolves, this foundational work supports:

- Future enhancements like order tracking, receipt downloads, and review systems
- Analytics features for visualizing purchase patterns
- Integration with payment gateways or external CRM systems

By combining **secure storage** with **flexible schemas**, the project is positioned to grow beyond its MVP (minimum viable product) scope and adapt to future e-commerce needs with minimal refactoring.

📦 Final Deliverables Overview

The following finalized components encapsulate the full scope of development, testing, and documentation efforts for the *Secure Order E-Commerce Platform*. These deliverables collectively demonstrate the successful implementation of a functional, secure, and user-friendly web application, backed by sound software engineering principles and a complete software development lifecycle.

📖 *1. Complete Source Code Repository*

- **Frontend Stack**: The client-side interface includes all responsive HTML pages (Home, Login, Register, Shop, Checkout, Confirmation) styled with CSS and powered by JavaScript for dynamic interactions and real-time validation.
- **Backend Stack**: Built with Node.js and Express.js, the backend features a modular, RESTful API design covering user authentication, product management, shopping cart actions, and order processing.
- **Database Integration**: MongoDB (via MongoDB Atlas) is utilized for persistent data storage. Schemas include User.js, Product.js, CartItem.js, and Order.js, each reflecting the evolving needs of the application.
- **Security Features**: Integrated JWT-based session management and bcrypt password hashing safeguard all protected routes and sensitive user data.

All code is structured for clarity, scalability, and maintainability, with clearly documented comments and folder structures.

📚 *2. Comprehensive Documentation*

- **Project Proposal**: A clearly defined overview, objectives, and scope of the e-commerce system, including a rationale for the chosen technologies.
- **Requirements Specification**: A detailed breakdown of functional and non-functional requirements, supported by traceability to ensure each feature aligns with business needs.
- **System Architecture Plan**: Visual and textual descriptions of the application's multi-layer architecture, including client-side logic, server routes, data models, and security layers.
- **Change Log & Version History**: Chronological record of milestone releases (v1.0–v8.0), each version reflecting incremental progress aligned with status reports and instructor feedback.

🧪 *3. Testing Reports*

- **API Testing**: All API endpoints (authentication, cart, product, orders) have been validated using **Thunder Client**, with test cases covering success and failure paths, input validation, and response codes.
- **Usability Testing**: Frontend interfaces have been tested for:

- Responsive design across mobile, tablet, and desktop views
- Real-time feedback (color-coded input fields, dynamic buttons)
- Cart and checkout behavior with edge case handling

- **Security Checks**: Manual validation of protected routes, JWT expiry handling, and user-specific data access has been performed to ensure strong access control mechanisms.

🧭 *4. Deployment Instructions*

- **Local Setup Guide**:
  - Clone the GitHub repository
  - Run npm install to install backend dependencies
  - Configure the .env file with MongoDB URI and JWT secret
  - Use **Live Server** to launch the frontend
  - Run node server.js or nodemon to start the Express backend
- **System Requirements**:
  - Node.js v18+
  - MongoDB Atlas account
  - Modern browser (Chrome, Edge, or Firefox)

A README file provides step-by-step walkthroughs for launching the app, testing endpoints, and interacting with the UI.

Conclusion

The E-Commerce Web Application project represents a complete, secure, and responsive online shopping experience that successfully integrates all core elements of a modern retail platform. From initial registration to final order confirmation, the application provides a seamless and intuitive workflow designed to simulate real-world e-commerce functionality while prioritizing usability, security, and performance.

This project was developed using a robust technology stack—**HTML, CSS, JavaScript (frontend)** and **Node.js, Express.js, MongoDB (backend)**—with industry-standard practices such as **JWT-based authentication**, **bcrypt password hashing**, and structured **RESTful API architecture**. Every feature was purposefully designed and tested to ensure data integrity, session management, and dynamic content handling, resulting in a scalable and maintainable solution.

As of November 1**, 2025**, the platform has reached full implementation status. Users can register securely, browse a live product catalog, manage a persistent shopping cart, and complete a validated checkout process. Key enhancements such as real-time form validation, responsive design for all devices, and toast-based feedback ensure an elevated user experience.

Looking forward, the project's architecture is flexible enough to support future enhancements like **order history pages**, **dynamic product filtering**, **email verification**, and **advanced account management**—paving the way for continued innovation and greater feature depth.

This capstone project not only meets the academic objectives of CST-452 but also provides a solid foundation for real-world deployment, professional portfolio inclusion, and potential business expansion.