FINAL REPORT

BY: The FantasticFive H13_W

INDEX

Business Case

The Problem

SMEs (Small and Medium sized Enterprises) require an efficient way to generate UBL-compliant order documents. Manually generating these documents is time-consuming, error-prone, difficult to scale and increases friction with other UBL dependent enterprises, an internationally recognised XML standard. Without automation, SMEs struggle with handling high order volumes efficiently and face administrative overhead that takes time away from growth and customer engagement. The foremost issues plaguing SMEs surrounding order creation are as follows:

- **Time-consuming manual processes**: Repetitive data entry consumes valuable time that could be directed toward growth and customer engagement.
- **Increased risk of errors**: Manual input leads to mistakes in order details, quantities, and pricing, damaging customer trust.
- **Lack of standardization**: Without uniform document formats like UBL (Universal Business Language), integrating with vendors, customers, and internal systems becomes complex
- **Scalability concerns**: As businesses grow, manual order handling becomes unsustainable, creating operational bottlenecks.
- **Employee burnout**: Statistics show 90% of employees feel burdened by repetitive tasks, 65% report work overload, and 81% risk burnout without automation.
- **Integration challenges**: Businesses struggle to connect diverse systems, leading to information silos and workflow disruptions.

The financial impact of these challenges cannot be overlooked. Studies indicate that process inefficiency can cost businesses up to 20-30% in unnecessary operational expenses. Furthermore, the opportunity cost of diverting skilled employees toward repetitive administrative tasks rather than strategic initiatives creates a hidden drag on business potential and market competitiveness.

Solution Requirements

. The multifaceted challenges of SME necessitate a solution require the following features:

- **Standardise order / invoice formats:** A uniform, industry-standard format like UBL ensures smooth integration between systems, reducing the need for manual intervention and eliminating compatibility issues.
- **Automate order creation:** Automation is key to minimising human errors, speeding up processing, and enabling real-time data exchange.
- **Ensure scalability and flexibility:** The solution should be able to handle varying volumes of orders and scale as the business grows.
- **Enable seamless integration:** Businesses need a solution that integrates easily into existing systems without complex overhauls or literacy in complex software.
- **Maintain compliance and accuracy:** Orders must be generated correctly and in line with industry standards to meet legal and operational requirements.

Our Solution: OrderNova

OrderNOVA is a comprehensive, cloud-based order management platform specifically designed for SMEs that addresses the operational challenges of modern business document management. By leveraging cutting-edge technology and adhering to international standards, OrderNOVA transforms document processing from a resource-intensive burden into a streamlined, automated advantage that drives business efficiency and growth.

Table. OrderNova Feature Table

Feature	Category	Description	Business Benefit
UBL-compliant documents	Functional	Ensures all documents meet international standards for business document exchange	• Eliminates compatibility issues with trading partners • Streamlines business-to-business transactions • Reduces document formatting errors • Ensures compliance with global standards
Order Creation and Management	Functional	Generate, edit, delete and retrieve order documents	• Enhanced operational efficiency • Facilitates seamless B2B operations • Allows business to focus on growth rather than admin
Invoice Creation and Management	Functional	Generate, edit, delete and retrieve invoice documents	• Enhanced operational efficiency • Facilitates seamless B2B operations • Allows business to focus on growth rather than admin
Analytics Dashboard	Functional	Track key business performance metrics with intuitive visualisation tools	• Delivers actionable insights on business performance • Identifies trends and patterns in ordering behavior • Supports data-driven decision making • Highlights opportunities for operational improvements
Bulk-Order CSV Upload	Functional	Import orders via CSV or excel format to streamline the processing of large orders	• Eliminates manual data entry for batch transactions • Handles seasonal spikes in order volume efficiently • Minimizes data entry errors in complex orders • Enables rapid onboarding of new customers with existing data

Chatbot	Functional	Receive proactive support from a chatbot on navigating our platform	• Provides 24/7 assistance without staffing costs • Reduces learning curve for new users
Email-Notifications	Functional	Automatically send emails on account creation, order creation, order deletion and order update to keep customers informed	• Keeps customers informed at every stage of the process • Reduces customer service inquiries about order status • Builds trust through transparent communication
API-Integration / Interoperability	Non-Functional	Easily integratable with external API's without requiring complex configuration or additional development	• Connects seamlessly with existing business systems • Eliminates data silos across the organization • Reduces implementation time and costs • Creates a unified business ecosystem
JWT Security	Non-Functional	All endpoints are encrypted preventing access without sufficient authentication.	• Protects sensitive business and customer data • Prevents unauthorized access to systems • Meets compliance requirements for data security
Cloud-Based Accessibility	Non-Functional	Access OrderNova's system from anywhere on any device without installation	• Enables remote work and management • Requires no IT infrastructure investment • Eliminates software installation and maintenance
User-Friendly Interface	Non-Functional	Sleek-modern user interface that supports dark/light themes to minimise eye strain, designed for people with no expertise.	• Minimizes training time and costs • Increases user adoption and satisfaction • Reduces errors from user confusion • Accommodates various work environments and preferences
Scalable Infrastructure	Non-Functional	Vercel Pro's cost efficient hosting scales with demand, requires no server maintenance and allows for zero downtime deployment	• Grows seamlessly with your business without system changes • Eliminates downtime during high-traffic periods • Reduces IT maintenance costs and complexity

OrderNova delivers a streamlined solution with carefully designed features that address the specific needs of SMEs in today's competitive business environment. Each component has been developed based on extensive research into user requirements and operational pain points, resulting in a coherent ecosystem of tools that work together to maximize efficiency and minimize friction in document processing workflows.

OrderNova's Competitive Edge:

OrderNova stands out in the crowded order management space by delivering a holistic, user-centric solution tailored to the unique needs of SMEs. Our platform not only simplifies order generation through UBL-compliant automation but also incorporates a blend of advanced features that enhance efficiency, security, and scalability.

Unlike market competitors such as Xero, which typically bundle order functionality within broader—and more expensive—business management suites, OrderNova provides a focused, specialized solution at a significantly lower cost. Our subscription model is priced at $29 per month, compared to Xero's $35 entry point, making our service a more accessible and cost-effective choice for smaller enterprises.

From an operational perspective, our lean infrastructure—built on Vercel Pro ($20) and Google Workspace ($14)—results in monthly operating costs of just $34. This means OrderNova achieves breakeven after securing just two subscribers, with all additional subscriptions generating near-100% profit margins. This model positions us strongly for sustainable growth and aggressive scaling while continuing to offer unmatched value to our customers.

User Stories:

User Story 1: Wilson – Solo Vintage Clothing Store Owner

Background:

Wilson runs a small vintage clothing store by himself. Each week, he manually types out supplier orders for new stock and invoices for buyers.
This process is time-consuming, error-prone, and leaves him working late nights, cutting into his personal time and delaying customer orders.

Pain Points:

- Typing out orders and invoices manually takes hours every week.
- Errors in order quantities or pricing lead to confusion with suppliers and lost money.
- Lack of standardised templates means Wilson must manually format each document.
- Managing multiple suppliers at once becomes chaotic without an organised system.
- His business growth is limited because he spends too much time on administration.

How OrderNOVA helps Wilson:

- **Automated Order Creation:** Wilson can select products, suppliers, and quantities through a simple web form, and OrderNOVA automatically generates and sends the orders.
- **Bulk Order Upload:** When Wilson needs to send a large batch of orders, he can simply upload a CSV or Excel file to import all orders at once — saving him from typing each one individually.

- **Invoice Management:** Wilson can generate polished, standardised invoices with a few clicks, minimising human errors.
- **Real-Time Notifications:** He receives immediate updates when orders are confirmed, updated and created allowing him to manage his inventory more efficiently.
- **Analytics Dashboard:** He can see which products sell the most, helping him to make smarter stock purchasing decisions.

Use Case Summary:
As a solo store owner, Wilson needs an automated system with bulk upload support to generate orders and invoices efficiently, reduce manual mistakes, and reclaim valuable working hours.

User Story 2: Mario – Busy Kebab Shop Owner

Background:
Mario operates a high-volume kebab shop that also offers catering services for events and large groups. He manually tracks bulk ingredient orders and manually calculates invoices for catering jobs. Errors in these processes have led to stock issues and lost clients.

Pain Points:

- Bulk ordering large quantities of ingredients is tedious and prone to mistakes.
- Mispriced catering invoices have led to disputes and lost trust.
- Tracking multiple catering events manually leads to missed orders or stock shortages.
- Manual entry slows him down when business volume spikes.

How OrderNOVA helps Mario:

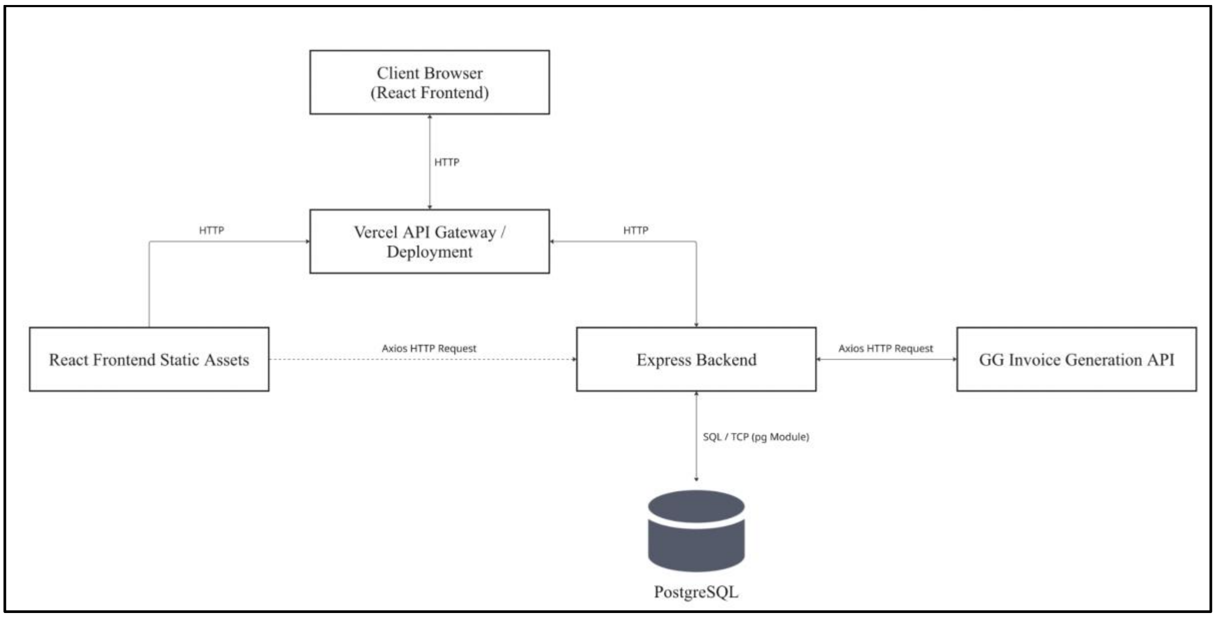
- **Bulk Ordering System:** Mario can quickly build large orders through the web platform — ideal for ordering hundreds of items at once.
- **CSV/Excel Bulk Import:** For even faster ordering, Mario can prepare a simple Excel or CSV file and upload it into OrderNOVA, automatically creating all his orders without retyping everything manually.
- **Real-Time Transaction Updates:** Mario receives instant updates on order statuses, so he can act quickly if suppliers delay.
- **Chatbot Support:** Immediate chatbot guidance helps him troubleshoot issues without disrupting operations.

Use Case Summary:
As a restaurant and catering business owner, Mario needs a system that can import bulk orders via spreadsheets and automate invoicing, allowing him to grow his business without increasing administrative overhead.

Architecture Diagram:

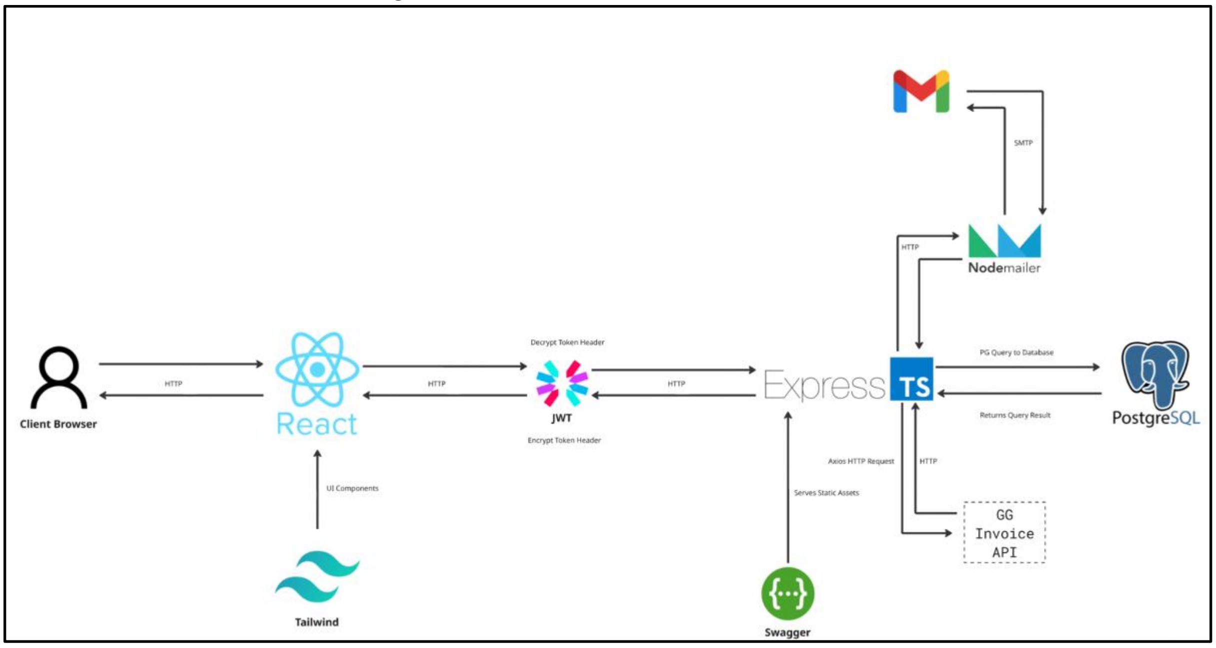
The following diagrams illustrate the architecture of OrderNova. The first provides a high-level overview of the major services and their interactions, while the second breaks down the internal module structure, detailing how core functionalities are organized and maintained.

Diagram. OrderNova Major Service Architecture



Client Browser (React Frontend)
HTTP
Vercel API Gateway / Deployment
HTTP
HTTP
React Frontend Static Assets
Axios HTTP Request
Express Backend
Axios HTTP Request
GG Invoice Generation API
SQL / TCP (pg Module)
PostgreSQL

Diagram. OrderNova Modules Architecture



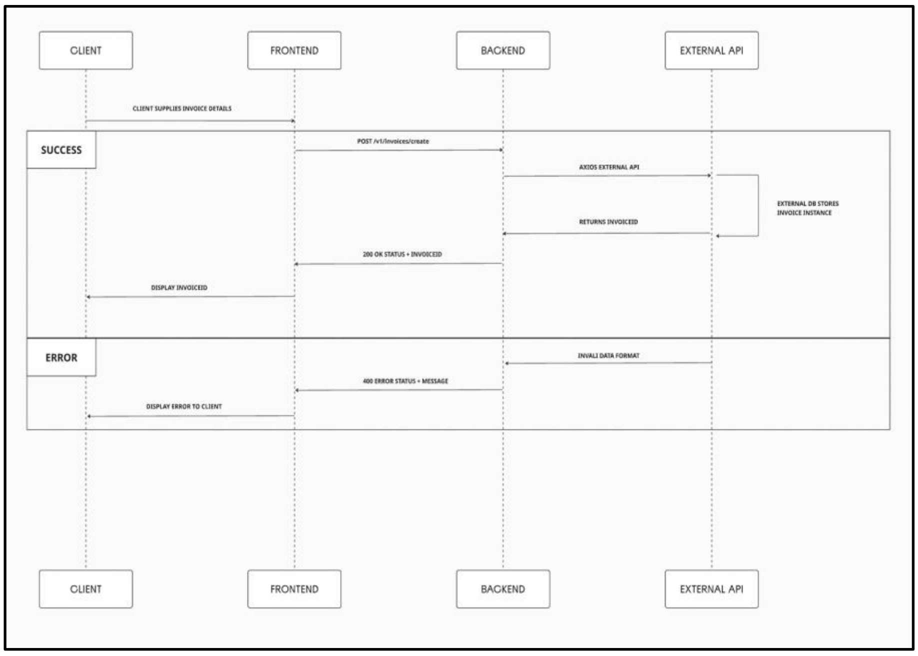
Sequence Diagrams:

Invoice Creation

Given: Wilson has selected products and entered order details in the OrderNOVA system.

When: Wilson clicks the "Generate Invoice" button to create an invoice for the selected order.

Then: OrderNOVA generates a standardized, polished invoice with the correct order details and sends it to the customer, minimizing human errors and saving time.

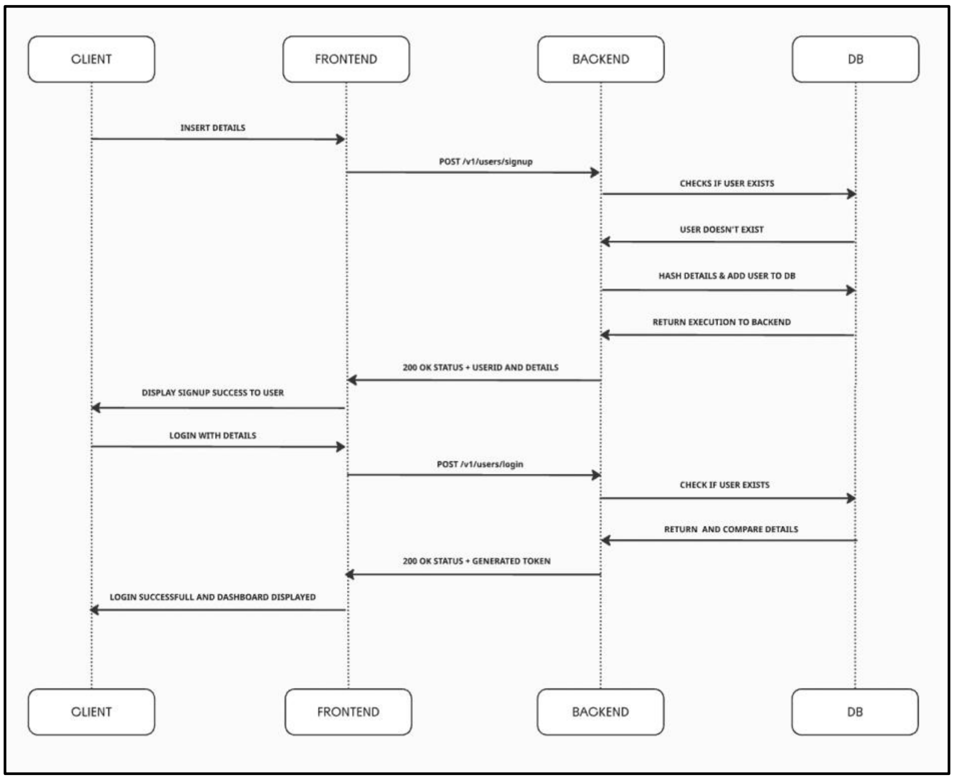


Order Creation

Given: Wilson needs to place an order for new stock from a supplier.

When: Wilson selects the desired products, suppliers, and quantities using the web form.

Then: OrderNOVA automatically generates and sends the order to the supplier, saving Wilson from manually typing out each order and reducing the chance of errors.

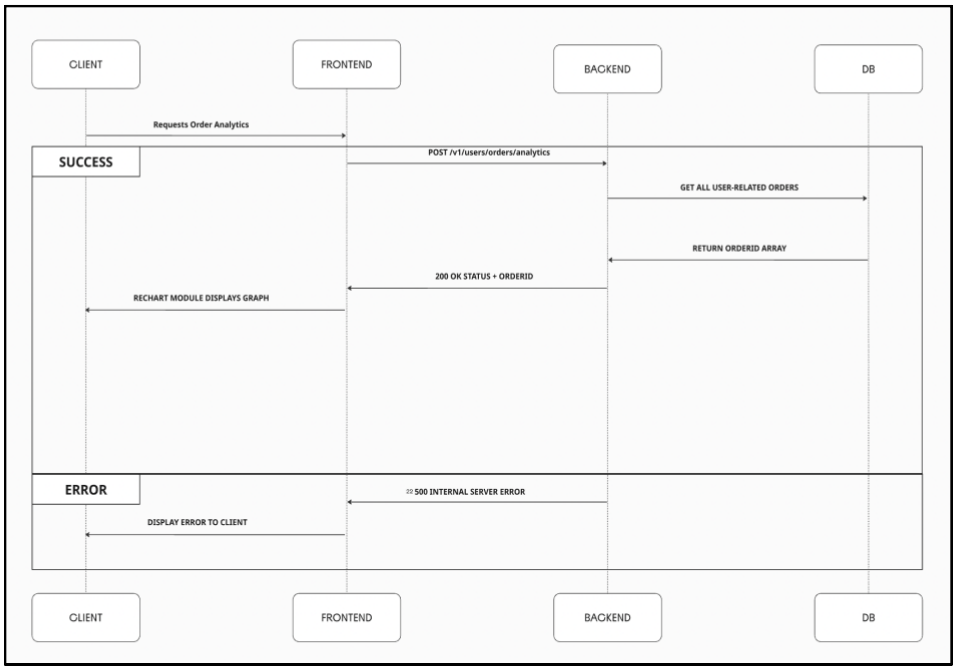

CLIENT
FRONTEND
BACKEND
DB
INSERT DETAILS
POST /v1/users/signup
CHECKS IF USER EXISTS
USER DOESN'T EXIST
HASH DETAILS & ADD USER TO DB
RETURN EXECUTION TO BACKEND
200 OK STATUS + USERID AND DETAILS
DISPLAY SIGNUP SUCCESS TO USER
LOGIN WITH DETAILS
POST /v1/users/login
CHECK IF USER EXISTS
RETURN AND COMPARE DETAILS
200 OK STATUS + GENERATED TOKEN
LOGIN SUCCESSFULL AND DASHBOARD DISPLAYED
CLIENT
FRONTEND
BACKEND
DB

Sign Up / Login

Given: Wilson is a new user or has an existing account on OrderNOVA.

When: Wilson enters his account credentials or signs up with his business information.

Then: He gains access to the OrderNOVA platform, allowing him to automate order and invoice generation, track analytics, and improve his business efficiency.

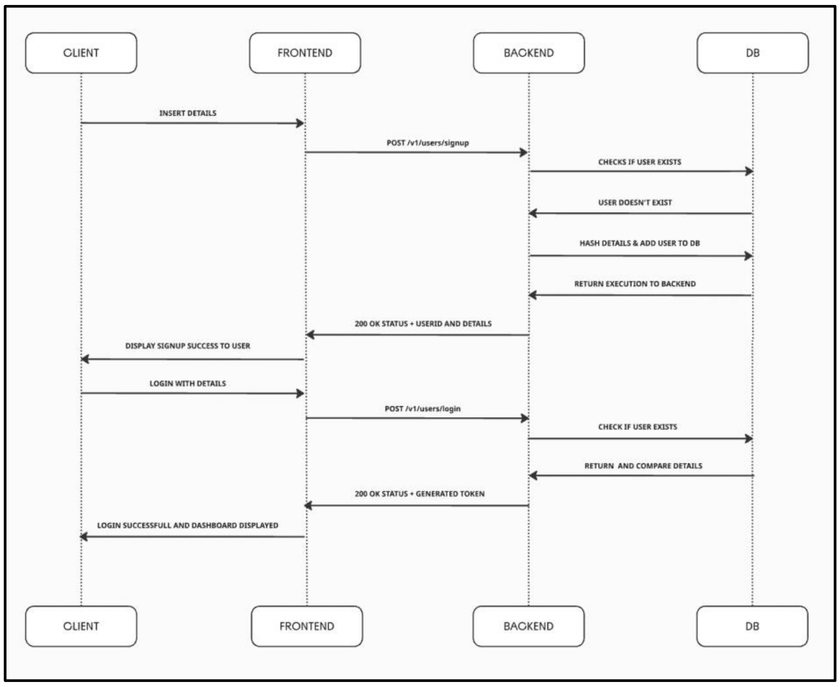


Analytics

Given: Mario has been using OrderNOVA for some time and has generated orders and invoices.

When: Mario navigates to the analytics dashboard.

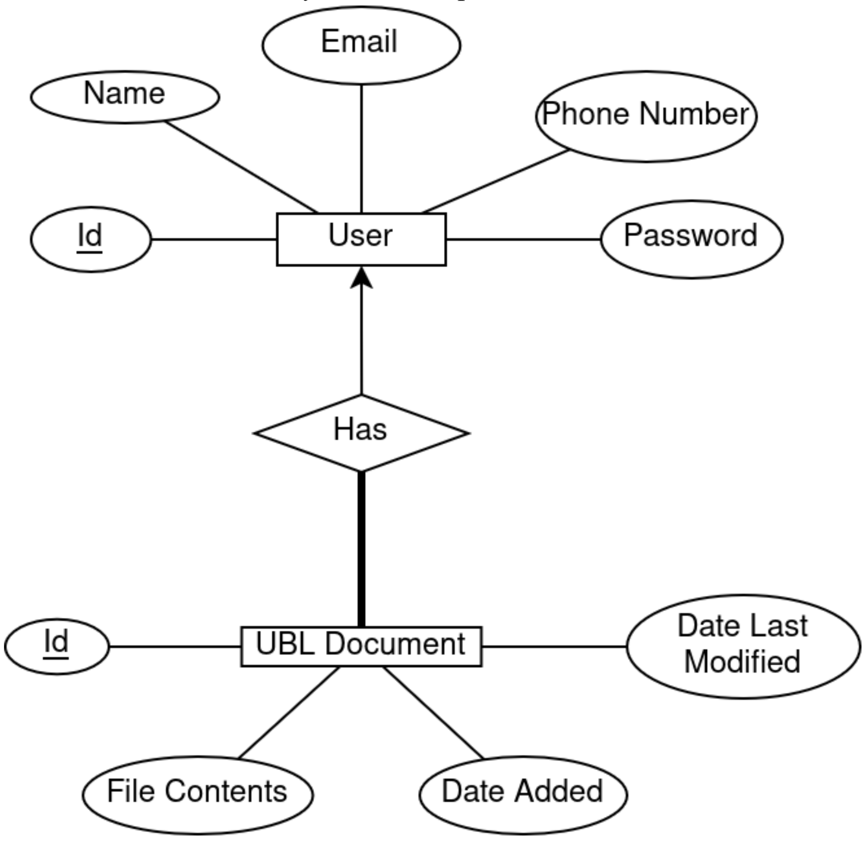
Then: The system displays a report showing which products are selling the most, helping Mario make informed decisions about stock purchasing and improving his business operations.


CLIENT
FRONTEND
BACKEND
DB
INSERT DETAILS
POST /v1/users/signup
CHECKS IF USER EXISTS
USER DOESN'T EXIST
HASH DETAILS & ADD USER TO DB
RETURN EXECUTION TO BACKEND
200 OK STATUS + USERID AND DETAILS
DISPLAY SIGNUP SUCCESS TO USER
LOGIN WITH DETAILS
POST /v1/users/login
CHECK IF USER EXISTS
RETURN AND COMPARE DETAILS
200 OK STATUS + GENERATED TOKEN
LOGIN SUCCESSFULL AND DASHBOARD DISPLAYED
CLIENT
FRONTEND
BACKEND
DB

Data Model:

Since we decided to use PostgreSQL, a relational database engine, we used the entity-relationship data model.

Entity Relationship Data Model



PostgreSQL Schema

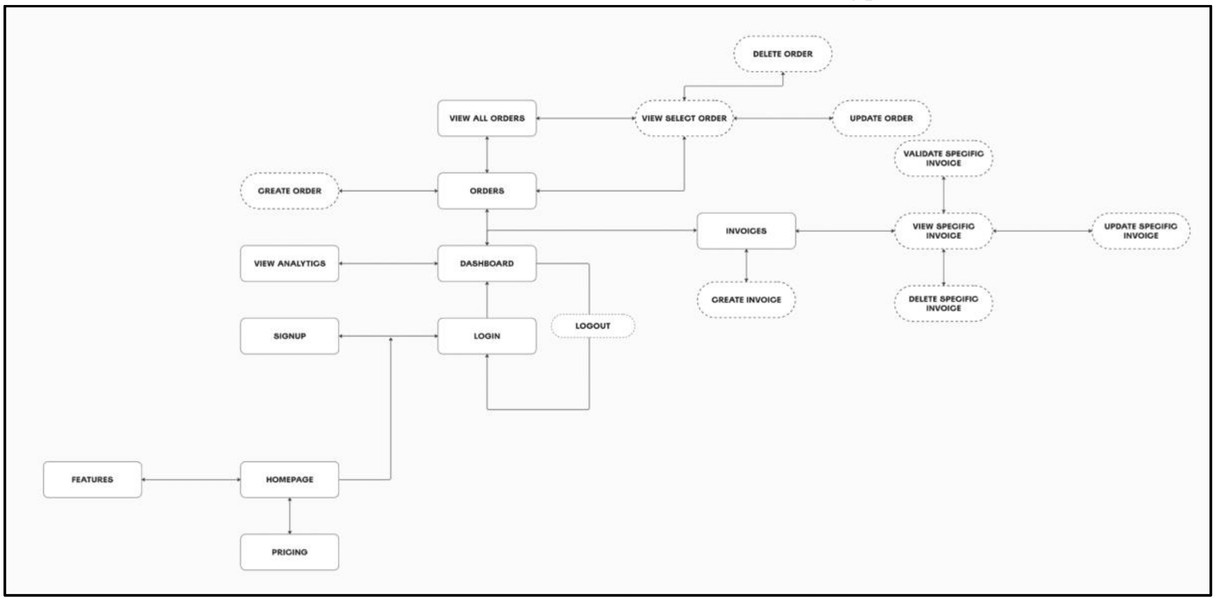
```
create table if not exists users(
        userId text primary key,
        userEmail text not null unique,
        name text,
        phone text unique,
        password text not null);

create table if not exists ubldocs(
        orderId text primary key,
        userId text references users(userId),
        fileContents text,
        dateAdded bigint not null,
        dateLastModified bigint not null);
```

Password hashes are stored in the database, userId and orderId are generated by the backend.

Frontend Models.

The following models represent the design progression of OrderNova's user interface. The low-fidelity wireframe outlines the basic flow and layout of the website, focusing on user navigation and structure, while the high-fidelity model presents a detailed visualization of the final frontend design, incorporating branding, styling, and functionality.

OrderNova Frontend Flow WireFrame Prototype



[Figma frontend prototype]

APIs Used:

API Name	**Link**
GG Invoice Generation and Validation	GG Invoice API

Testing Report

Testing In Isolation

In our testing, we encountered a challenge with testing due to Jest's default behaviour of running tests in parallel. This caused issues when multiple tests accessed our PostgreSQL database simultaneously, leading to conflicts, such as tables being dropped between tests or data being inadvertently modified by concurrent test runs. Since our project heavily relies on a shared database resource, it became clear that we needed a solution to avoid these issues and ensure the integrity of our tests.

To address this, we decided to use Jest's --runInBand option, which forces Jest to run tests sequentially, one after another, rather than in parallel. This approach allowed us to isolate each test and avoid the complications associated with concurrent access to the database. By ensuring that only one test is running at a time, we could maintain a consistent state in the database and prevent race conditions from affecting the results of our tests.

While running tests in isolation can sometimes increase the overall test execution time, this trade-off was necessary to ensure that our database remained stable and reliable during testing. Additionally, by using --runInBand, we were able to pinpoint issues more easily, as each test ran in a controlled environment with no interference from other tests. This approach significantly improved the reliability of our test suite, and we were able to ensure that our database interactions were thoroughly validated without the risk of side effects from parallel test execution.

Testing Time

Because of the complex nature of our testing, the time for each test was artificially extended to be within 10000-20000 ms because of the sheer amount of database requests and asynchronous logic taking place, a time below this is not feasible given the bare minimum necessity to make queries to the database.

Testing Tools

We utilised the jest testing software to test our endpoints. Where we could not directly test an endpoint or feature we have approved of functionality directly by observing behaviour such as for our email sending feature.

Testing Commands

For coverage:

- npx jest --runInBand --coverage

For General Testing

- npm test --runInBand anyTestFile.test.ts OR npm test --runInBand

If you would like to test our service without directly accessing the repo we have OpenAPI ver.3 swagger documentation that allows you to test endpoints.

Test Results

The below image is the result of running npx jest --runInBand --coverage on the MVP for sprint 4, it has a coverage of 96.86% which is extremely high owing to the breadth of tests we have developed. It is also 63/63 passing test cases.

Table. Endpoint Tests

Endpoint	**Test File**	**Test Cases**	**Notes**
/v1/users/signup	login.test.ts	● success case ● Invalid name ● Invalid email ● Invalid password ● Invalid phone number	
/v1/users/login	login.test.ts	● success case ● email doesn’t exist ● incorrect password	
/v1/users/logout	login.test.ts	● successful logout ● invalid token	
/v1/users/orders/create	orderCreate.test.ts	● success case ● missing order data fields ● negative item quantity	

/v1/users/orders/fetch/{orderId}	orderFetchSingleOrder.test.ts	● success case ● orderId is invalid ● userId doesn't match on orderId	
/v1/users/orders/update/{orderId}	orderUpdate.test.ts	● success case ● missing fields in update order data ● orderId is invalid ● order body is empty	
/v2/users/orders/update/{orderId}	orderUpdate.test.ts	refer to email.test.ts	refer to the Qualitative table.
/v1/users/orders/delete/{orderId}	orderDelete.test.ts	● success case ● invalid orderId	
/v2/users/orders/delete/{orderId}	orderDelete.test.ts	refer to email.test.ts	refer to the Qualitative table.
/v1/users/orders/fetch	orderFetch.test.ts	● success case ● incorrectly formatted JSON query ● user has no orders	
/v1/users/analytics/monthlycosts	analytics.test.ts	● success case ● invalid user ● date range is invalid	

/v1/invoices/create	invoice.test.ts	● success case ● invalid start date ● invalid end date ● invalid date range ● invalid number of items ● invalid item price	All Invoice related tests needed to be overhauled as the external API changed authentication and did not support backwards compatibility. Tests rewritten to support this.
/v1/invoices/update/{invoiceId}	invoice.test.ts	● success case ● invalid invoiceId	
/v1/invoices/fetch/{invoiceId}	invoice.test.ts	● success case ● invalid invoiceId	
/v1/invoices/delete/{invoiceId}	invoice.test.ts	● success case ● invalid invoiceId	
/v1/invoices/validate/{invoiceId}	invoice.test.ts	● success case ● invalid invoiceId	
/v1/health	health.test.ts	● success case	

Table. Qualitative Tests

Non-Endpoint	Test File	Test Case
Email sending functionality for signup, orderCreate, orderUpdate and orderDelete	email.test.ts	• manually had to check if each route would send and email to the designated email when any of the routes were completed, could not be tested conventionally of course.
JWT Authentication	JWT.test.ts	• success case for intermediate authentication we used on all routes
Database Functionality	database.test.ts	• successful setup • successful delete • successful add user • successful remove user • successful get user • successful add order • error case order userId doesn't match user • successful remove order • successful list all order • successful get order document from user • error document doesn't exist • success update order • error cannot update order because it does not exist

Team Description:

Our team consisted of five members, each bringing a unique skill set to the project:

- Kai: Focused mainly on the backend development, handling core functionalities like API integration, deployment, and testing of the backend. He also contributed in developing the frontend for Invoice creation.

- Manish: Led frontend development alongside Thomas, focusing on implementing the UI and ensuring the design was responsive and user-friendly. He implemented the backend and frontend for bulk ordering, smart chatbot and contributed to the backend by implementing routes for the first version of the service.
- Thomas: Worked closely with Manish on frontend development. He was responsible for translating the Figma designs into functional components, focusing on user experience and interface refinement. He also contributed to the backend by implementing routes for the first version of the service.
- Sarvika: Took the lead in creating the Figma design and wireframes for the web application. She also contributed to the backend by implementing routes for the first version of the service and contributed to the frontend tasks to ensure seamless integration of design with functionality.
- Luke: Managed the Jira board, ensuring proper task organization, sprint planning, and tracking. He helped us prioritize work and kept the team aligned with project deadlines. He also contributed to the backend by implementing routes for the first version of the service.

.

Throughout the project, we used tools like Figma for design, Jira for project management, and GitHub for version control. Regular communication was maintained through Teams to discuss progress and address any blockers.

Challenges encountered:

1. **User Input Validation and Error Handling:**
   Challenge: Ensuring that user inputs were correctly validated and handled, especially when it came to complex data fields.
   Solution: We implemented strict input validation both on the front-end and back-end to ensure all data met required formats. Descriptive error messages were added to provide clear feedback, guiding users to resolve any issues quickly.
2. **Error-Free Navigation and Redirection:**
   Challenge: Ensuring that users were redirected to the correct page after performing actions, such as submitting orders or completing forms. Solution: We carefully tested and implemented reliable navigation and redirection paths, ensuring that every action led users to the appropriate page, creating a seamless user experience.
3. **Integrating Invoice API:**
   Challenge: Integrating the Invoice API presented difficulties due to different data formats from the API.
   Solution: We thoroughly tested the data exchange between our system and the API, making adjustments to ensure compatibility and correct formatting of data before submitting.
4. **Bulk Orders – Invalid Fields and Missing Information:**
   Challenge: Users occasionally added invalid fields or missed important information when submitting bulk orders.

Solution: We addressed this with real-time input validation and implemented a preview table before submission, allowing users to review and correct any errors before finalizing their orders. This minimized errors and improved overall order accuracy.

5. **Password Hashing:**
   Challenge: Storing user passwords securely is critical to protect against potential breaches.
   Solution: We implemented password hashing using a secure algorithm to store passwords in a hashed format. This ensures that even if the database is compromised, the actual passwords remain secure and unreadable.
6. **Email Notifications:**
   Challenge: It's important to inform users about account-related activities to ensure they are aware of any unauthorized actions.
   Solution: We implemented an email notification system that sends a confirmation email whenever an account is created. This provides an additional layer of security by alerting users if an account is created without their knowledge, enabling them to take quick action if necessary.

Improvements:

Throughout SENG2021, our team reflected on progress through daily stand-ups and sprint reviews, addressing blockers and reassessing priorities regularly. Jira helped us manage tasks and visualize progress, while Teams and Instagram facilitated real-time communication.
At the project's conclusion, we recognized both successes and challenges, particularly with integration and time management. Despite these, we adapted quickly, made necessary adjustments, and completed the project on time.
We are proud of our work and have identified areas for improvement, such as better time estimation and integration planning. These experiences have helped us refine our technical and teamwork skills, which will benefit future projects.
SENG2021 could further improve by:

1. **Prerequisite Knowledge and Help Sessions**
   Current Challenge: Students may struggle with some of the advanced concepts in SENG2021 because they haven’t been adequately briefed on the foundational knowledge required.
   Improvement: Ensure that prerequisite courses or preparatory material are made available to students ahead of time, so they have a solid foundation. Organize help sessions or office hours where students can drop in and get clarification on topics or get assistance with challenges they face during their projects. These sessions can help bridge gaps in knowledge and allow students to address issues as they arise.

2. **Expanded Coverage of Software Architecture and Design Patterns** Current Challenge: The course may not provide enough practical exposure to architectural decisions and design patterns, which are crucial for building scalable and maintainable applications. Improvement: Increase the focus on software architecture

concepts like microservices vs. monolithic design, and the importance of scalable and modular systems. Include in-depth discussions of common design patterns such as Singleton, Factory, Observer, and Strategy, showing students how to apply these patterns in real-world scenarios. This will help them make better decisions when designing systems and avoid pitfalls that lead to poorly structured or unscalable code.

3. **Stronger Focus on Soft Skills and Teamwork**
   Current Challenge: While technical skills are emphasized, soft skills like communication and teamwork are not highlighted enough in the course.
   Improvement: Incorporate workshops that focus on developing soft skills such as effective communication, teamwork, and conflict resolution. These are vital in professional settings where collaboration is key to the success of projects. Encouraging these skills early will help students navigate real-world challenges, ensuring they're prepared for a career in the tech industry.

4. **More Peer Reviews and Feedback**
   Current Challenge: Students may not receive sufficient peer feedback, which can result in missed opportunities for improvement and learning gaps.
   Improvement: Introduce structured peer review sessions where students critique each other's work, providing constructive feedback. These reviews should be focused on code quality, design choices, and overall project organization. By receiving feedback early and iterating based on it, students can avoid making the same mistakes and improve the quality of their final projects. Additionally, peer feedback can create a more collaborative learning environment. To support this, ensure that clear guidelines and expectations for peer reviews are communicated.