Tech Feasibility

Date: 10/17/2025
Team Name: Evergreen Systems
Project Sponsor: Kyle Montgomery
Faculty Mentor: Dr. Ana Paula Chaves
Team Members:

- (Team Lead) Asher Romanenghi
- Mark Johnson
- Tyler Sturm
- Melvin Agram

Overview: The purpose of this document is to demonstrate that the proposed project is technically achievable within the given constraints of time, resources, and performance goals. It identifies major technological challenges, explores alternative solutions, and justifies the team's chosen approaches through structured analysis. This ensures a realistic foundation for successful system implementation.

Table of Contents

1. Introduction:

1.1 The Big Picture

Gift-giving is a universal practice that strengthens personal and professional relationships, yet it remains a significant challenge for many. In the U.S. alone, the gifting market is valued at over $150 billion annually, with consumers and businesses often struggling to find thoughtful gifts and manage timely deliveries. The rise of ecommerce has amplified these challenges, as navigating vast product catalogs and coordinating across platforms can be overwhelming.

1.2 The Problem

Currently, gift-givers rely on disparate ecommerce platforms or manual processes, leading to inefficiencies like missed events, inappropriate gifts, or delayed fulfillment. Generous, Inc., a Flagstaff-based startup, aims to streamline this process but faces scalability hurdles. Brands must provide product catalog data, fulfill orders as Seller of Record, and reconcile payments, but one-off integrations with each brand are time-consuming and unsustainable. Without reusable connectors to major commerce platforms like Shopify and Salesforce Commerce Cloud, Generous cannot efficiently onboard new brands or scale its marketplace.

1.3 Our Solution Vision

Our team is developing a reference integration SDK and platform-specific connectors for Generous, targeting core platforms like Shopify, Salesforce Commerce Cloud, Zapier, and Feedonomics. This solution leverages AI-driven agentic commerce protocols (e.g., MCP, A2A) and RESTful APIs to enable seamless catalog and order integration. Key features include a flexible database for diverse product schemas, a custom checkout flow, and a seller dashboard, all built to support Generous's unique "chat to gift" experience.

1.4 Our Impact

By delivering these connectors, our solution will enable Generous to onboard brands faster, expanding its marketplace and enhancing the gifting experience for consumers and businesses. Successful implementation will result in at least one core connector in production and reusable SDK to accelerate future integrations. This will position Generous as a leader in agentic commerce, potentially capturing a significant share of the growing ecommerce market.

2. Technological Challenges

2.1 Introduction

The Generous project involves building a scalable marketplace platform with a unique agentic commerce experience, requiring integration with multiple commerce platforms and a custom checkout flow. To ensure a robust implementation, we have identified several major technological challenges that must be addressed. These challenges involve integrating emerging AI protocols, managing diverse data formats, enabling seamless platform connectivity, implementing a custom checkout system, and ensuring secure API interactions. The following subsections outline these hurdles, focusing on the high-level requirements and technical complexities involved.

2.2 Secure and Scalable Web Server

- Challenge: Ensure the satisfaction of both our client and the end users by providing a fast, reliable, and secure web server to build the program on top of.
- Details: The chosen webserver technology must be easily extensible, feature rich, proven reliable and secure, well documented, and have wide community or first party support. Each of these criteria are critical to the success of the product, as they are all core components to determining whether or not a solution can easily scale and accept future alterations to requirements after the product is handed off to the client.

2.3 Data Storage and Scalability

- Challenge: Implement a reliable and scalable database system capable of handling large volumes of unstructured catalog and order data from multiple retailer platforms. The system must accommodate varying data formats while maintaining performance, consistency, and security.
- Details: The selected data storage must flexibly manage varying data structures while maintaining speed, integrity, and security. It should support dynamic schemas, efficient querying, and indexing to ensure fast performance under high load. Scalability, reliability, and strong backup and security measures are essential to support future growth and protect sensitive retailer and customer data. Additionally, it should be documented well and widely supported to simplify integration and future maintenance.

2.4 Modular Interface

- Challenge**:** Design and implement a modular interface architecture that enables flexible feature development, independent component updates, and seamless integration of new modules without disrupting core functionality.
- Details**:** The Generous platform requires a modular front end and back end structure to support continuous innovation and easy integration of new features such as AI driven agents, checkout customizations and commerce connectors. Achieving this demands a well defined interface layer that standardizes communication between modules while maintaining clear separation of concerns. On the front end this involves implementing a component based design where each module can operate independently and be dynamically loaded or replaced.

2.5 Code Reliability

- Challenge: Prevent runtime errors and improve maintainability when handling diverse, unstructured data from multiple commerce platforms.
- Details: The chosen technology must enforce consistency and prevent runtime errors when handling diverse data formats from external APIs. It should support scalable development practices, improve code clarity, and enable early detection of mismatches between expected and actual data structures. This is especially critical in a system that integrates with multiple third-party platforms, where reliability and maintainability are key to long-term success.

2.6 Structuring Data

- Challenge: Enforce data integrity and schema consistency across diverse product catalogs while maintaining flexibility and performance.
- Details: The chosen technology must provide a way to impose structure on highly variable product and order data while maintaining flexibility for future schema changes. It should support validation to ensure data integrity, offer efficient querying for real-time access, and integrate smoothly with the broader backend stack. As the platform scales, this layer must also facilitate maintainable data models that can evolve with new connectors and catalog formats.

3. Technological Analysis

3.1 Introduction:

This section analyzes the core technologies chosen for the Generous Commerce Connectors platform. The goal is to establish a cohesive JavaScript-based tech stack that supports scalable catalog ingestion, data normalization, and order synchronization across multiple ecommerce systems. Each technology was evaluated based on scalability, flexibility, reliability, and developer efficiency to ensure both short-term feasibility and long-term maintainability. Through this process, the team identified the most effective combination for delivering a robust, secure, and high-performing connector system capable of integrating with diverse retailer APIs. The following subsections explain each technology's role, justification, and proof of feasibility within the overall architecture.

3.2 Secure and Scalable Web Server

In choosing a framework to build our webserver on top of, we are looking for wide community adaptation, a rich feature set, extensibility, detailed documentation, and proven reliability. We believe these characteristics are essential in providing our client with a solid foundation to build their business on top of while providing their end users with a low latency, modern experience.

3.2.1 Desired Characteristics

- **Prevalence:** Wide community support for extensions and plugins will help to reduce development time. Additionally, these well known modules are used and maintained by large tech companies, ensuring they are secure and reliable for public use.
- **Features:** A wide set of built in features help provide a solid foundation that allows for shorter development times, as well as greater scalability. Features such as code generation, built in optimizations, and flexible rendering provide a solid foundation for future development.
- **Scalability:** A good web server should be easily extensible for larger applications and should be geared toward flexibility. It is unclear what features may be added months or even years after the project is handed off, which means the project should be built using technologies that are highly flexible and allow for any future additions.
- **Documentation:** The inclusion of highly detailed descriptions of each subset of features greatly helps to flatten the learning curve behind the web server. Solid technical documentation can save hundreds of hours in development time by providing comprehensive knowledge of the technology all in one place.
- **Security & Reliability:** Security is essential to maintain retailer confidence in the service, storing login and product information without fear of leaks or hacks.

Additionally, the service should be able to withstand millions of user requests without encountering errors that may shut down operations.

3.2.2 Alternatives

Next.js:

The most popular and widely adopted full stack framework with React integration [2]. Comes with built in optimizations and easily extensible middleware that provide a powerful yet flexible base to build any app from. Has strong documentation [3] and provides many built in modules that handle a variety of different tasks.

Express.js:

A minimalist, unopinionated web framework built on top of Node.js to handle web server routing, middleware and processing [4]. Extremely extensible, providing a blank template to build an app on top of. Provides high performance at the cost of a much smaller feature set.

RedwoodSDK:

Prioritizes developer control by minimizing abstraction with web-first architecture. RedwoodSDK earns its namesake for being more of a development kit for Cloudflare based serverless applications rather than a full-stack framework [5]. With native React support and builtin methods to interact with Cloudflare's API, the SDK is designed with serverless architecture in mind.

3.2.3 Analysis

Desired Characteristic	*Next.js*	*Express.js*	*RedwoodSDK*
Prevalence	5/5	5/5	2/5
Feature Set	5/5	2/5	2/5
Scalability	5/5	4/5	4/5
Documentation	5/5	5/5	3/5
Security & Reliability	5/5	4/5	4/5
Total:	25/25	20/25	15/25

In every aspect, Next.js is far above the competing technologies. As the most popular full stack web framework currently, it is no surprise that it outperforms the others, providing a wider feature set, and broader community support.

3.2.4 Choose Approach

Our team has decided to use Next.js on this project because of its wide community support, proven reliability and rich feature set. We believe that this decision will help us develop our application quickly and with plenty of room for growth as the project is taken over by our client.

3.2.5 Proving Feasibility

As our first test to ensure technical feasibility, we initialized a project using the built in create-next-app utility. This operation was completed without a hitch and we built the rest of our test application on top of this base for the purposes of testing module compatibility. The skeleton code for this application is now being hosted on GitHub for the purpose of documentation and ease of access [16].

3.3 Data Storage and Scalability

In choosing a database system to support Generous's connectors, we are prioritizing flexibility, scalability, performance, and reliability. The system must efficiently handle large volumes of diverse and unstructured product and order data originating from multiple ecommerce platforms. These characteristics are essential to provide our client with a robust and future proof foundation for managing dynamic catalog information while ensuring fast, consistent, and secure access for end users.

3.3.1 Desired Characteristics

- **Schema flexibility:** The database must accommodate a wide range of product catalog formats provided by different retailers without enforcing rigid structures. This flexibility ensures smooth integration of new data sources while reducing the need for constant schema redesigns. It also allows for efficient updates as product attributes evolve across platforms.
- **High scalability:** As Generous expands to include more brands and connectors, the database must handle growing data volumes and user activity without performance degradation. Horizontal scaling and efficient data distribution are essential to maintain speed and responsiveness as the platform grows.
- **Native JSON support:** Native handling of JSON documents allows for seamless communication between the database and technologies like Node.js, TypeScript, and React. This consistency simplifies data parsing, reduces middleware complexity, and ensures smooth end-to-end integration throughout the stack.
- **Fast querying and indexing:** Efficient query execution and indexing are vital to deliver real-time responses for catalog updates and order lookups. Properly optimized indexes

and query patterns will minimize latency and maintain a fluid user experience, even with large datasets.

- **Security and reliability:** The system must include robust encryption, authentication, and access control to protect sensitive retailer and customer information. Automated backups and recovery mechanisms will further ensure data integrity and minimize downtime in case of failures or breaches.

3.3.2 Alternatives

MongoDB:

A NoSQL, document oriented database that stores data as BSON (binary JSON) documents [6]. It supports schema flexibility and horizontal scalability, making it ideal for handling unstructured or semistructured data. MongoDB Atlas also provides automated backups, security controls, and monitoring tools.

PostgreSQL:

A relational database system known for its strong ACID compliance and SQL querying capabilities [9]. It supports JSONB columns for unstructured data, but enforcing relationships between heterogeneous schemas can increase complexity and slow down performance.

Firebase Firestore:

A cloud hosted NoSQL database that offers realtime synchronization and auto scaling [10]. While user friendly and fast to deploy, Firestore has limited flexibility for complex indexing and large scale data transformations.

3.3.3 Analysis

Desired Characteristic	*MongoDB*	*PostgreSQL*	*Firebase FireStore*
Schema Flexibility	5/5	3/5	4/5
Node.js Integration	5/5	4/5	4/5
Scalability	5/5	4/5	5/5
Query Performance	4/5	5/5	3/5
Security & Reliability	5/5	5/5	3/5
Total:	24/25	21/25	19/25

MongoDB outperformed the alternatives in terms of flexibility, scalability, and integration with JavaScript frameworks. PostgreSQL remains strong for structured datasets but introduces unnecessary rigidity when dealing with dynamic catalog formats. Firebase Firestore offers convenience for smaller applications but lacks the indexing depth and control required for enterprise level commerce integrations [11].

3.3.4 Chosen Approach

After evaluating all options, MongoDB was selected as the database for Generous's SDK and Vendor Catalog Portal. Its ability to store unstructured data in JSON documents makes it ideal for managing product feeds from different platforms. Combined with Mongoose, MongoDB enables the team to define schemas where necessary, add validation rules, and optimize queries without losing flexibility. Hosting the database on MongoDB Atlas ensures reliability through automated scaling, monitoring, and backup systems allowing the platform to grow as new connectors and partners are introduced.

3.3.5 Proving Feasibility

We have successfully implemented a MongoDB skeleton program with native drive CRUD operations, comprehensive test suites, and relationship management. All implementation details are available on Github. This implementation provides development and validates our technical approach for the Generous platform [16].

3.4 Modular Interface

React was selected as the foundation for Generous' frontend interface because of its component based architecture, efficient rendering capabilities and extensive developer support. The platform's primary interface used by vendors to manage product catalogs, monitor sales data, and interact with Generous' API requires a responsive, dynamic and maintainable user experience. React's modular component structure can also streamline collaboration among developers. Because the team already possesses strong JavaScript proficiency, React provides an optimal balance between performance, scalability, and maintainability without adding unnecessary complexity.

3.4.1 Desired Characteristics

For Generous's frontend to meet the project's usability and performance goals, the ideal framework should provide:

- **Reusability:** The component based structure promotes consistency across the user interface by reusing elements such as buttons, forms and layouts. This approach reduces development time, minimizes redundancy and simplifies long term maintenance as updates can be made at the component level without affecting the entire system.
- **Performance:** The interface will leverage efficient rendering and state management techniques to ensure smooth updates during frequent UI changes. By optimizing component lifecycles and minimizing unnecessary renders the system can maintain fast load times and responsive interactions even under heavy usage.
- **Scalability:** The modular architecture enables the platform to support an expanding range of features and interface elements with minimal code refactoring. As the system grows, new components and modules can be added without disrupting existing functionality, ensuring long-term adaptability and maintainability.
- **Integration:** Designed for seamless compatibility with Next.js, RESTful APIs, and MongoDB, the modular interface will ensure smooth communication between the front end and back end systems. This alignment allows for efficient data flow, real time updates and simplified deployment pipelines across multiple environments.
- **Community Support:** Leveraging technologies with large and active communities provides access to extensive libraries, detailed documentation and powerful developer tools. This ecosystem accelerates development, reduces troubleshooting time and ensures long term sustainability through ongoing updates and community contributions.

3.4.2 Alternatives

React:

A declarative, component based JavaScript library focused on building interactive UIs [13]. Maintained by an active open-source community, React supports reusable components, hooks for state management, and seamless integration with frameworks like Next.js.

Vue.js:

An approachable framework with two way data binding and simple syntax [14]. Vue is well suited for smaller projects or rapid prototypes but can become harder to scale for large enterprise applications due to less rigid structure.

Angular:

A full featured framework built with TypeScript, offering comprehensive tooling for large applications. While powerful, Angular introduces a steep learning curve, making it less ideal for teams learning a new language.

3.4.3 Analysis

Desired Characteristic	*React*	*Vue.js*	*Angular*
Reusability	5/5	4/5	5/5
Performance	5/5	4/5	4/5
Scalability	5/5	4/5	5/5
Integration	5/5	4/5	3/5
Community Support	5/5	4/5	4/5
Total:	25/25	20/25	21/25

React scored highest across all critical areas, particularly in integration, performance, and developer support, making it the most practical choice for the Generous platform.

3.4.4 Chosen Approach

React will serve as the frontend library for building all user interfaces within Generous's Vendor Catalog Portal and internal management tools. The UI will be structured into reusable functional components, each responsible for rendering data retrieved from our API routes. React's seamless integration with Next.js enables server side rendering for improved load performance, ensuring smooth transitions between static and dynamic content.

3.4.5 Proving Feasibility

To validate React's feasibility the team prototyped core interface elements such as the product catalog view, vendor dashboard, and order summary components. These prototypes proved React's rendering performance and integration with live API data. The team measured key metrics such as responsiveness across different devices to confirm React's ability to deliver a fast and user friendly experience at scale. The initial code is hosted on Github for updates and testing [16].

3.5 Code Reliability

TypeScript was selected to enhance the reliability and maintainability of our JavaScript-based codebase, which includes Next.js and React. As Generous's SDK and connectors handle complex data from multiple commerce platforms, TypeScript's static typing ensures type safety, reducing runtime errors and improving code clarity for our team, which is critical for integrating diverse APIs and schemas.

3.5.1 Desired Characteristics

For Generous's development needs, an ideal programming language or extension must include:

- **Type Safety**: The language must provide strong compile-time type checking to catch errors early in the development process. This is especially important when handling complex and inconsistent data structures from third-party commerce platforms.
- **Developer Productivity**: The ideal tool should enhance the developer experience through features like intelligent code completion, inline documentation, and real-time error detection. These capabilities reduce debugging time and accelerate development cycles. A productive environment also encourages cleaner, more maintainable code.
- **Compatibility**: The language must integrate seamlessly with the existing JavaScript ecosystem, including frameworks like Next.js and React. This ensures that developers can adopt the tool without significant rewrites or learning curves. Compatibility also supports the reuse of existing libraries and tooling.

- **Community Support**: A large and active community ensures that developers have access to up-to-date documentation, tutorials, and third-party libraries. This support network reduces onboarding time for new team members and provides solutions to common challenges. It also signals long-term viability and continued innovation.
- **Scalability**: As the codebase grows to support new connectors and features, the language must support modular architecture and maintainable abstractions. This includes features like interfaces, generics, and modular typing systems. Scalability ensures that the platform can evolve without becoming brittle or overly complex

3.5.2 Alternatives

TypeScript:

A superset of JavaScript that adds static typing, widely used in modern web development [15]. It integrates natively with Node.js, React, and Next.js, offering strong IDE support and a large community for libraries and tools.

Flow:

A static type checker for JavaScript developed by Facebook, designed for type safety in React applications. It is less intrusive but has a smaller community and less comprehensive tooling.

Plain JavaScript (ES6+):

The standard for web development, offering flexibility but no static typing, increasing the risk of runtime errors in complex applications.

3.5.3 Analysis

Desired Characteristic	*TypeScript*	*Flow*	*JavaScript*
Type Safety	5/5	4/5	1/5
Developer Productivity	5/5	3/5	3/5
Compatibility	5/5	4/5	5/5
Community Support	5/5	3/5	5/5
Scalability	5/5	4/5	3/5
Total:	25/25	18/25	17/25

TypeScript excels in type safety and developer productivity due to its robust typing system and excellent IDE integration. Flow offers similar type safety but has less community support and tooling. Plain JavaScript, while highly compatible, lacks type checking, posing risks for a project with complex data integrations.

3.5.4 Chosen Approach

TypeScript was chosen for its superior type safety, seamless integration with Next.js and React, and strong community support. Its ability to catch errors at compile time will reduce bugs when handling diverse catalog data, while its scalability supports the modular SDK development. TypeScript's adoption in major frameworks ensures our team can leverage existing libraries and documentation.

3.5.5 Proving Feasibility

We tested TypeScript by developing sample Next.js API routes and React components that handle mock catalog data. These tests verified type safety for complex data structures and improved developer productivity through code completion and error detection in VS Code. [16]

3.6 Structuring Data

Mongoose was selected as the Object Data Modeling (ODM) library for MongoDB to provide structure and validation for the diverse, unstructured data from commerce platforms. Its schema-based approach allows the team to define flexible yet enforceable data models, simplifying integration with Next.js and ensuring data consistency across connectors.

3.6.1 Desired Characteristics

An ideal ODM or data management tool for Generous must include:

- **Schema Flexibility**: The data modeling tool must support dynamic schemas to accommodate the wide variety of product catalog formats used by different commerce platforms. This flexibility allows the system to ingest new data sources without requiring rigid schema migrations. It also enables iterative development as data models evolve over time.
- **Validation**: Built-in validation mechanisms are essential to ensure that incoming data adheres to expected formats and constraints. This protects against malformed or incomplete records that could disrupt downstream processes. Validation also improves data quality and reliability across the platform.
- **Integration with MongoDB**: The tool must offer seamless integration with the underlying database to simplify data access and manipulation. Tight coupling with the database enables efficient querying, indexing, and schema enforcement. This integration reduces boilerplate code and improves overall system cohesion.
- **Ease of Use**: The ideal solution should abstract away low-level database operations while providing intuitive APIs for defining models and performing CRUD operations. This reduces development overhead and allows engineers to focus on business logic. A user-friendly interface also accelerates onboarding and reduces the likelihood of implementation errors.
- **Performance**: Efficient data access and manipulation are critical for real-time catalog updates and order processing. The tool must support optimized queries and indexing strategies to maintain low latency under high load. Performance at scale is essential to ensure a responsive and reliable user experience.

3.6.2 Alternatives

Mongoose:

A MongoDB ODM that provides schema-based modeling, validation, and middleware for Node.js applications [8]. It simplifies complex queries and integrates well with TypeScript and Next.js.

Prisma:

A modern ORM/ODM supporting multiple databases, including MongoDB, with a type-safe query builder. It offers strong TypeScript integration but is less optimized for MongoDB's document model.

MongoDB Native Driver:

The official MongoDB driver for Node.js, offering maximum control but requiring manual schema and validation logic, increasing development time.

3.6.3 Analysis

Desired Characteristic	*Mongoose*	*Prisma*	*MongoDB Native Driver*
Schema Flexibility	5/5	3/5	2/5
Validation	5/5	4/5	1/5
Integration with MongoDB	5/5	4/5	5/5
Ease of Use	5/5	5/5	2/5
Performance	4/5	4/5	5/5
Total:	24/25	20/24	15/25

Mongoose outperforms alternatives due to its tailored integration with MongoDB, flexible schema support, and robust validation. Prisma is strong for type safety but less suited for MongoDB's document model. The MongoDB Native Driver, while performant, requires significant manual effort for schema management and validation.

3.6.4 Chosen Approach

Mongoose was chosen for its ability to balance flexibility and structure, enabling the team to define schemas for diverse product data while enforcing validation. Its seamless integration with MongoDB and TypeScript simplifies development, and its middleware support enhances query efficiency. Mongoose will streamline catalog and order management, supporting Generous's scalability goals.

3.6.5 Proving Feasibility

We tested Mongoose by creating schemas for sample catalog data, implementing validation rules, and performing operations via Next.js API routes. These tests proved that mongoose integrates easily and helps measure and validate data integrity. Ongoing testing will ensure Mongoose supports scaling as new connectors are added. [16]

4. Technology Integration

To bring together the individual technologies analyzed above into a unified system the Generous Commerce Connectors platform will follow a full-stack JavaScript architecture built for scalability and maintainability. Each technology plays a critical role in ensuring the platform's reliability and long term sustainability.

At the top level Next.js serves as the application framework that unites both the client and server sides. It provides the infrastructure for routing, server side rendering and API endpoints which enables efficient data transfer between the front end and the database. React powers the front end delivering a fast and modular user interface. Its reusable components communicate with the backend through Next.js API routes to display real time catalog and order data.

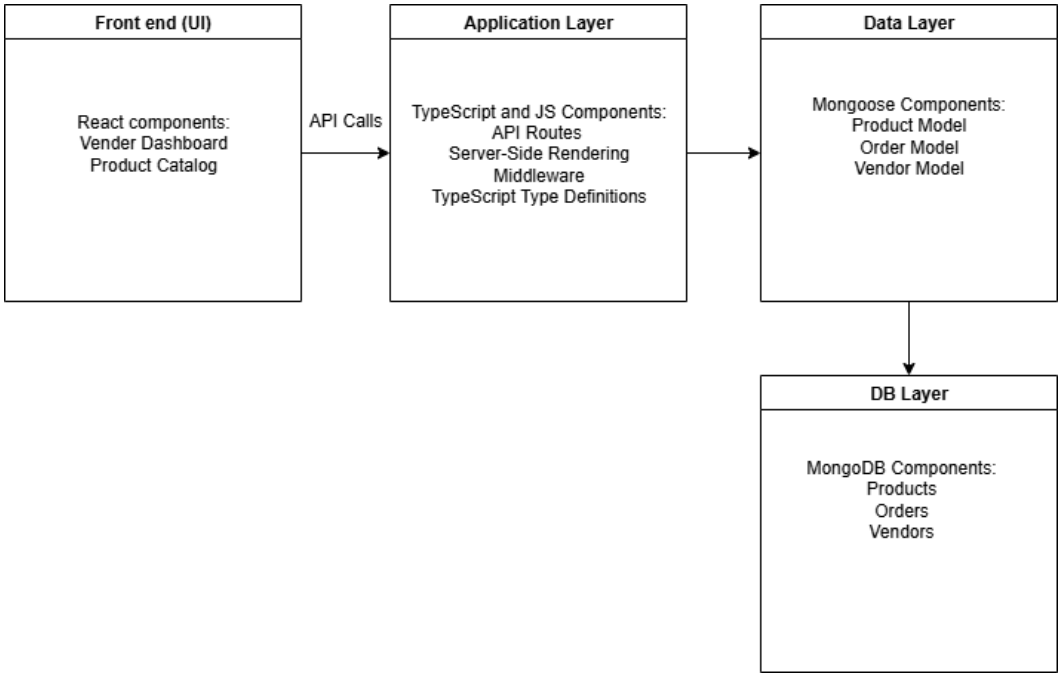
On the backend TypeScript ensures strong typing and code reliability across the full stack, preventing data mismatches between APIs and the database. TypeScript's static typing enhances maintainability as the project grows and supports a cleaner and more predictable development process.

The MongoDB database handles diverse and unstructured product and order data received from multiple retailer platforms. Its schema flexibility allows seamless ingestion of varying catalogs, while Mongoose provides structure and validation where necessary. Mongoose acts as the intermediary between Next.js API routes and MongoDB.

4.1 System Architecture Overview

Data flows through the system as follows:

1. Vendors interact with the React based frontend to view and manage catalogs.
2. User actions trigger API calls through Next.js routes written in TypeScript.
3. These routes interface with Mongoose models that validate and process the data.
4. MongoDB stores the resulting documents, which can later be queried or updated.
5. Responses are sent back up through the same pathway, updating the React interface in real time.



5. Conclusion

Understanding the current state of ecommerce integrations is essential to improving how brands connect with marketplaces and customers. Generous Commerce Connectors will enable retailers to streamline catalog uploads, manage orders, and scale their operations across multiple platforms efficiently. Identifying and selecting the right technologies for this system has been the foundation of the project, ensuring its reliability and success. Combining technologies such as React, MongoDB, Mongoose, ExpressJS, and TypeScript will create a unified, high performing platform that simplifies the integration process for retailers while supporting future scalability. We are confident in these technological decisions and believe they provide the necessary tools to move forward in building the Generous SDK and Vendor Catalog Portal, beginning with the database and core architecture that will power future connector development.

References

[1] Choosing a fullstack framework:

https://dev.to/brilworks/top-9-react-js-frameworks-developers-are-using-in-2025-4ff

[2] Stack Overflow Developer Survey:

https://survey.stackoverflow.co/2025/technology#most-popular-technologies-webframe

[3] Next.js:

https://nextjs.org/

[4] Express.js:

https://expressjs.com/

[5] RedwoodSDK:

https://rwsdk.com/

[6] MongoDB Inc. (2025). *MongoDB manual: The official documentation.* MongoDB Inc. :

https://www.mongodb.com/docs/manual/

[7] MongoDB Inc. (2025). *MongoDB Atlas documentation:*

https://www.mongodb.com/docs/atlas/

[8] Mongoose (2025). *Mongoose ODM v8 documentation.* Automattic. :

https://mongoosejs.com/docs/

[9] PostgreSQL Global Development Group. (2025). *PostgreSQL documentation.* :

https://www.postgresql.org/docs/

[10] Firebase by Google. (2025). *Firestore documentation.*:

https://firebase.google.com/docs/firestore

[11] StackShare. (2025). *MongoDB vs PostgreSQL vs Firebase Firestore comparison.* StackShare Inc. :

https://stackshare.io/stackups/mongodb-vs-postgresql-vs-firestore

[12] Next.js. (2025). *Data fetching and API routes documentation.* Vercel.

https://nextjs.org/docs/pages/building-your-application/data-fetching

[13] React:
https://react.dev/

[14] React Vs Angular Vs Vue:
www.browserstack.com/guide/angular-vs-react-vs-vue?

[15] TypeScript (2025) documentation:

https://www.typescriptlang.org/docs/

[16] Github Repository:

https://github.com/AsherSSB/generous-commerce-connectors