RavenDB

GUILHERME DINIS CRUZ, Faculty of Engineering of Porto, Portugal
LUÍS CARLOS NOVAIS ALVES, Faculty of Engineering of Porto, Portugal
NUNO MIGUEL CARVALHO DE JESUS, Faculty of Engineering of Porto, Portugal

This report provides an extensive overview of RavenDB, a NoSQL document database, and how it is used in a prototype e-commerce application for computer hardware. The report begins with an introduction to RavenDB's history, licensing schemes, and community backing, with an understanding of the technology through its background. The report talks about crucial aspects of RavenDB, including indexing, clustering, querying, scalability, and multi-platform compatibility. Methodologically, it describes the setup and normalization of product data, the creation of a dynamic JSON-based data model, and the incorporation of RavenDB into a working e-commerce proof-of-concept. The query optimization, indexing approach, and sharding performance aspects are covered, as well as a discussion of data consistency, replication, and transaction handling. The prototype shows the benefits in the real world of RavenDB for dealing with complex data structures, supporting high availability, and maintaining excellent performance. Ultimately, this document presents findings about RavenDB's suitability for application development today, contributing to developers and researchers studying document-oriented database options.

Additional Key Words and Phrases: RavenDB, NoSQL Database, ACID Compliance, Indexing, Querying, Clustering, Sharding,E-Commerce Platform.

ACM Reference Format:
Guilherme Dinis Cruz, Luís Carlos Novais Alves, and Nuno Miguel Carvalho de Jesus. 2025. RavenDB. In *Proceedings of RavenDB.* , 14 pages.

1 Introduction

1.1 History

RavenDB was founded in 2008 as a project by Oren Eini, better known as Ayende Rahien, under the name "Rhino DivanDB." Eini, an independent consultant and NHibernate developer, was inspired by the paucity of document databases designed for the.NET ecosystem. Reading CouchDB's source code inspired him to construct a REST-based document store that solved friction points in existing solutions while also providing robust support for.NET developers. According to Hibernating Rhinos Ltd., RavenDB was the first document database to run natively in the.NET Framework, as well as one of the first to give full ACID (Atomicity, Consistency, Isolation, and Durability) guarantees. [5]

Authors' Contact Information: Guilherme Dinis Cruz, up202403107@edu.fe.up.pt, Faculty of Engineering of Porto, Porto, Portugal; Luís Carlos Novais Alves, up202108727@edu.fe.up.pt, Faculty of Engineering of Porto, Porto, Portugal; Nuno Miguel Carvalho de Jesus, up201905477@edu.fe.up.pt, Faculty of Engineering of Porto, Porto, Portugal.

Manuscript submitted to ACM

1.2 Licenses and Pricing

RavenDB features a highly customizable licensing model, which accommodates all categories of users, from individual developers to large corporations. There are three free licenses—Community, Developer, and Cloud—and two paid ones—Professional and Enterprise. This enables users to choose a plan suitable for their specific needs and budget.

The Community License, the one we used, can be obtained free of charge and allows users to try out RavenDB without investment, which is ideal for small projects or studying. The Developer License is also intended for development environments with comparable benefits, but on the grounds of renewal every six months. The RavenDB Cloud free license takes this even further by allowing one to try out a managed cloud RavenDB deployment, where RavenDB itself takes care of such critical things as installation, configuration, security, patching, updating, providing high availability, and monitoring.

For others who require more intensive performance, the Professional and Enterprise licenses offer richer capabilities and support. Commercial licenses differ in cost per core, core limit, cluster memory limit, and supported nodes. The Professional license is a budget-friendly option for mid-sized companies, while the Enterprise license is for high-need companies that require advanced scalability, security, and support features.

This license model allows users to start with a free license, get used to the functionality of RavenDB, and seamlessly upgrade to a paid license as the needs grow. This provides flexibility for RavenDB to stay a viable choice for a vast spectrum of use cases, from individual developers to enterprise-level usage at large. [7]

1.3 Documentation

RavenDB offers vast documentation to help new users.[4] The official RavenDB documentation is a comprehensive guide to every part of the NoSQL database, from the most basic concepts to advanced features. It is intended to meet both beginner and seasoned developers' needs, providing in-depth details on how RavenDB operates and what it can accomplish. The documentation for the API is available for C, Java, Node.js, Python and PHP, providing code snippets and examples.

The **Basic Concepts** describes what a document store, how it is used in RavenDB and how to create one. Instruction about loading, storing and deleting entities provided in the **CRUD Operations** section. As other NoSQL databases, RavenDB provides its own query language, **RQL**, which is explored in **Querying**. The **Indexing** section demonstrates how to define and use indexes to enhance query performance. The **Advanced** section presents more complex features such, as full-text search, facets, spatial indexes, data subscriptions, and attachments, among others.

For beginners, there is a detailed getting started guide in RavenDB that discusses all the fundamentals and initial setup. [6] .

1.4 RavenDB Community

RavenDB has a well-established community that is a benefit to both beginners and veterans. One of the primary forums in which this community flourishes is GitHub Discussions[3], where users can ask questions, share ideas, and demonstrate their own custom implementations of RavenDB[2].

In addition to GitHub, RavenDB is also a popular topic on Stack Overflow, with thousands of questions tagged with 'ravendb'. The users can search for answers to common problems, ask for suggestions, or help others by providing their insights.

Besides community support, RavenDB offers official channels of support for users to resolve technical issues. The official website and the user manual can be utilized by all users to access the free support mechanism. However, quicker and advanced support is accessible for a fee via Professional Support and Production Support options. Both plans provide users direct access to RavenDB's technical team with the provision of quick solutions for problems that have to be treated seriously.

In addition to this, the CEO of RavenDB, Oren Eini, also has a blog upon which he continually publishes insights, technical guides, and arguments relating to database technologies, including RavenDB. This provides a direct interface for users to interact with the CEO.

2 RavenDB Key Features

RavenDB is a complete ACID compliant NoSQL document-oriented database written for high performance, scalability, and ease of development. The following are some of its key features.

2.1 Indexing

RavenDB utilizes a sophisticated indexing mechanism to allow high-performance queries. Static and dynamic indexes are supported:

- Static indexes: developers manually define this feature to support particular query patterns.
- Dynamic indexes: automatically created by RavenDB on an as-needed basis by queries, thus queries are always supported by an appropriate index.

This makes querying efficient without having to scan the entire data set, since RavenDB indexes each field separately and combines results at query time.

2.2 Clustering

RavenDB supports distributed cluster via multi-master replication that offers both fault tolerance and high availability. It uses the Raft consensus algorithm (implemented as Rachis) to help guarantee consistency across nodes. ACID transactions across the cluster are supported, in that transactions can cross multiple nodes and will be committed only once a majority of nodes concur.

2.3 Querying

RavenDB has a powerful querying engine with support for:

- **Raven Query Language (RQL)**: An SQL-like language specifically designed for document databases. JavaScript can be incorporated in the queries.
- **Language Integrated Query (LINQ)**: For developers using .NET, offering strong typing and IntelliSense support.
- **Full-Text Search**: Driven by Lucene.net, which enables effective text search within documents.
- **Graph Queries**: Allowing navigation of connected documents as vertices and edges, particularly appropriate for hierarchical document structures.

2.4 Sharding and Scalability

RavenDB has built-in sharding support, meaning that a database can be split over several servers. This enables horizontal scaling, meaning high-performance apps and large volumes of data can be handled.

3 Data Model and Operations

RavenDB supports a dynamic document-oriented data model, in which data is maintained as JSON documents, making it a good fit for a broad set of applications. The structure of the database is schema-free, meaning it is flexible as data evolves and changes. There are many ways to model data, including attachments (embedding binary data into documents), time series and counters.

In addition to document storage, RavenDB supports graph-like queries, time series and distributed counters.

The usual CRUD operations are available over documents, but update operations require more effort, as there is the need to bring JavaScript syntax to RQL in order to match the documents or properties to update. As for other operations, RavenDB is equipped with Lucene, a good ally that offers full-text search capabilities and enhancing queries with wildcards, fuzzy matching and other complex structures.

4 APIs and Client Libraries

There are several ways of communicating with the RavenDB database, and they are mainly through a RESTful API and specific client libraries for C#, Java, Node.js, Python, and PHP. Among them, the C# client is (definitely) the most documented and complete.

In addition to that, RavenDB Studio also provides a graphical user interface (GUI) for creating and managing databases, querying, performing direct edits on documents, visualizing queries and indexes, and viewing performance metrics, making it a useful tool for developers and administrators.

5 Consistency and Replication Features

RavenDB supports data consistency by both ACID (Atomic, Consistent, Isolated, Durable) and BASE (Basically Available, Soft State, Eventually Consistent) models, albeit on different sides of its functionality. In general, RavenDB is ACID compliant with high reliability and data integrity for most operations. Document operations, insertion, modification, or removal by their IDs, are always performed within an ACID transaction, ensuring full consistency.

Yet RavenDB also delivers performance optimization by pursuing a BASE approach in distributed systems. This yields quicker, more scalable operations by relaxing consistency where it is safe to do so. This ACID vs. BASE tradeoff is accomplished in part through Voron, a bespoke high performance storage engine for RavenDB. Voron extracts as much performance as possible while still maintaining data integrity.

Replication-wise, RavenDB follows a distributed cluster model relying on the Rachis Protocol, which is an implementation of the Raft consensus Algorithm. The protocol provides strong consistency for operations across the cluster like creating a database, node management, and electing a leader. Three and five nodes are the most typical cluster configurations, with fault tolerance and high availability.

RavenDB has a dynamic replication model. Databases can be spread over a static set of nodes with user-defined replication levels. Data is replicated asynchronously between nodes over TCP connections, so writes are always available even if some of the nodes are down temporarily.

To manage conflicting changes, RavenDB provides a Conflict Resolution Mechanism that can be customized. Users can decide to take the most recent version, utilize a particular authoritative node, or combine conflicting documents based on custom logic.

6 Use Cases

- **Real-Time Applications**: RavenDB is ideally appropriate for real-time use cases because it has low-latency and high-performance data access.
- **Financial Systems**: RavenDB provides strong consistency (ACID compliant operations) and coupled with TLS support, it enables this database to be a choice among competitors.
- **E-Commerce Platforms**: E-commerce sites benefit from RavenDB's ability to handle complex data models with ease. It secures the flexibility and evolution of product catalogs and one of its features can provide suggestions of similar documents. The Lucene integration provides full-text search, which is crucial when browsing products.
- **Healthcare Systems**: RavenDB is ideal for healthcare applications where performance, privacy, and integrity of data are the biggest concerns. Patient records, appointment management, medical images metadata, and real-time patient vital monitoring can all be achieved with ease. Its time-series data support is especially handy to track patient statistics and vitals over a range of time.

7 Prototype

7.1 Topic

We aim to create an e-commerce website for selling computer hardware and peripherals. The website will offer users a comprehensive list of products ranging from processors, graphics cards, memory modules, storage drives, monitors, cases, and other peripherals. It will enable users to browse, search, and buy products.

7.2 Data Preparation

We needed a complete and properly structured dataset. After experimenting with many options, we decided to use a collection of CSV files from a GitHub repository[1], each of which represented a different type of computer hardware. We retrieved a portion of the dataset, but we still took, an astounding, 60 000 records.

To achieve data consistency, we employed a uniform structure with a shared set of fields for all products, while category-specific fields were placed in a nested object. Images were added manually to each product, from a range of 4 images per product.

7.3 Conceptual Model

The conceptual model includes primarily **Products** and **Users** and is presented in Figure 1. Each Product has a set of mandatory attributes, along with additional, category-specific attributes. Products can also receive **Reviews** submitted by **Users**. Each user can only review a product once. Additionally, each User can place **Orders** for one or more products

7.4 Physical Model

Below we will demonstrate how the conceptual data model above was translated to a physical data model. Its important to note that IDs are automatically assigned by RavenDB by joining the name of the collection, a slash and a counter. That ID is stored in each document as metadata, but at query time, an "id" field is added to the document.

7.4.1 Product. A product contains all the fields known of a generic product while also containing a map of other attributes specific to its category. Accessing the product means retrieving the reviews and therefore, since there are no other access patterns, the reviews are tied to the product.

```
{
    "name": "Fractal Design Ridge PCIe 4.0",
    "price": 129.99,
    "category": "case",
    "stock": 5,
    "attributes": {
        "type": "Mini ITX Tower",
        "color": "White",
        "side_panel": "Mesh",
        "external_volume": 16.3,
        "internal_35_bays": 0
    },
    "reviews": [],
    "discount": 20,
    "image": "/products/case/4.jpg",
}
```

7.4.2 Review. Reviews are fetched whenever a product is requested. All reviews are retrieved at the same time.

```
{
    "user": "Anonymous",
    "rating": 5,
    "title": "Great product!",
    "text": "I love this product!",
    "date": "17-05-2025"
}
```

7.4.3 User. The Users collection was manually populated with simple users. Each user detains the orders they complete, since those orders are only requested when accessing the user's order history. Those users can be accessed via id, which, again, is formulated in query-time.

```
{
    "firstName": "Nuno",
    "lastName": "Jesus",
```

```
    "email": "nuno.jesus@gmail.com",
    "orders": []
}
```


7.4.4 Order. An order detains its date, the inputted address and the items the order encompasses. We chose to store the items as an array of nested objects, since changes to the database, like updates or deletions, could compromise the display of an order's products.


```
{
    "date": "2025-05-17T12:24:46.190Z",
    "address": {
        "street": "Rua Nova",
        "city": "Porto",
        "state": "Gondomar",
        "zip": "4435-123"
    },
    "items": [
        {
            "name": "Intel Core i9-14900K",
            "price": 544.99,
            "category": "cpu",
            "stock": 4,
            "attributes": {
                "core_count": 24,
                "core_clock": 3.2,
                "boost_clock": 6,
                "tdp": 125,
                "graphics": "Intel UHD Graphics 770",
                "smt": true
            },
            "reviews": [],
            "discount": 20,
            "image": "/products/cpu/intel-3.jpg",
            "id": "products/8369",
            "quantity": 1
        },
    ]
```

}

7.5 Architecture

RavenDB is used as the document-oriented NoSQL database. On the back-end, we use Node.js with Express.js, which interfaces with RavenDB to handle data access and API endpoints. The front-end is developed with React, providing a web interface that displays the data and allows interactions with the API. This is depicted in Figure 2.

7.6 Implemented Features

The system integrates several advanced features to enhance querying, performance, and user experience. These functionalities leverage the capabilities of RavenDB along with custom logic in the Node.js backend.

7.6.1 Pagination. To efficiently manage large datasets and improve performance, *pagination* is implemented using the `skip` and `take` parameters. This allows the front-end to request a specific subset of documents, reducing the data transferred and optimizing rendering on the client side.

7.6.2 Auto-indexing. Auto-indexing is one of the most important features of RavenDB. Each time we query the database, a new index is formed to enhance future accesses to the database. Other queries may benefit from other indexes as well.

7.6.3 Full-Text Search with Wildcards. The search functionality uses **RavenDB's StandardAnalyzer** for *full-text search*, with support for:

- Wildcards (*) for partial matches (e.g., `compu*` matches "computer", "computing", etc.).
- Tokenization and normalization to improve match accuracy.

7.6.4 More Like This. The *More Like This* feature from RavenDB is used to implement content-based recommendations to suggest products similar to the one currently viewed.

7.6.5 Aggregation through Time Series Values. This is used to track reviews over a product and produce the average score, since aggregation is not directly supported on the documents.

7.7 RavenDB vs MongoDB

RavenDB and MongoDB are both document-oriented NoSQL databases, yet they diverge significantly in their architectural philosophies and operational features. RavenDB emphasizes strong consistency and operational simplicity, offering ACID transactions by default, which ensures data integrity across distributed systems without additional configuration. Its built-in features, such as full-text search powered by Lucene, automatic indexing, and a SQL-like query language (RQL), provide a comprehensive out-of-the-box experience. These capabilities reduce the need for external tools and extensive administrative oversight, making RavenDB particularly appealing for applications where data consistency and ease of maintenance are paramount.

Conversely, MongoDB offers a flexible, schema-less design that facilitates rapid development and scalability. Its robust ecosystem, extensive community support, and managed cloud services like MongoDB Atlas make it a popular choice for a wide range of applications. MongoDB's support for horizontal scaling through sharding and its rich querying capabilities cater to high-throughput, large-scale deployments. However, achieving strong consistency and optimal performance often requires careful configuration and a deeper understanding of its operational intricacies.

At first glance, RavenDB seems more friendly mainly to its SQL-like syntax and the auto-indexing. However, the documentation often fails developers. The documentation may seem well-designed at first, but the deep nesting of sections makes it hard to find concepts. Moreover, despite the languages availability mentioned earlier, the documentation lacks implementation of some features on all languages, narrowing our choices when designing the backend of the application. Another problem entails how incomplete the examples are. For instance, demonstrating how to implement a query in Node.js but failing to do so in RQL.

The auto-indexing feature is good for large projects, but as any normal index, it occupies space. Within an environment with low memory space, the auto-indexing feature might have to be disabled. Moreover, in periods of high writes, the indexes might provide inaccurate data as they haven't been updated yet.

8 Conclusion

In summary, this project presented a detailed examination of RavenDB as a document-oriented NoSQL database, its features, and structural make-up, as well as its real-world application in an e-commerce web application. The analysis and utilization of RavenDB flexibility, ACID support, and its sophisticated features, including indexing, clustering, and full-text search, were instrumental in the prototype design.

The proof-of-concept showed RavenDB's ability to support sophisticated data models, efficient querying, and scalability, making it a suitable candidate for high-performance applications. There were, however, noted limitations, especially with respect to sophisticated analytics and scalability within specified parameters.

This research provided significant information on the development, configuration, and tuning of RavenDB and hence facilitated a better understanding of its advantages and limitations. The findings of this research are anticipated to guide developers in their quest for effective NoSQL technology and influence future research on document-oriented database technologies.

References

[1] docyx. 2025. *PC Part Dataset.* https://github.com/docyx/pc-part-dataset
[2] Hibernating Rhinos. 2025. *RavenDB Discussions on GitHub.* https://ravendb.net/articles/ravendb-discussions-on-github
[3] RavenDB Community. 2025. *RavenDB Discussions on GitHub.* https://github.com/ravendb/ravendb/discussions
[4] Hibernating Rhinos. 2025. *Docs Guide - RavenDB NoSQL Database.* https://ravendb.net/learn/docs-guide
[5] Hibernating Rhinos. 2025. *RavenDB About Us.* https://ravendb.net/about
[6] Hibernating Rhinos. 2025. *RavenDB Getting Started.* https://ravendb.net/docs/article-page/7.0/csharp/start/getting-started
[7] Hibernating Rhinos. 2025. *RavenDB Licenses and Prices.* https://ravendb.net/buy

A Conceptual Model

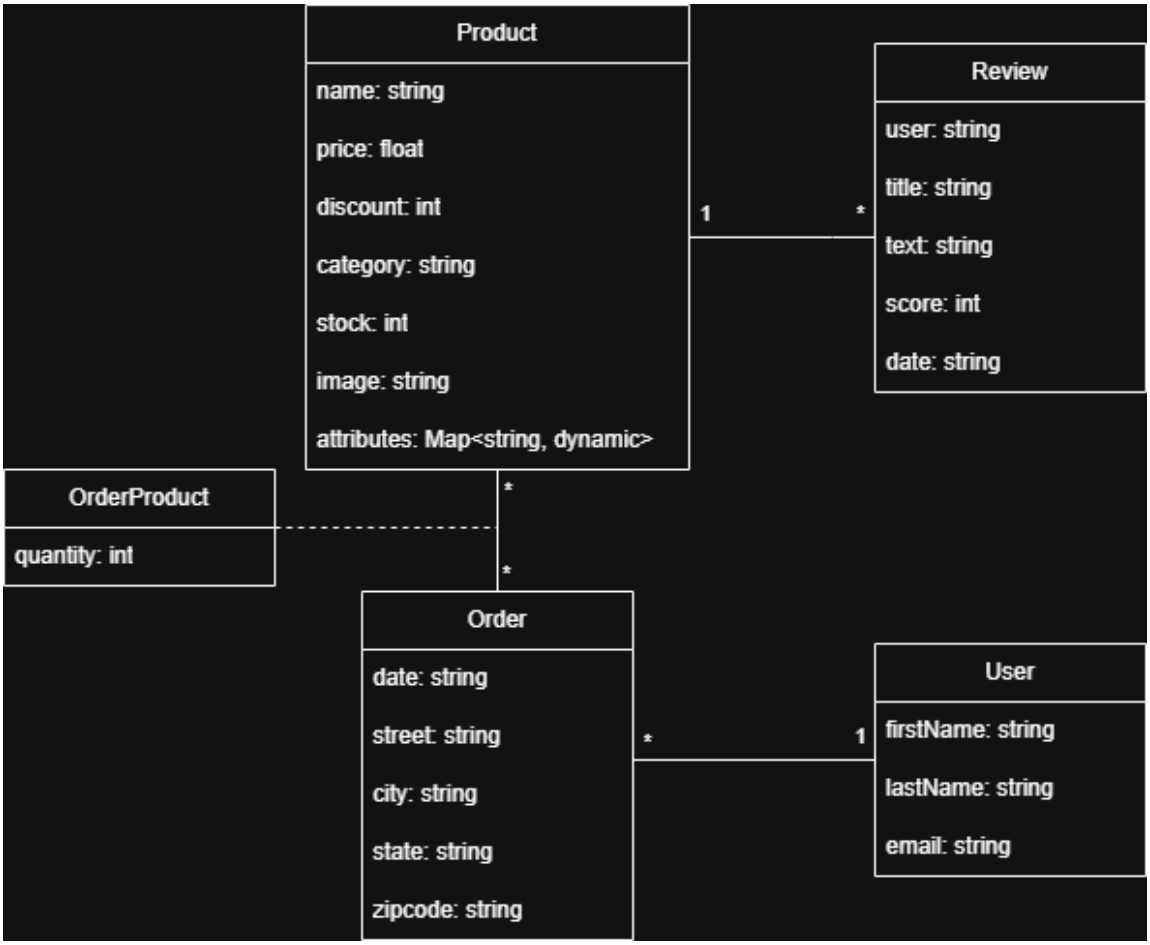

Fig. 1. Conceptual Data Model

B Architecture

Fig. 2. Prototype Architecture

C Screenshots

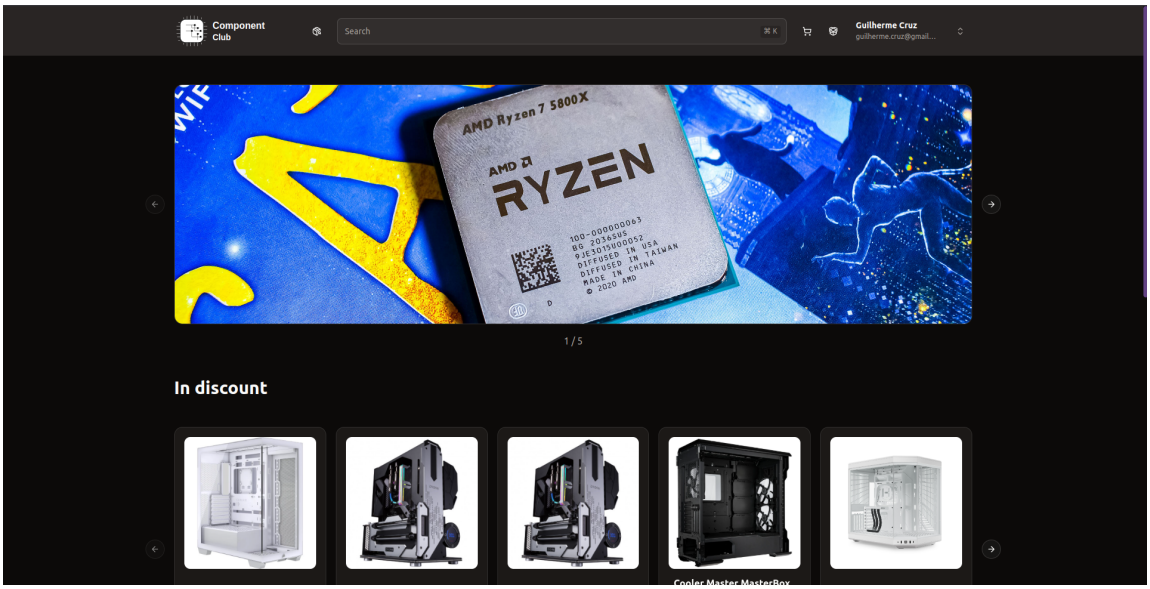

Fig. 3. Home Page

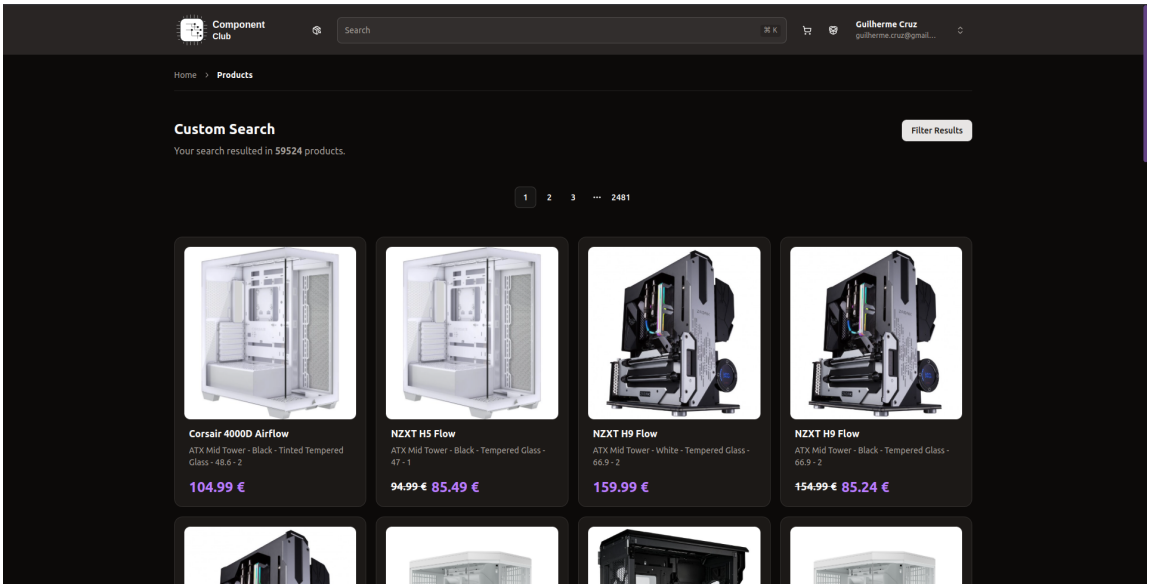

Fig. 4. Products Page

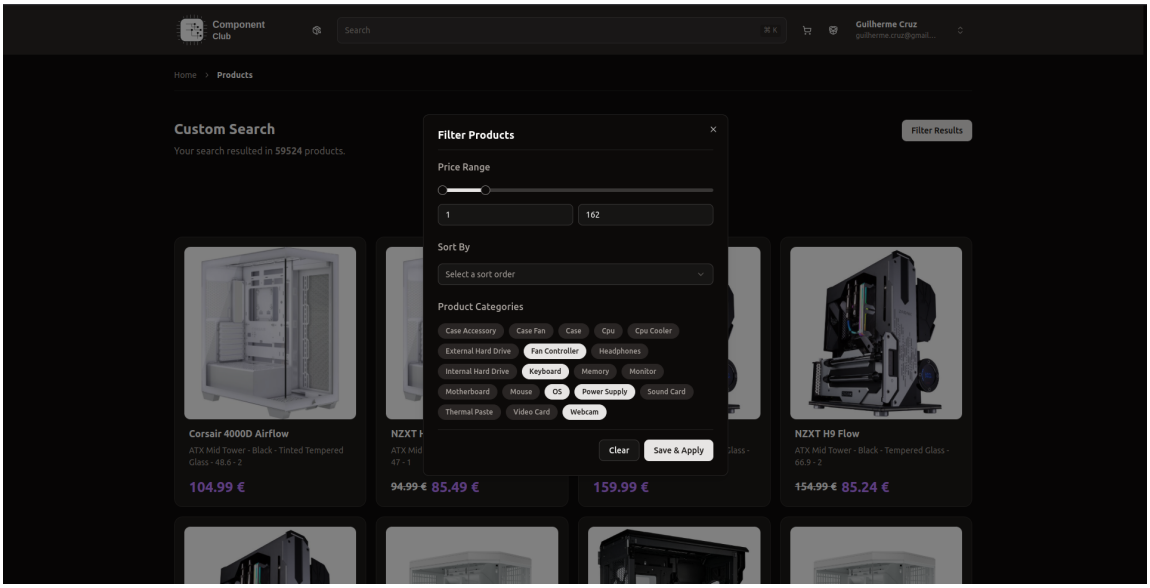

Fig. 5. Filters

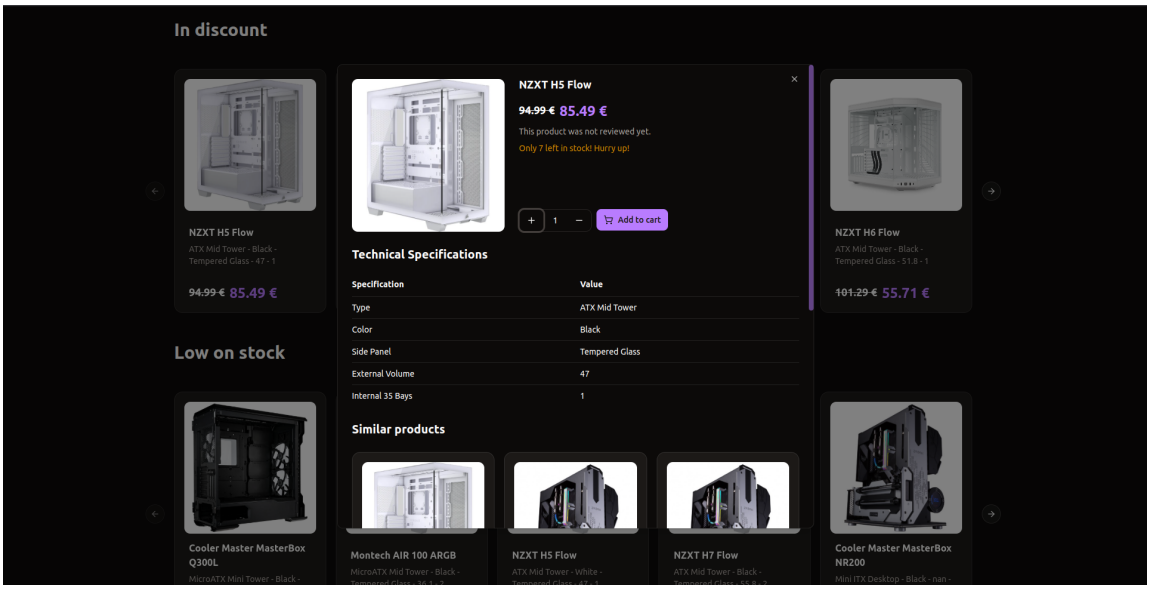

Fig. 6. Product Dialog

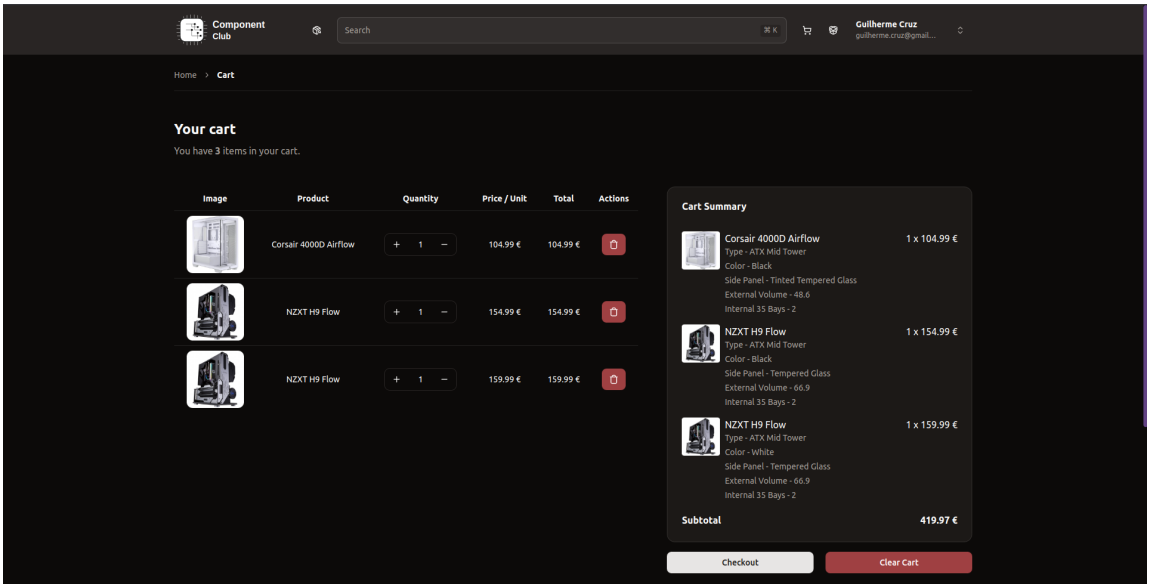

Fig. 7. Cart Page

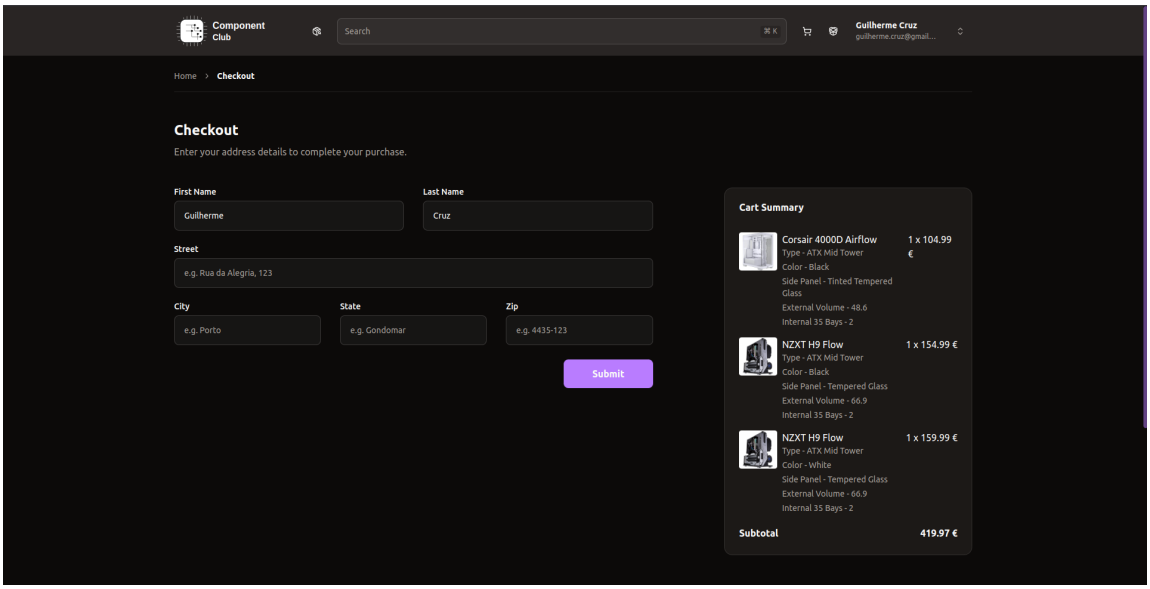

Fig. 8. Checkout Page

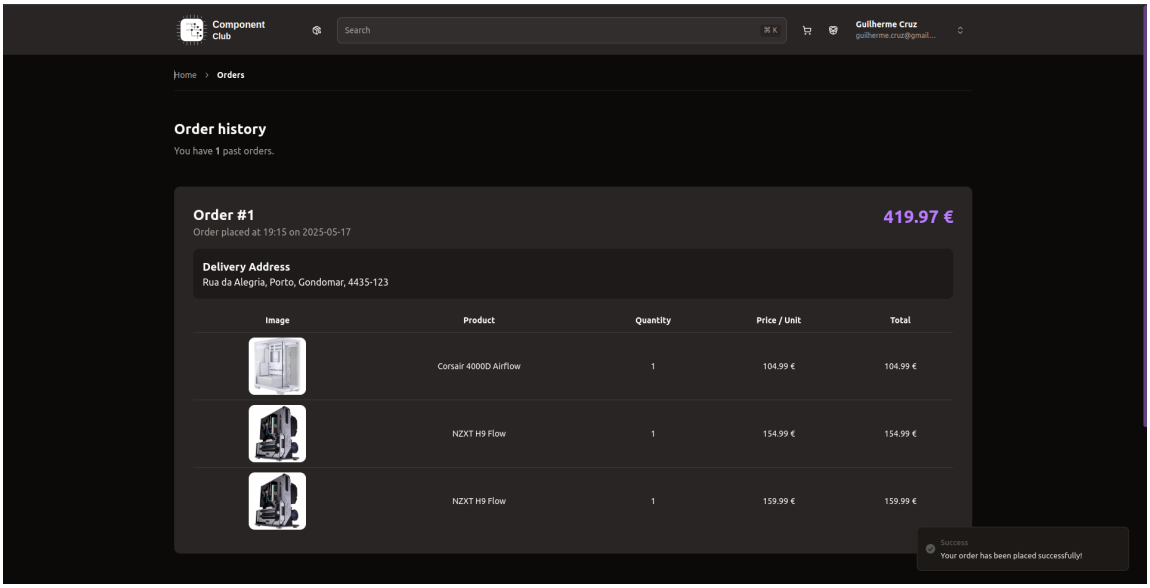

Fig. 9. Orders Page