Design & Architecture of Erbut, An Elegant Web Based Linkfolio

Yazeed AlKhalaf
Khalil Melhem
Khaled Hazzam
Course: SWE 302 - Software Design & Architecture
Instructor: Dr. Ahmed Ghoneim

12 Feb, 2024

Contents

Abstract

Erbut presents a dynamic, web-based platform enabling users to create and manage personalized 'Erbuts'—elegant linkfolios showcasing a curated list of links. These linkfolios, hosted as subdomains on erbut.me or linked to custom domains, offer a unique way to display online presence.

Additionally, users can integrate Google Analytics for enhanced data collection. The subscription model, facilitated via Stripe, unlocks premium features, enhancing user experience and customization. Erbut stands out as a versatile solution for creating and monitoring professional and personal online portfolios and brands.

This paper extensively explores the design and architecture of Erbut, focusing on its robust, scalable backend and user-friendly frontend. It highlights the integration of technologies for domain management, analytics, and payment processing, emphasizing the platform's efficiency and security.

Chapter 1

System Overview

This section will showcase the system from an internal point of view. Using UML diagrams to draw the system, reading this section should allow the reader to understand how the system is built on a high level.
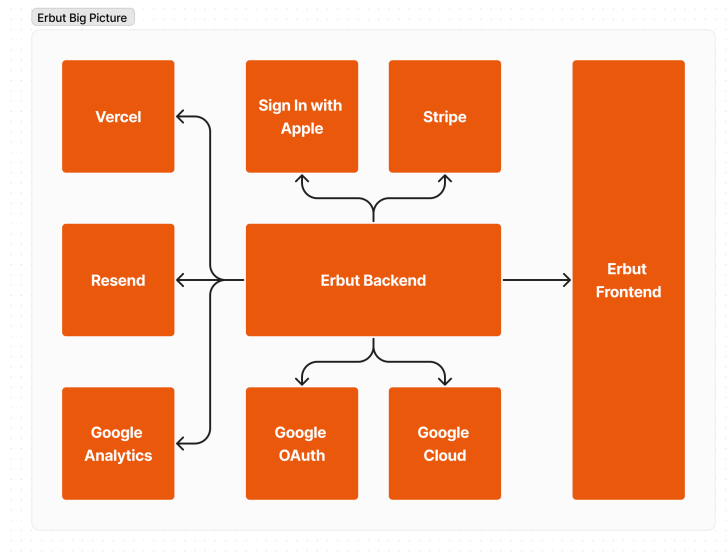
1.1 Big Picture

Figure 1.1: Big Picture Diagram of Erbut

The Big Picture diagram sketch seen in Figure 1.1 provides a high-level overview of Erbut's system architecture, highlighting its backend central to operations, integrating with services like Vercel, Stripe, Resend, Sign in with Apple, Google OAuth, Google Cloud, and Google Analytics.
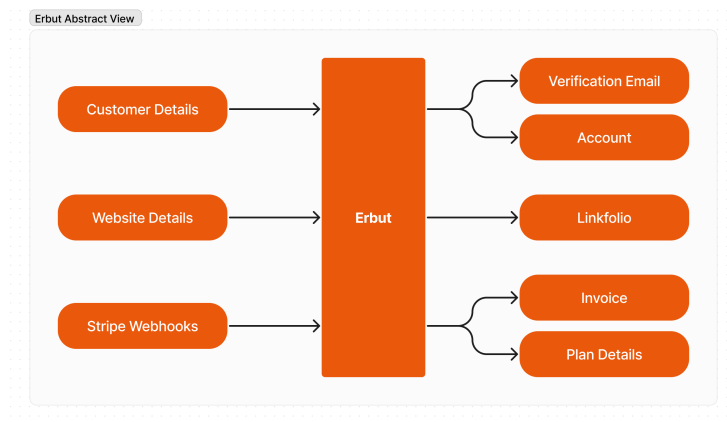
Figure 1.2: Black Box Diagram of the System

1.2 Abstract View

The abstract view diagram seen in Figure 1.2 helps us understand the inputs the system requires and the outputs it will produce. A glance at the diagram shows that we use Customer Details to create an account initially and then to verify the user's identity in subsequent interactions. Website Details are used to construct the Erbut Linkfolio. Additionally, Stripe webhooks provide real-time information about our users' subscription statuses.
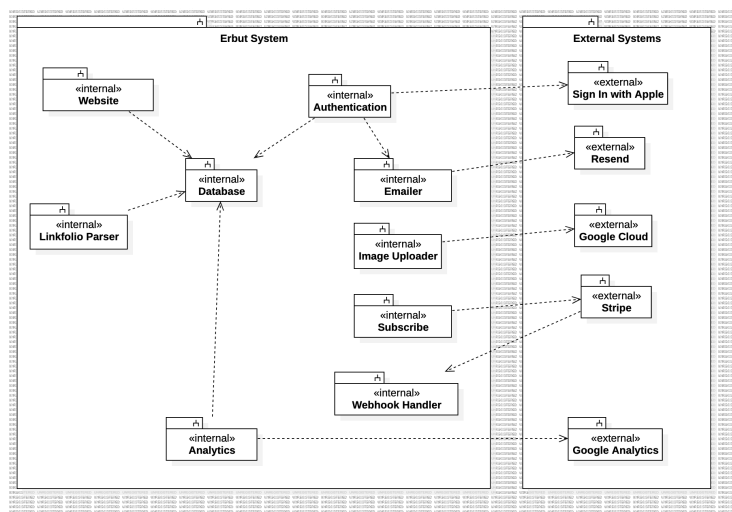
1.3 Subsystem View

Figure 1.3: Package Diagram of Erbut

The subsystem diagram found below in Figure 1.3 represents all the components that our system (Erbut) consists of internally and externally. External

subsystems represent the components we are using directly without building them from scratch. On the other hand, internal subsystems are built by us from the ground up.

Chapter 2

Functionalities of The Subsystems

This section highlights the description of the sub-systems previously presented in the Sub-System View of Erbut.

2.1   Database (`internal`)

Stores the data for the system, such as our customers' details and their website data.

2.2   Authentication (`internal`)

Responsible for the authentication of users whether it's their first time accessing the system or a regular user. Integrates with Google OAuth and Sign in with Apple to offer more authentication options.

2.3   Website (`internal`)

API for the front-end to allow the customer to create and manage websites (Erbuts).

2.4   Linkfolio Parser (`internal`)

It fills out the template of the Erbut based on the data in the database. It knows which Erbut to query by its subdomain. It then parses the template and fills it with data, then finally returns it to the client.

2.5   Emailer (`internal`)

This service is responsible for sending emails. Other systems like Authentication depend on it to send verification and reset password emails. It accepts email and a template id.

2.6 Image Uploader (`internal`)

This service allows customers to upload an image and get a URL for it. This is helpful to put images in their Erbuts by hosting it on our servers.

2.7 Subscribe (`internal`)

This service allows the customer to manage their billing information and subscribe or change their plan.

2.8 Webhook Handler (`internal`)

This service allows Stripe, our payment gateway, to send us updates about our customers' billing state. Whether they are subscribed, paid, did not pay, or even are late on payment.

2.9 Analytics (`internal`)

This service collects and aggregates data from the Erbuts and saves it for our customers to view and take decisions based on it.

2.10 Google Analytics (`external`)

Collects and analyzes data of visitors and their behaviors when visiting a customer's Erbut (Linkfolio).

2.11 Google Cloud (`external`)

Responsible for providing computing resources, data storage, and hosting the system backend services. It's a cloud computing service that runs on the same infrastructure that Google uses for its own products.

2.12 Stripe (`external`)

Responsible for handling all payment transactions, including processing customer credit cards, managing secure payment information, and handling subscription billing services.

2.13 Resend (`external`)

Responsible for automated email communications, such as account verification, notification, and other automated email responses based on user actions or system triggers.

Chapter 3

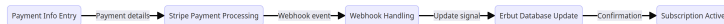# Data Flow of Erbut

3.1 Batch Sequential

Figure 3.1: Batch Sequential of Website Creation

In Figure 3.1 above, the diagram shows how the data in the subscription process of Erbut. Below are the steps explaining how and when the interactions in the diagram happens:

1. The Payment Info is entered and Payment Details are extracted from it.

2. The Payment Details are now an input to the Stripe Payment Processing, which is an external system, and the output is is a webhook event.

3. The webhook event is now an input to the Webhook Handler, which is an internal system of Erbut, and the output is an update signal for the database.

4. The update signal for the database is now an input to the Database, which is an internal system of Erbut, and the output is a confirmation signal.

5. The confirmation signal shows that the subscription is active now.

3.2 Pipe and Filter

In Figure 3.2 below, the diagram shows how the data in the registration process of Erbut flows from one sub system to another. Below are the steps explaining how and when the interactions in the diagram happens:

1. The Input Data pipe reads the email, username, and password from the User filter and then writes this registration data to the Authentication Service filter.

2. Then the Authentication Service filter reads the passed registration data and writes it to the Database, and waits for confirmation.
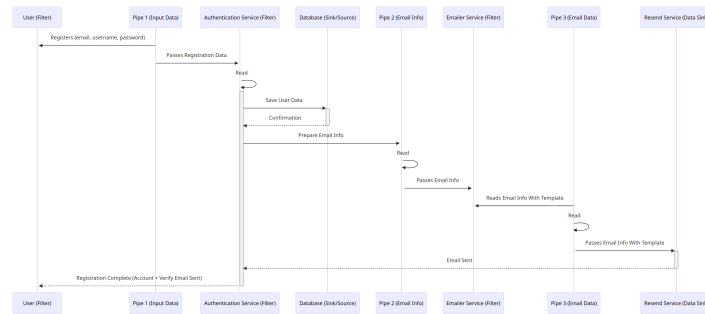
Figure 3.2: Pipe and Filter of User Registration

3. Once the confirmation is received, the Authentication Service filter writes the Email Info to the Email Info pipe.

4. The Email Info pipe reads the email and writes it to the Emailer filter.

5. The Email Data pipe reads the Email Info with Template and writes it to the Resend Service filter, which is our data sink.
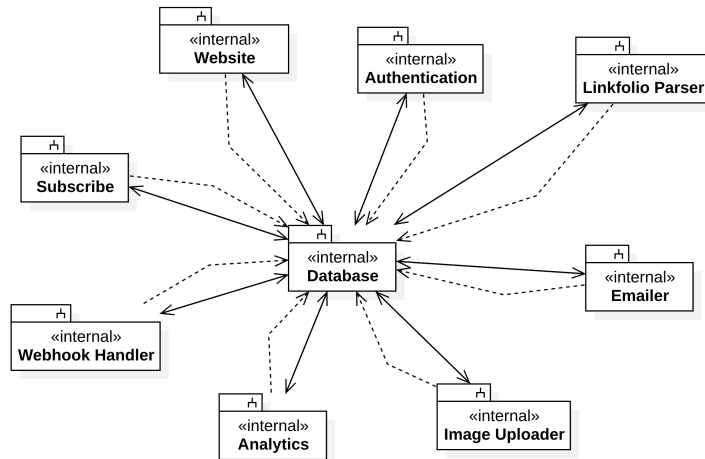
Chapter 4

Repository Diagram

Figure 4.1: Repository Diagram of Erbut

The Repository Diagram of Erbut, shown in Figure 4.1, illustrates the system's centralized data management structure, highlighting how different subsystems interact with the central repository. This visual representation underscores the central database's role as the core repository where all data is stored and accessed, ensuring data consistency and integrity across the system.

1. **Central Database (Repository)**: The primary storage for all data related to users, their linkfolios, subscription information, and analytics, acting as the single source of truth.

2. **Authentication System**: Retrieves and verifies user credentials from the central database, managing secure login processes and session tokens.

3. **Website Management System**: Utilizes the central database to manage data related to user-created linkfolios, including layouts, links, and custom domains.

4. **Subscription Management System**: Manages user subscription details and interfaces with external systems like Stripe to process transactions and update subscription statuses.

5. **Analytics System**: Stores and analyzes data on user interactions with linkfolios, providing insights into performance based on information stored in the central database.

6. **Email Communication System**: Sends automated emails for various user notifications, accessing contact information from the central database to ensure accuracy.

7. **External Integration Points**: Includes connections to services like Stripe and Google Analytics, synchronizing data with the central database for cohesive system functionality.

This diagram emphasizes the significance of the central database in facilitating efficient data flow and interaction between subsystems and external services, ensuring the Erbut platform operates cohesively.

Chapter 5

Architecture

The architecture of Erbut is elegant and scalable, designed to accommodate the platform's dynamic and user-centric functionalities. The system's architecture is based on a layered structure, ensuring a clear separation of concerns and modularity. The layered architecture diagram showcases the system's presentation, application, and data layers, each responsible for specific functionalities.

5.1 Layered Architecture

In the Layered Architecture diagram shown in Figure 5.1, the system is divided into 3 layers, each responsible for specific functionalities. The frontend is the layer that the user interacts with. It is built in React. The frontend communicates with the backend through an API. This API is a gRPC API. The backend is built in Go. The backend communicates with the database, which is a PostgreSQL database. So the communication happens through SQL.

5.2 MVC Architecture

MVC stands for **M**odel **V**iew **C**ontroller. In our system, Erbut, we use MVC on the frontend that is build in React. We use Zustand for state management. The Model is the data that the user interacts with. The View is the UI that the user interacts with. The Controller is the logic that connects the Model and the View.

The MVC architecture is as follows:

- The Controller is the Zustand store.

- The View is the React components.
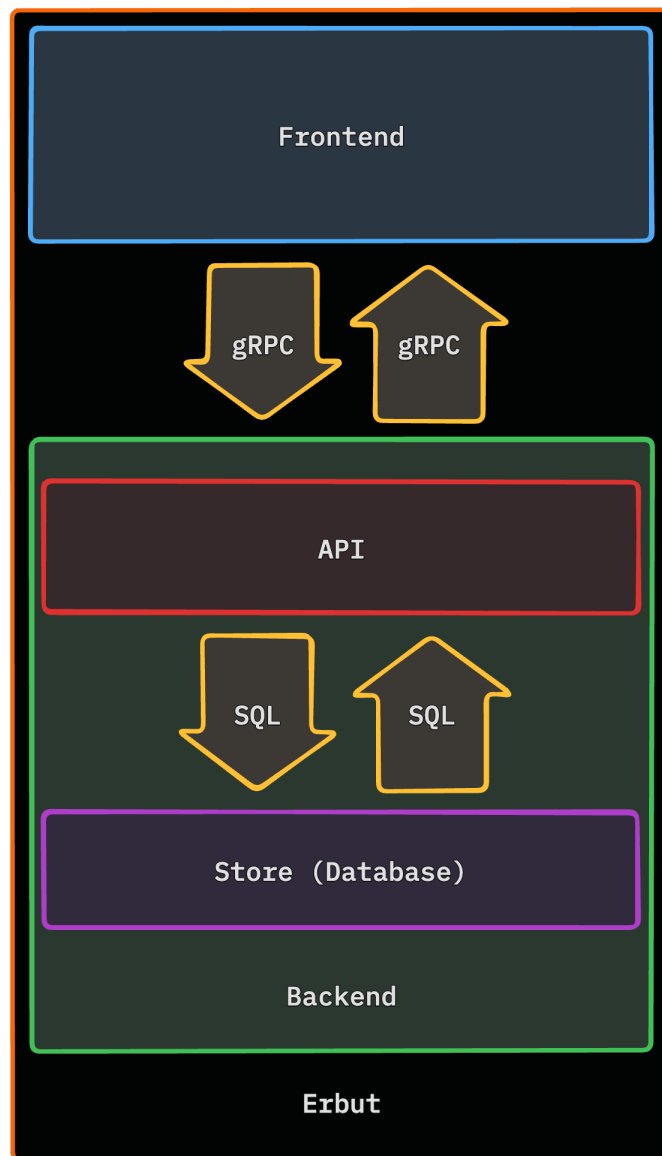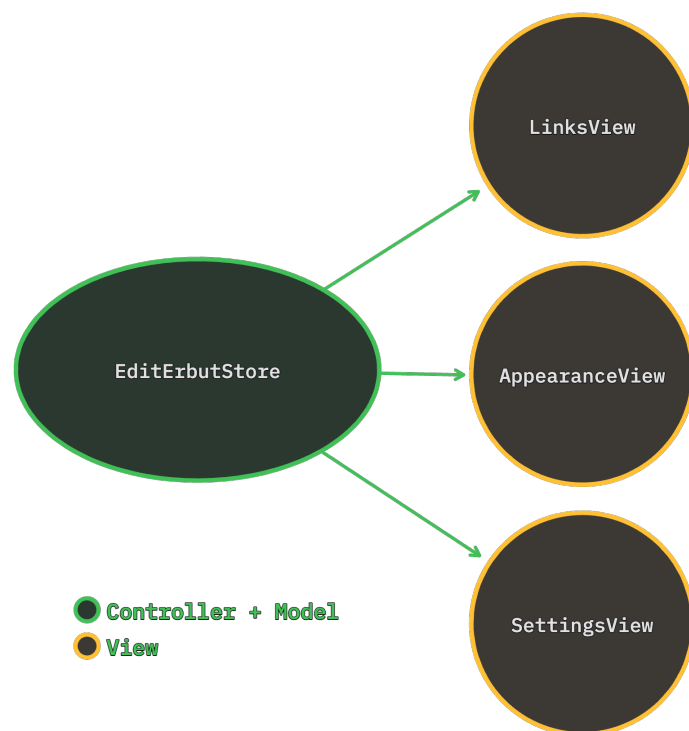
- The Model is the data that the Zustand store holds.

Figure 5.1: Layered Architecture of Erbut

Figure 5.2: MVC Example from Erbut