**BuddyToys**

Online Pet Shop

Architecture Document

Background

This document describes the BuddyToys web application architecture, the young and promising BuddyToys companys' main product.

BuddyToys sells everything from the pet industry - from pets to toys and equipment.

Because of the pandemic, people started isolating themselves, working from home and lowering their social communications. These things lead to many mental problems, like the increased cases of depression - a dangerous medical illness. Here come in help, pets. They can reduce stress levels drastically and help people.

The website will sell pets, toys, and other equipment they might need. The system should be with very low downtime. It should be prepared to meet a high increase in concurrent users during sales and marketing campaigns without failing because this would cost us money and users trust.

This document describes the system architecture of BuddyToys's application.

The architecture comprises technology and modelling decisions that will ensure the final product, assuming the architecture is followed, will be fast, reliable, scalable, and easy to maintain.

The document outlines the thought process for every aspect of the architecture and clearly explains why specific decisions were made.

The development team must follow the architecture depicted in this document closely. In any case of doubt, please consult the Software Architect.

Requirements

Functional requirements

* The system will have authentication / authorization mechanism
* Provide sales and other important data to BI analyzer
* Users should be able to add items to card
* Users should be able to buy items
* Forum
* Users should be able to search clinics
* Users should be able to rate, comment and see info about clinics
* Payments will be made through an external API
* Deliveries will be made through an external API

Non-functional requirements

* The system should be able to scale for hundreds of concurrent users
* The database will grow by 300GB annually
* SLA: The system should be available as much as possible

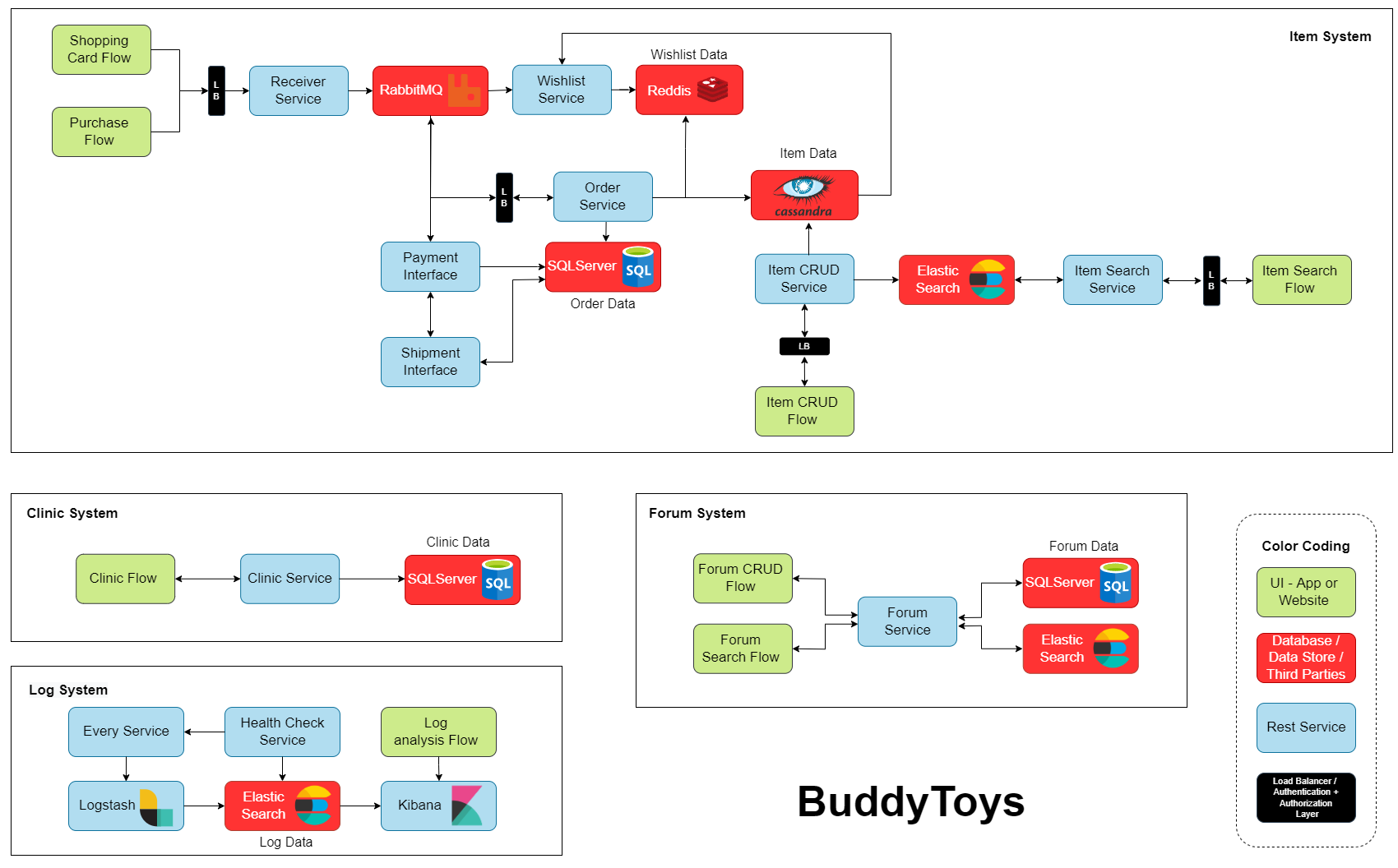
Executive summary

This document describes the new BuddyToys web application architecture, an online pet shop, allowing end-users to buy all kinds of pets and toys for them, bringing them comfort and happiness.

When designing the architecture, a strong emphasis was put on two major features:

* The application should be reliable
* The application should be scalable

The architecture is based on the most up-to-date best practices and methodologies to achieve these features, ensuring high-availability and performance.

Here is a high-level overview of the architecture:

As shown in the diagram above, the application comprises four separate, independent, loosely-coupled services. Each has its own tasks, and communicates with the other services using standard protocols.

All the services are built as stateless services, meaning – no data is lost of the service is suddenly shutting down. The only places for data in the application are the Queue, Redis, and the DataBases, all of them serialize the data to the disk, thus protecting it from shutdown cases.

In conjunction with a modern development platform (.NET Core), this architecture will help create a modern, robust, easy to maintain, and reliable system that can serve the company successfully for years to come and helps it achieve its financial goals.

Overall Architecture

Here is the architecture diagram for the BuddyToys application:

The diagram from 1 to 4 shows a high­level view of the user interaction with the system and the interactions between the technologies involved.

1. Legend
2. Diagram

   Description automatically generatedItem System

That is our product layer. It is the heart of our business. Our profit comes from it, and users come to us for it. Thus we have to make sure that it is scalable and resistible. We need to invest most of our resources into it.

Services

The system comprised of the following services:

* **Receiver** Service:

Receives shopping card and order data and adds them to a Queue for further handling. The receiver puts a strong emphasis on performance.

* **Wishlist** Service:

Validates and adds products to the wishlist. Pulls data from the Queue (where the Receiver Service placed them).

* **Order** Service:

Validates and adds orders. Pulls data from the Queue (where the Receiver Service placed them).

* **Payment** **Interface**:

Payments are made through an external API, so Payment Interface communicates with it.

* **Shipment Interface**:

Shipments are made through an external API, so Shipment Interface communicates with it.

* **Item CRUD** Service:

Processes CRUD Request for Items. Adds every new Item to Casandra Database and Elastic Search Cluster.

* **Item Search** Service:

Sends search queries to Elastic Search and returns the results.

Messaging

The various services communicate with each other using multiple messaging methods. Each method got selected based on the specific requirements of the services. Here are the different messaging methods used in the system:

* **Receiver** Service:

Receives data through HTTP and forwards it to RabbitMQ***.***

* **Wishlist** Service:

Receives data from RabbitMQ and forwards it to Redis.

* **Order** Service:

Receives data from RabbitMQ and sends it to Cassandra Database.

* **Payment** **Interface**:

Receives data from RabbitMQ and sends requests to external API through HTTP.

* **Shipment Interface**:

Receives data through HTTP and sends requests to external API through HTTP.

* **Item CRUD** Service:

Communicates through HTTP Rest API.

* **Item Search** Service:

Receives data through HTTP and sends requests to Elastic Search. Returns results again through HTTP.

1. A picture containing diagram

   Description automatically generatedClinic System

Services

* **Clinic** Service:

Processes CRUD Request for Clinics.

Messaging

* **Clinic** Service:

Communicates through HTTP Rest API.

1. A picture containing diagram

   Description automatically generatedForum System

Services

* **Forum** Service:

Processes CRUD Request for Forum. Adds every new Item to SQLServer Database and Elastic Search Cluster.

Messaging

* **Forum** Service:

Communicates through HTTP Rest API.

1. Diagram

   Description automatically generatedLog System

Services

* **Every** Service:

Not actual service but abstraction over all the other services. It just explains that the other services are expected to send log reports to Logstash.

* **Health Check** Service:

Sends requests to all services and checks if they are alive. For better abstraction, we can have such a service per each system.

Messaging

* **Health Check** Service:

Makes requests through HTTP.

Scaling

This architecture allows efficiently scaling the services as needed. Each service can be scaled independently, either automatically (by service managers such as Kubernetes) or manually.

Also, the service’s inner code is fully stateless, allowing scaling to be performed on a live system without changing any code lines or shutting down the system.

Module drill-down