AnyChange
Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

2. Architectural goals and philosophy

The architecture of AnyChange has to be adaptable to integrate with many different external product selling platforms and has to be scalable to meet the performance criteria of a growing customer base. It should provide a secure way of communicating between the users and application software.

Scalability: The architecture must be scalable to handle the increasing customer base and growing transaction volumes. As the number of users and concurrent requests rises, the system should gracefully handle the load without compromising performance. The scalability solution should ensure that response times remain within acceptable limits and that the system can be easily expanded as the business expands.

Integrity: Data integrity is essential to maintain the accuracy and consistency of the system's data. The architecture should include mechanisms to validate and protect data from unauthorized modifications, corruption, or loss. This includes implementing appropriate data validation checks, employing data encryption and backup strategies, and ensuring the integrity of data transfers between components**.**

Maintainability: The architecture should be designed and implemented with maintainability in mind, facilitating efficient development and future enhancements. Consider the following factors:

Modularity: Design the system with clear modular boundaries, allowing for independent development, testing, and maintenance of different components. This promotes code reusability and makes it easier to understand and modify specific parts of the system.

Documentation: Provide comprehensive documentation that describes the architecture, design patterns, and component interactions. This documentation should also include guidelines and best practices for developers to follow, ensuring consistency and ease of maintenance.

Code Maintainability: Emphasize clean code practices, such as following coding conventions, writing self-explanatory code, and properly organizing the codebase. Employ automated testing, continuous integration, and code review processes to catch and address issues early, reducing technical debt and enhancing maintainability.

Version Control: Utilize a robust version control system (e.g., Git) to track changes, manage branches, and facilitate collaboration among developers. Use branching and tagging strategies to support parallel development efforts and enable easy rollback or deployment of specific versions.

Monitoring and Logging: Implement logging and monitoring mechanisms to capture system events, errors, and performance metrics. These tools provide valuable insights into the system's behavior, aiding in identifying and resolving maintenance issues promptly.

3. Assumptions and dependencies

The team is composed of developers with varying experience. The project will be built from the ground up so it is assumed that there won’t be any legacy interfaces that will be depended on. AnyChange will be a web-based software product. It is assumed that there won’t be any publicly available APIs to pull price data from when collecting product data, if there is, however, the system should be able to use that as well.

4. Architecturally significant requirements

- All data, hence the database applications and clients are expected to be secure.
- Functional uptime of the system will be 166 hours per week.
- System shall scale up to support 10000 concurrent transactions.

5. Decisions, constraints, and justifications

- In order to achieve the flexibility required to accommodate several different platforms with the assumption that there are no publicly available APIs to pull price data from, AnyChange will use an external scripting platform that can scrape data from price sources. This will enable us to tailor custom solutions for each price source so that we can monitor a wide category of products and sources.
- AnyChange will be built using Model-View-Controller architectural pattern. It is a pattern readily supported by the Spring Boot framework and allows for the separation of data, business logic, and presentation of data. Spring Boot Framework is chosen because the development team is already experienced with it.
- AnyChange will be developed using Java to enable cross-platform operation.
- AnyChange will use Python as is external scripting platform that will be used for web scraping. Python is chosen because it is easy to use and has widely used high-quality libraries we can use.
- All communications between users and the AnyChange platform will be encrypted using HTTPS. HTTPS is an extension of the HTTP protocol that provides secure communication over the internet.
- All critical data such as credit card information, and user passwords will be encrypted.
- Any action on the user's account will be authenticated.
- User's will be authorized based on their type(normal user, seller user, and admin) to prevent unwanted actions.
- Some actions like changing passwords will require informing the user by a separate channel like e-mail to prevent malicious use of actions.
- To achieve the uptime requirement, an active redundancy method will be used. The system will be built using distributed services where a load balancer is used to keep service operation in case one of the services are down for maintenance or other reasons.
- To meet the required transaction demand, the system will make use of concurrent processing where possible.

6. Architectural Mechanisms

Scripting Platform

This mechanism is intended to provide fast to develop, flexible platform to develop web scraping solutions assuming the price sources do not provide free, easy-to-access API's to pull price data for products.

Inversion of Control

Inversion of Control is a pattern provided by Spring Framework, it is used to inject dependencies on a generic program to increase modularity and extensibility.

Availability

The system should be reachable and in working condition to be considered available. To meet the required availability, the system should be available during software updates, in the event of a crash of a component and it should operate as expected.

Security

The system requires the storage of security-critical data such as credit card information and passwords. These should be encrypted against possible attacks against the system. Also since this is a

web-based application all the communication will be done over public networks. Therefore all communications should be encrypted against eavesdropping and all actions must be authenticated to prevent man-in-the-middle attacks.

7. Key abstractions

The system will be composed of products and users. Each user will be authenticated to use the system and users will be able to monitor products that are added to the system. The products can have many different price sources. Each price source is responsible for providing price points at any point in time. A product and price source will have price information, which includes the product's current price and price history.

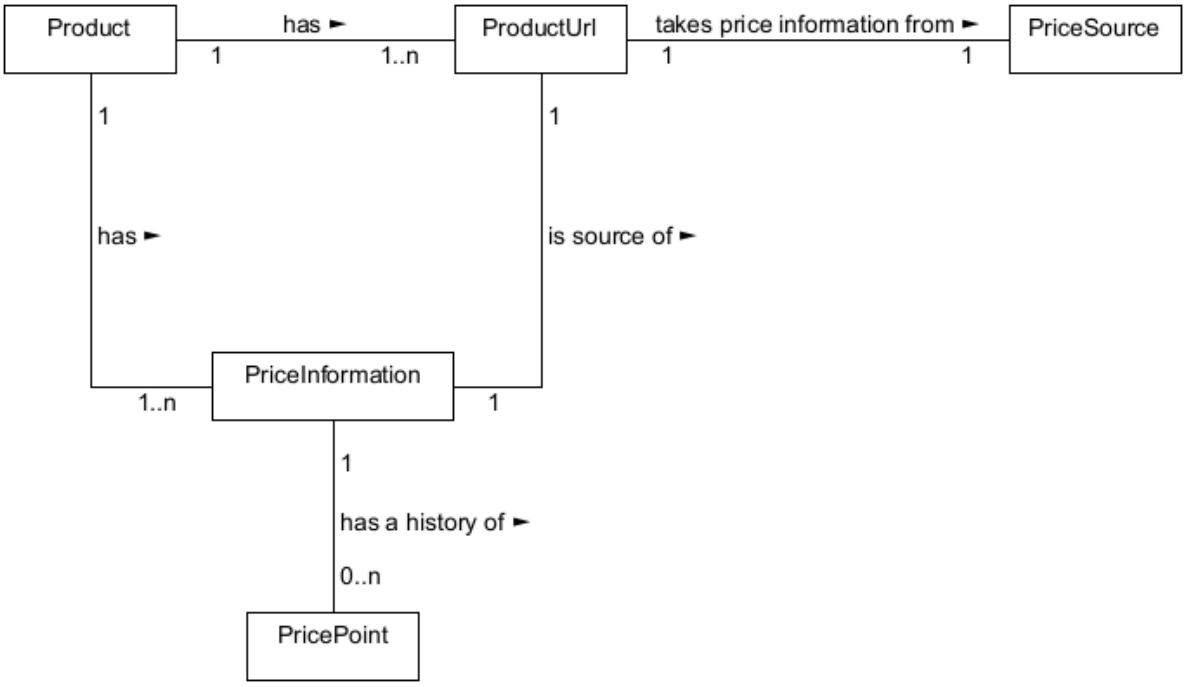

Figure 1. Key objects in AnyChange's abstraction of products

8. Layers or architectural framework

When using MVC pattern, the product and users will form the model part of the architecture. The controller part will be provided by the software and it will be responsible for reacting to user events and doing necessary business logic that will result in data that can be presented to user. The view will be made to represent the data produced by the controller.

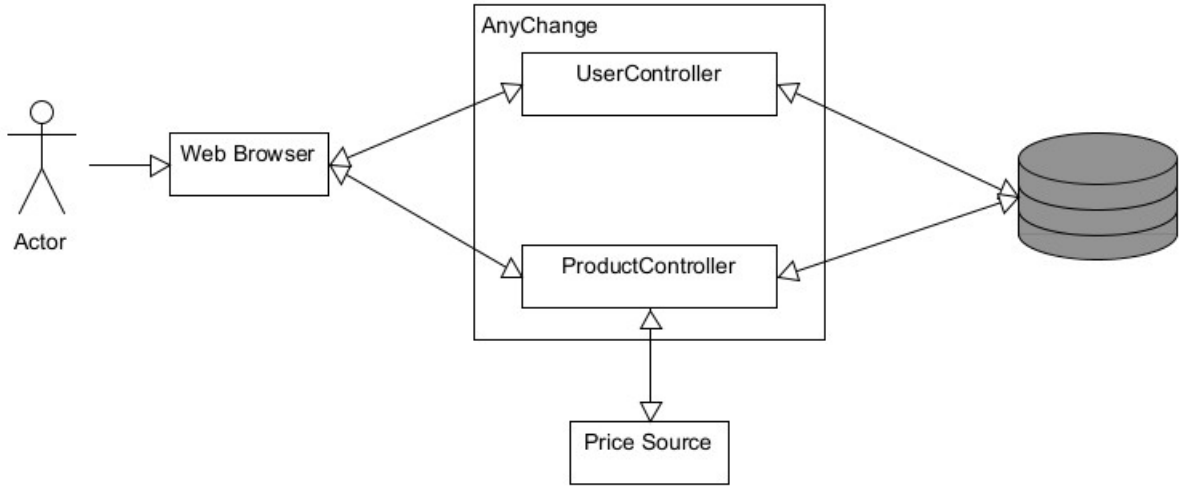

Figure 2. AnyChange MVC implementation, HTML provides view, UserController and ProductController classes provide controllers and models are present in persistence layer

9. Architectural views

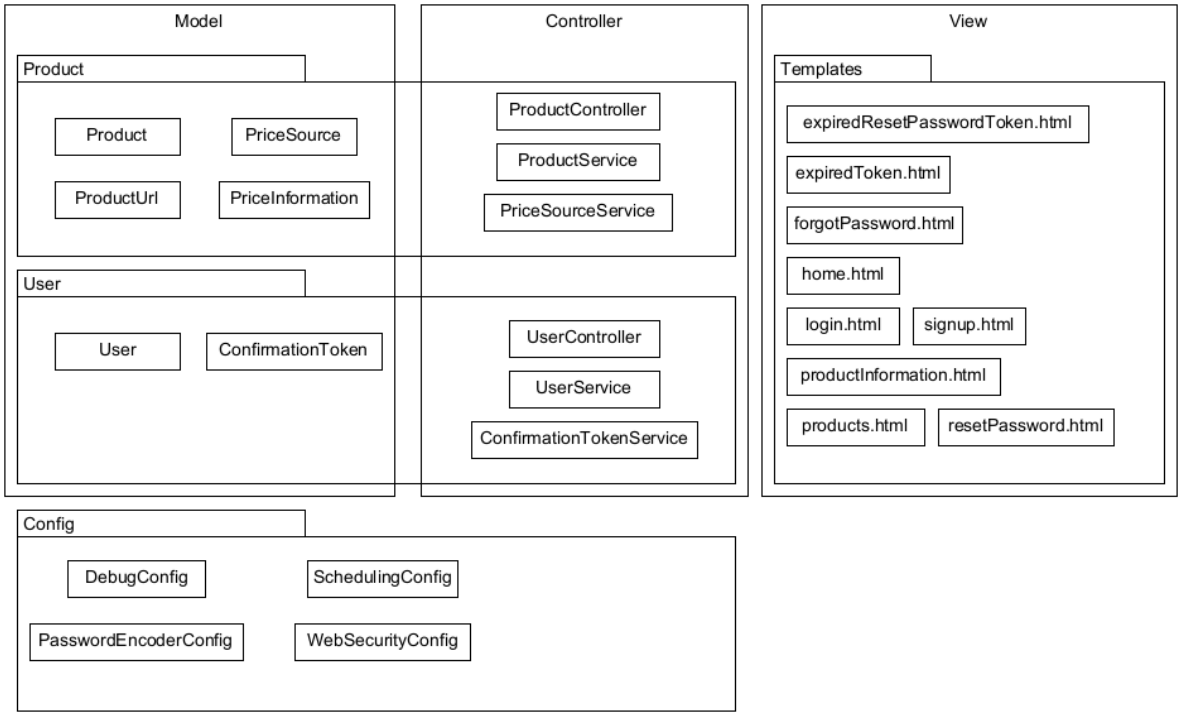

Figure 3. Package Diagram

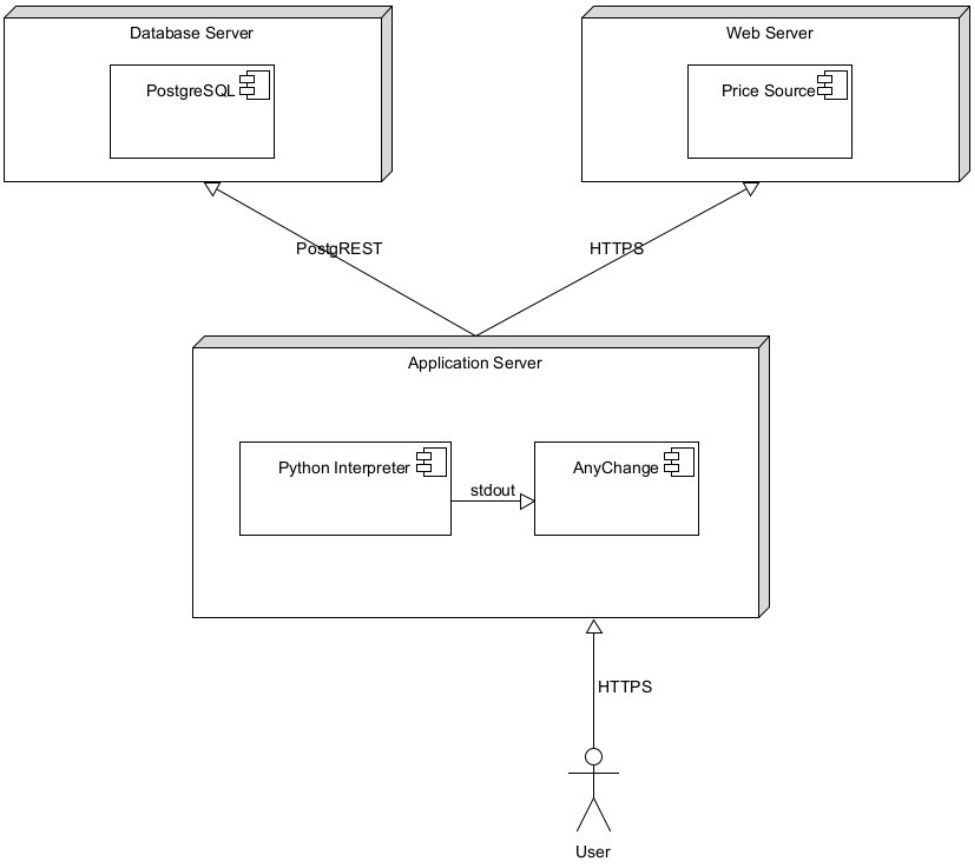

Figure 4. Deployment View