RecyPy: Architecture Design Document

Last Modified: April 21, 2021

Table of Contents

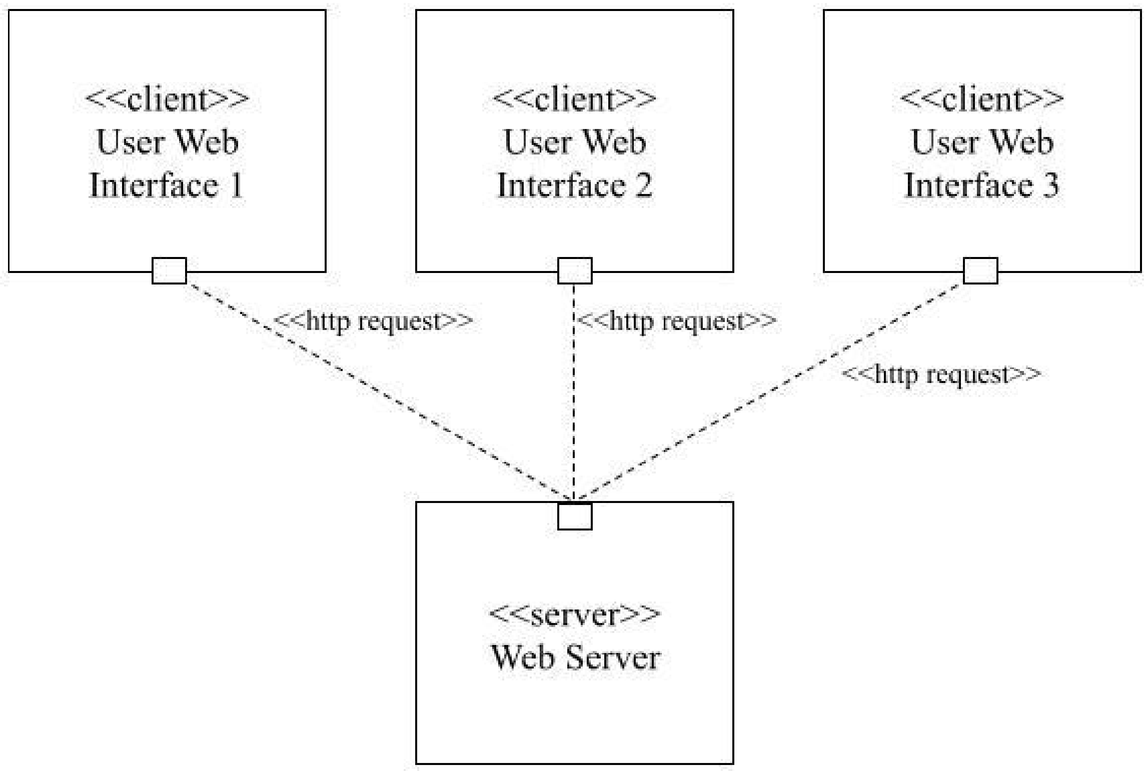
Introduction

RecyPy is a web application used for sharing recipes. Similar to a cross between a social media app and a recipe database, in this system users will be able to create an account. Using this account, they will be able to post their own recipes, as well as “follow” other users or “star” their recipes. This creates a personalized experience for each user, as well as a foundation for further implementations, such as a rating system for individual recipes and the ability to categorize them. Users will be able to view a feed of recipes from the people they follow, as well as search new recipes in the search bar.

This document provides an overview of the system architecture at a few different levels of specificity. First, it provides a high level view of the whole system, followed by diagrams breaking it down into two components at two different scales.

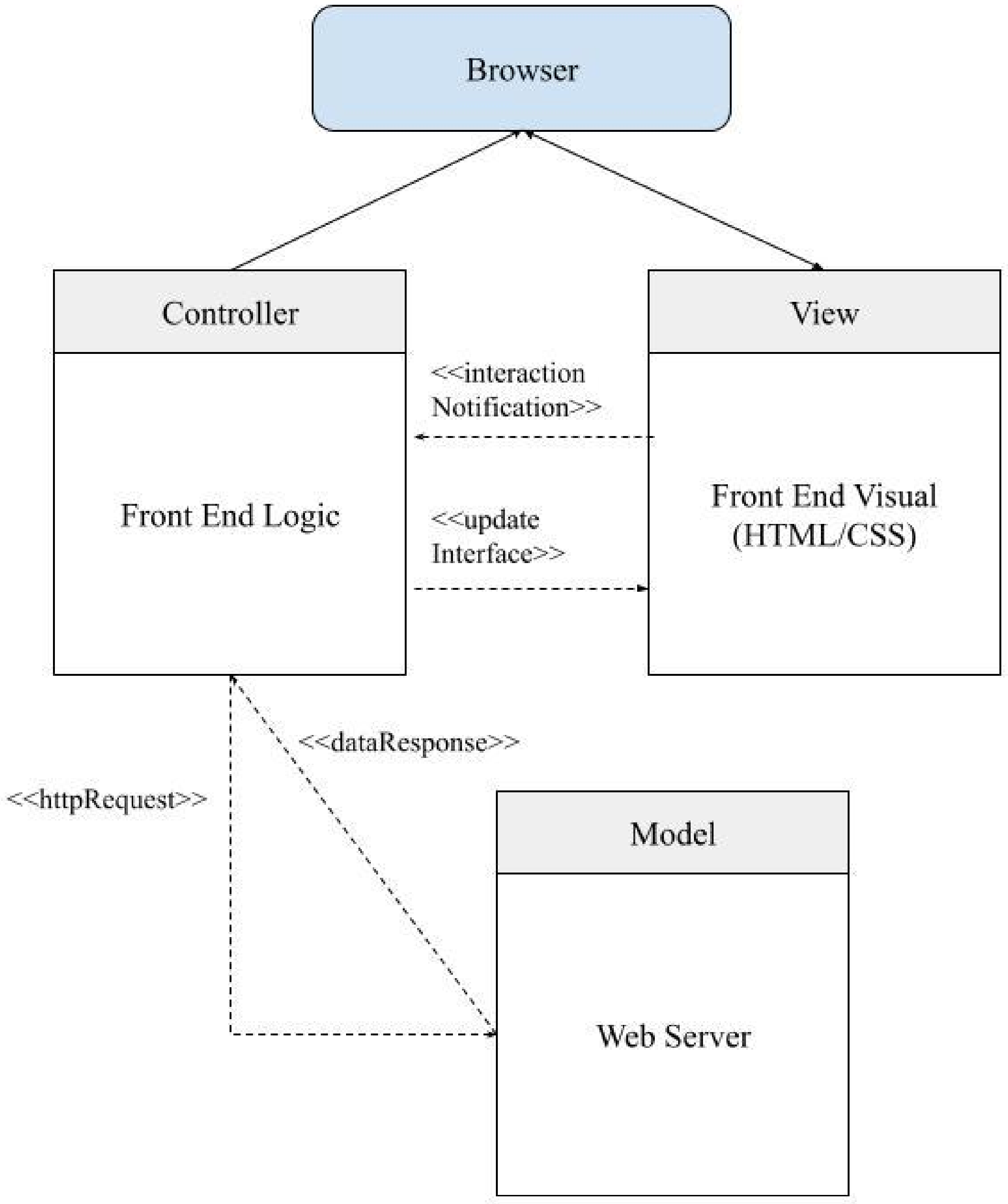
View 1: System Level

For the system as a whole, we use a client-server architecture, as is fairly standard for web applications. Data must be stored in a centralized database on the server in order for all clients to be able to interact with the same set of recipes. This high level diagram is as follows:


<<client>> User Web Interface 1
<<client>> User Web Interface 2
<<client>> User Web Interface 3
<<http request>>
<<http request>>
<<http request>>
<<server>> Web Server

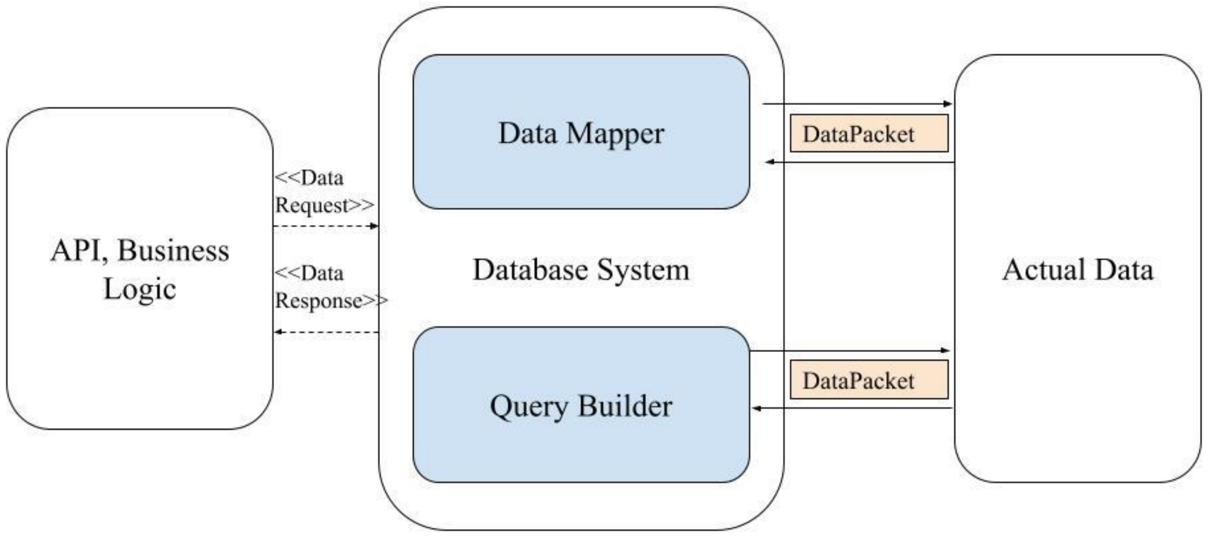
View 2: User Web Interface

For the user facing website, we use a simple model-view-controller architecture, as is fairly standard for web applications. The view is simply the CSS/HTML served to the user, while the controller is generated JavaScript. We call this a simple model-view-controller because the "controller" portion of the architecture is really the web server, which is somewhat external to this component. However, for clarity of the diagram, it is included as part of this component. A diagram of the structure is shown below:



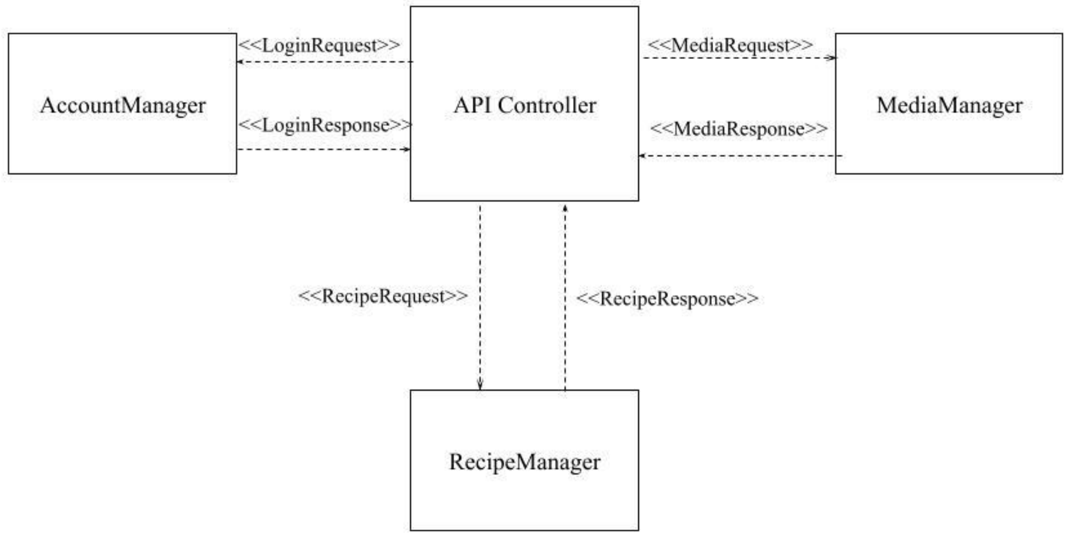
View 3: Web Server

Focusing in further on the "web server" from view 1, we use a repository architecture. The server will need to store data about recipes in a persistent manner, and it is worthwhile to separate logic out from the data. In more concrete terms, this takes the form of separating the web server into an API, made using PHP, and a database, using MySQL. This is depicted as:



View 4: API, Business Logic

For the API, we use an event-driven architecture. This makes sense because the API will need to respond to different requests, i.e. different events, as they come. It may also be dependent on state, since the API will respond differently to an event depending on the data it is supplied or that is requested. If there are no recipes in the system, it will need to deal with a recipe request differently than normal. A diagram of this structure is shown below:


AccountManager
<<LoginRequest>>
<<LoginResponse>>
API Controller
<<MediaRequest>>
<<MediaResponse>>
MediaManager
<<RecipeRequest>>
<<RecipeResponse>>
RecipeManager