Ev4 Developer Guide (Group 10)

I. Frontend
II. Backend
III. Database and schema
IV. Scraping
V. Configuration
VI. Our app and its functionalities

I. Frontend

The front end was constructed through a combination of html, css and vue.js. Vue.js is a javascript framework created by Evan You in 2014 and it connects really well to Django through the RESTAPI framework. “base.html” is where the script js modules and css were imported to all html files.

II. Backend

The back end (our apis, views, viewsets) was created through Django. Django is a python-based free and open-source wfeb framework created by the Django Software Foundation in 2005. Django follows the MVT (model-view-template) architecture.

III. Database

The database was created through SQL and stored on a postgres app under pgAdmin 4. Our database schemas are structured in such a way that large schemas with attributes are created and they are linked together by smaller relational schemas. Examples of large schemas we have would be formulas, skus, ingredients, and goals. These schemas would be linked together by smaller schemas if there is a many-to-one relationship. For example, many ingredients can be linked to one formula, therefore a smaller ingredients_to_formula schema is created to link one ingredient to one formula; if there are 5 ingredients linked to that one particular formula, 5 such schemas would be created. Instances of this can also be seen when multiple skus are linked to one goal. If there is simply a one-to-one relationship such as one formula linked to one sku then such a relationship would simply be handled within the larger schema. For example, the sku schema has a formula attribute that is referenced by foreign key constraints to a particular formula id.

IV. Scraping

The scraping is done using a combination of scrapy and billiard. Scrapy is an open source web crawling library that politely crawls data from given urls and billiard is a framework for asynchronous tasks management in python. As Django is inherently synchronous, we need a separate framework for running any heavy process parallel to our main UI thread. We planned to scrape all sales data every time there are changes to SKU’s as well as every morning. To ensure only the minimum number of requests are made to the website, we only request for sales data related to added SKUs. In other words, we keep track of the changes to the SKU table and update only fields as needed. Every morning we planned to completely wipe out the sales-related data and issue a complete re-crawling of the website. Due to time constraint, we

eventually implemented the crawling process slightly differently. We place a ‘scraping’ button on top of the sales report website, and rely on user to choose to start crawling sales data.

V. Configuration

In order to access the development/build environment, open terminal, git clone https://github.com/YuansongFeng/HypoMeals then enter “pip install -r requirements.txt”. After installing the requirements, enter “python manage.py runserver” then simply copy and paste “http://127.0.0.1:8000/” to enter development environment.

VI. Scheduler

The scheduler is done based on vis library with the Timeline gadget. We visualize everything based on the query results from backend database, including manufacturing activities, manufacturing lines, etc. Any time we add a manufacturing goal or modify an existing manufacturing activity, we make a POST request to the backend to update the related activity. For the backend, most information are tracked in manufacturing activity model and the rest information are in sku, manufacturing goal and goal models. If there are any updates to the model fields related to the manufacturing activity, e.g., manufacturing rates of skus, desired quantity of skus for a goal, we update the manufacturing activity model along with the target model. More information are tracked in https://github.com/YuansongFeng/HypoMeals/issues/33.

In terms of the automation algorithm, we use a greedy algorithm that offers a reasonably good solution. We schedule the activity with the earliest deadline first and in the case of the same deadline, we schedule the one with shorter timespan first. We choose the manufacturing line with the earliest possible availability to produce a specific active activity.

VII. Our App

Our app was deployed on a gunicorn server using python as the primary language and django as the web framework. The front end is a mix of vue, django templating, jquery, and vanilla javascript. We connect the Django with the front end through Django rest api. We build a set of powerful middleware to take care of the data logic and pass the fetched data to front end directly. By handling the complex logic in the backend, we do not put any heavy lifting on client side.

Our front end includes various functionalities, including viewing, updating and creating data in multiple models. We allow import and export of data in various formats. Further, we support a state-of-art timeline feature that allows user to drag and drop manufacturing goals onto the time schedule. We keep a very clean and intuitive user interface to allow easy user interaction. On every web page we dedicate a section to provide error and warning feedback to user.

The following versions of our systems were used:
Python = 3.7

Django = 2.1.5
PyHamcrest =1.9.0