Simpply: Simple Supply

Software Design Specification

Version 1.1

12-16-2014

Contributors:

Michael Yeaple

Jeremy Shulman

Curtis Burtner

Mustafa Al-Salihi

**Document History**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Revision Date | Description of Change | Author |
| 1 | 11/20/2014 | First Draft | Jeremy |
| 1.1 | 12/16/2014 | Addition of iteration 2 designs | Jeremy |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. Introduction](#h.bppo2iroxyqw)

[1.1 Purpose](#h.kg6doymlylkh)

[1.2 Overview](#h.rpgzwzi36v8d)

[2. System Architecture](#h.v61wgbonarac)

[2.1 Architectural Design](#h.dhyhwf6sxx10)

[2.2 Decomposition Description](#h.vrvwdnrf8kil)

[2.3 Design Rationale](#h.ch6iivit8gb6)

[3. Web Application](#h.ur8ilz24hsud)

[3.1 Implementation Design](#h.39rj8koyemml)

[3.2 Rationale and Breakdown](#h.2odey5j0plni)

[4. Engine](#h.2fzjd3l0ukfx)

[4.1 Implementation Design](#h.pmsj2prbtnxh)

[4.2 Rationale and Breakdown](#h.62qf1lo6d9nq)

[5. Customize Item Explorer](#h.gmseoweuc7ey)

[5.1 Implementation Design](#h.r75xyypecdza)

[5.2 Rationale and Breakdown](#h.6ttp8llat3s5)

[6. Algorithms](#h.ok3y0346dcf1)

[6.1 Sequence Diagram](#h.f25kooqyap7p)

[6.2 Rationale and Breakdown](#h.m5kvpkbjuqz8)

[7. Database](#h.3vb65qhlde6i)

[7.1 Implementation Design](#h.ic8wymp0m8a0)

[7.2 Rationale and Breakdown](#h.dvizdoms8c4v)

# 1. Introduction

## 1.1 Purpose

This document will have all of the implementation designs and rationales for why the system was designed in the manner it was designed. This document will help new developers spin up on the project and remind current developers why certain design decisions were made.

## 1.2 Overview

The system architecture section of this document will contain the bird’s eye view of the architecture for the project. This was a simple architecture that was done up front to guide the design for all features of the application. This follows our methodology’s core concepts for simplicity and to not have big design up front.

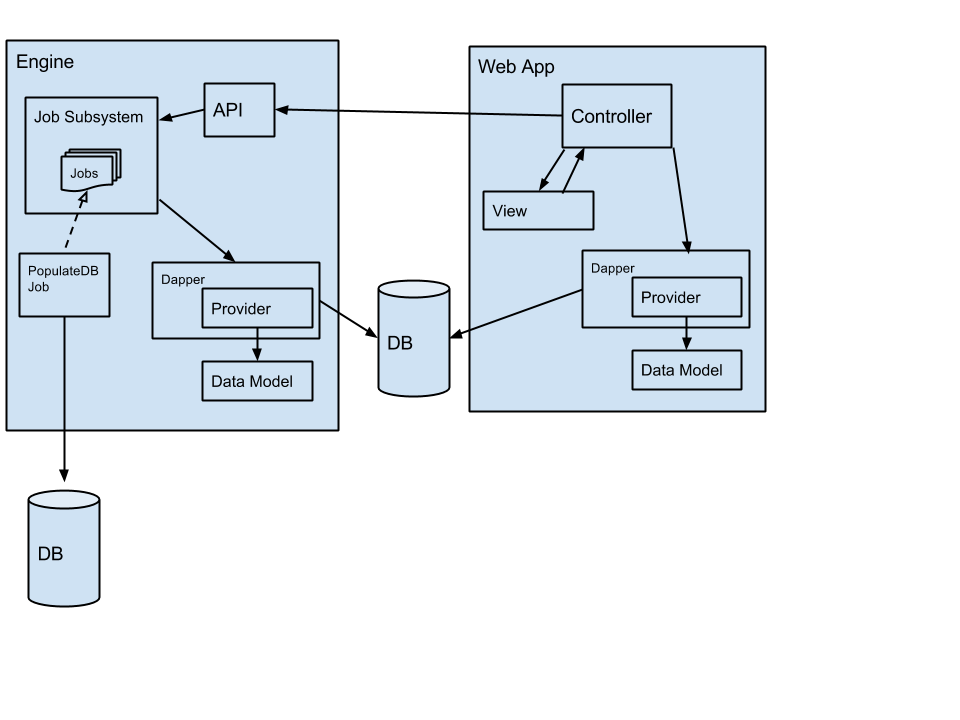
The other sections in this document will be different components or modules that are used in our project. This document will act as the one stop shop for design rationale and any potential future work for these components.

# 

# 

# 2. System Architecture

## 2.1 Architectural Design



## 2.2 Decomposition Description

The three main pieces of our architecture are the web application, the engine, and our database. We split it up this way because the web application should be able to be used on its own without the engine. The engine is going to be running the computations required for determining each item’s store distribution. The web application is going to be used for viewing the calculated distributions and saving the finalized distributions after they have been approved by a merchant analyst.

## 2.3 Design Rationale

For the web application we went with a MVC approach. One of the main reasons for this is that Wegmans commonly uses MVC for their applications so designing it in this way will allow them to more easily integrate our application into their code base as we’ll be following similar conventions. We also decided to use Dapper as an ORM for accessing the database. We will have stored procedures in the database that will be called from our provider layer. This is done to reduce the possibility of SQL injections and to make it easier for the developers to work with data in the database.

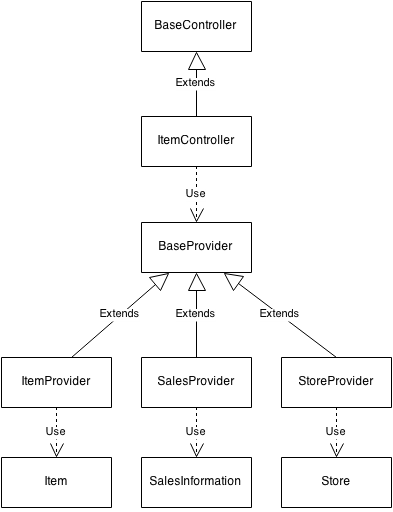
For the engine, we wanted to be able to kick off events with a “fire and forget” type pattern. Because of this, we went with a job subsystem. Each job will complete one action that is the responsibility of the engine. It will be done as a transaction. This way we do not have to worry about the web application hanging while the engine calculates. We will have a simple API in order to interact with this job subsystem. The engine will also use Dapper for database interactions for the reasons stated above.

# 

# 

# 3. Web Application

## 3.1 Implementation Design



## 3.2 Rationale and Breakdown

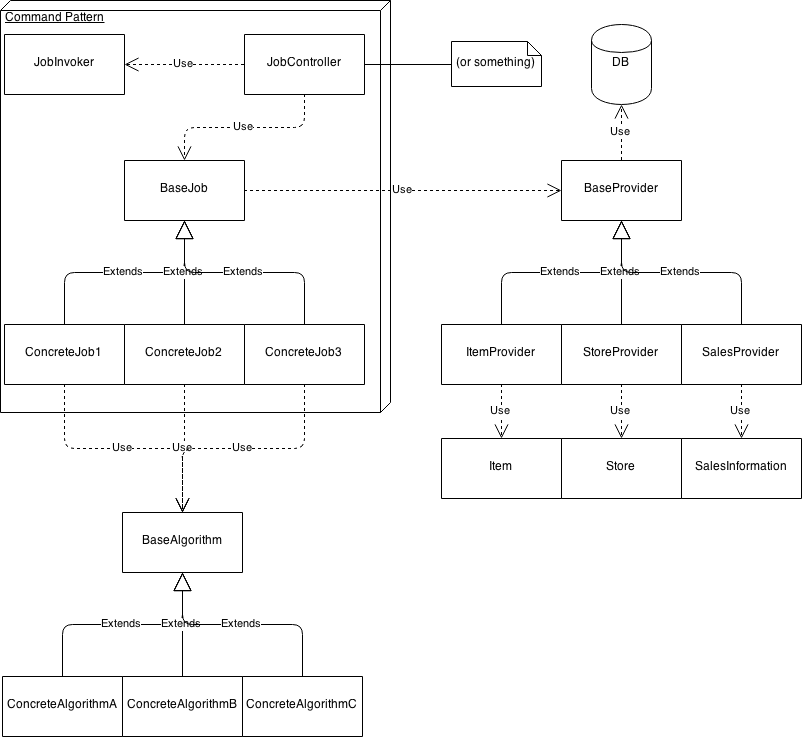
The web application is broken up using the commonly used MVC framework. We split the model up to better match our high level architecture into providers and data models. This lets us reuse the data access layer in the engine helping us adhere to the DRY principle. It also helps us encapsulate the ORM we’re using.

# 

# 

# 4. Engine

## 4.1 Implementation Design



## 4.2 Rationale and Breakdown

The engine is broken up into a few different parts. There is a web API so that the web application can kick off jobs, a command pattern tasked with the scheduling and execution of jobs, and the data access providers that are reused from the web application.

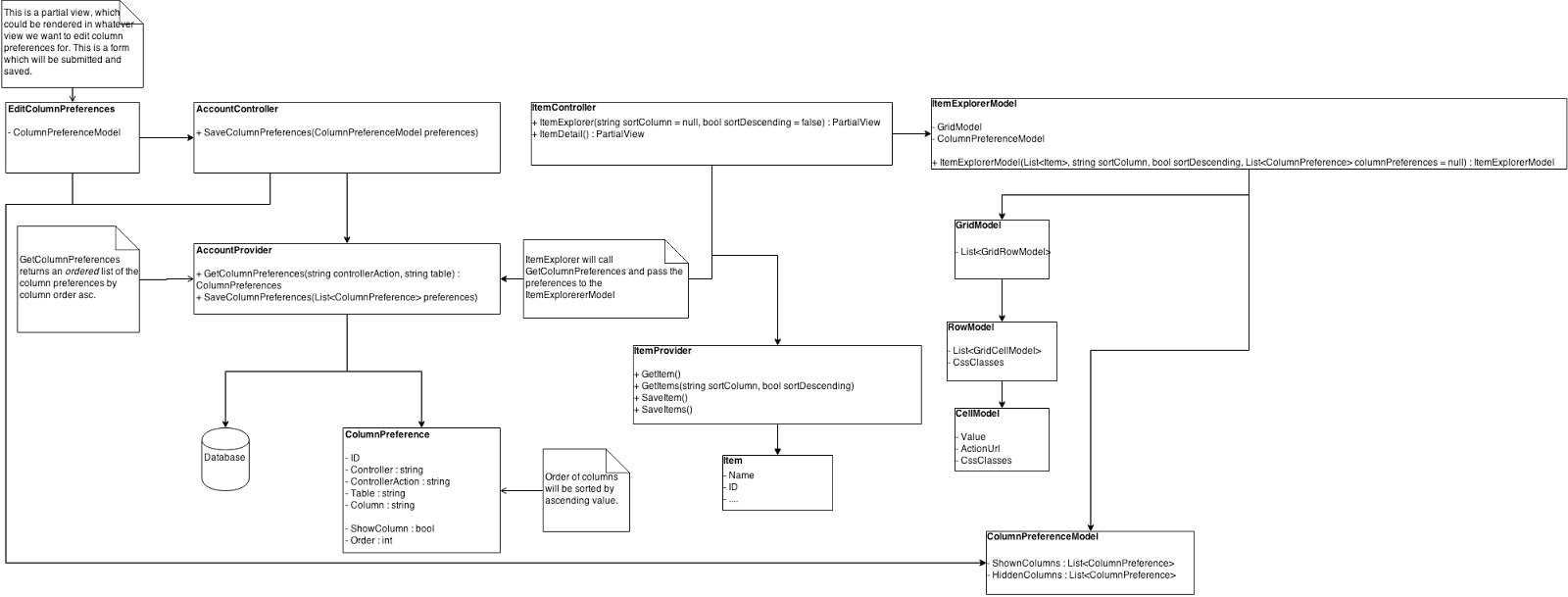
We wanted to decouple the computation from the presentation of the distribution information which is why the engine is separate from the web application, but we also wanted to have the web application be able to ensure it had up-to-date information. The solution we came up with was to allow the web application to start jobs. The jobs would log their progress in the database which would allow the web application to poll for the progress but still keep our separation of concerns. There is no communication from the engine to the web application other than to give a success response when a job starts.

# 

# 

# 5. Customize Item Explorer

## 5.1 Implementation Design



## 5.2 Rationale and Breakdown

This design was for being able to save the order of columns and which columns were hidden/visible. We wanted to be able to do this during a session, but more importantly, we wanted it to persist per user over many sessions. This required a structure that enabled us to store this configuration in the database. We wanted to make this configuration generic enough that it could apply to many tables in our application and not just the item explorer.

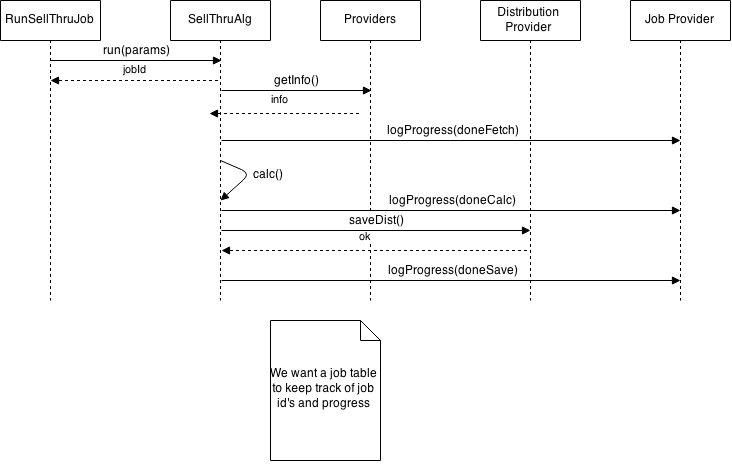
Since user accounts are out of scope for this project, this is currently implemented such that any changes will be seen by all users of this system. In the future, a ColumnPreference object will have a reference to a user id.

# 

# 

# 6. Algorithms

## 6.1 Sequence Diagram



## 6.2 Rationale and Breakdown

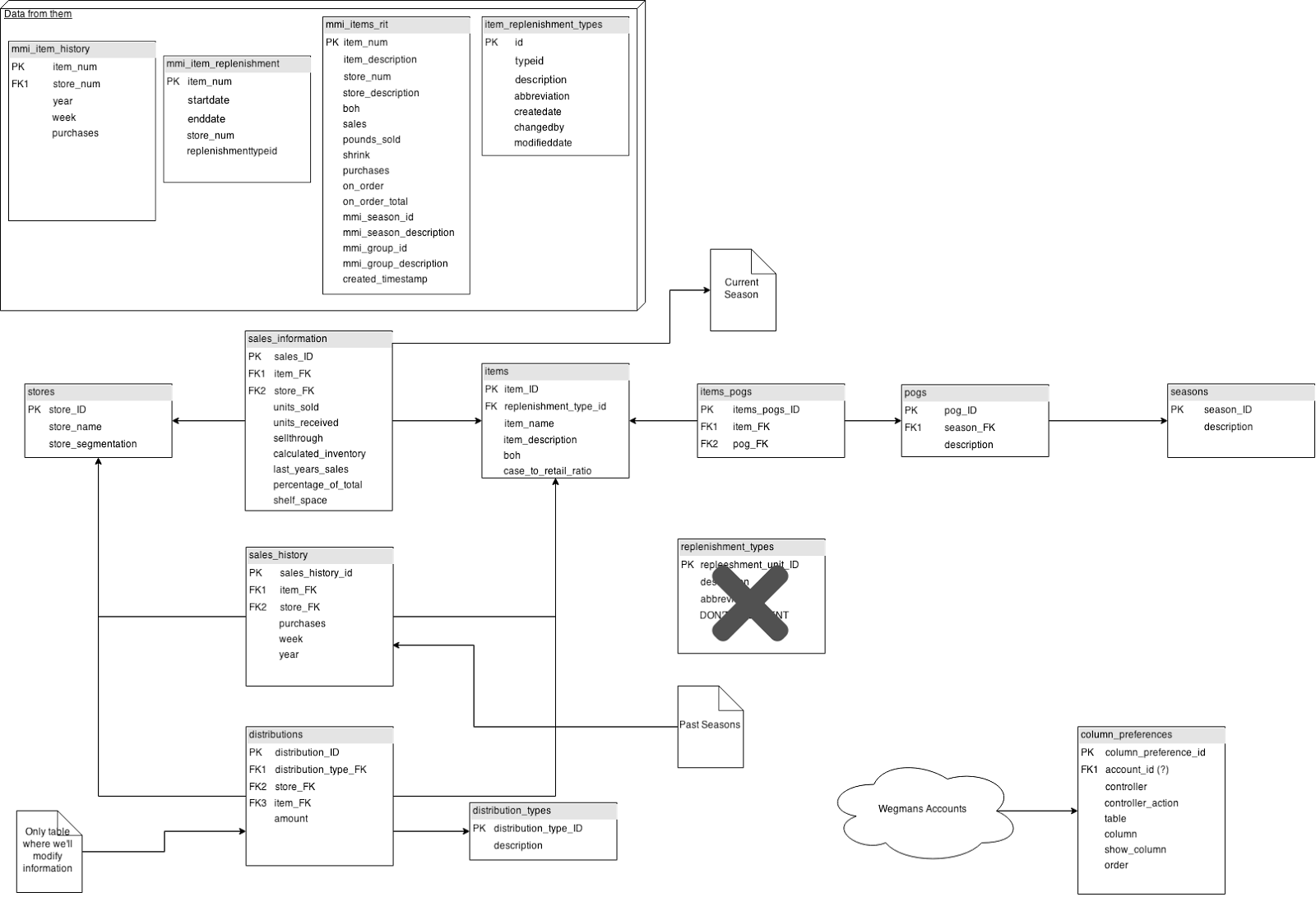
We wanted an outline of exactly how a job would log its progress in the database specifically for the calculation of an algorithm. The resulting sequence diagram follows the design of the engine as expected.

When the web application is polling for job status, we wanted the result to be as close to a concrete percent as possible. Brainstorming how to do this, we found that it would be difficult to estimate that. Instead we came up with a solution that provides useful status information even though it isn’t concrete. The job will update it’s progress at least four times. It will begin by logging that the job has started through the use of a job table. In the future, this table will be useful for seeing if jobs conflict with each other. After getting any required information from the providers it will log its progress again that it gathered the required information. During the calculation and after the calculation is complete, the status will be logged with an estimate of how much of the data has been parsed/calculated. Finally, the job will log when it is done saving any distribution results.

# 

# 7. Database

## 7.1 Implementation Design



## 7.2 Rationale and Breakdown

We knew the format of information that we were getting from the Wegmans team but we wanted to normalize it and have our own representation to suit the needs of our application. We did this to make data retrieval easier and to logically assist anyone who needed to look at the data in our database.