# TRNDii Iteration 3 Summary

## Team members

Jacqueline Luo 26938949
Michael Mescheder 27202202
Sam Alexander Moosavi 27185731
Eric Payette 27008058
Rameen Rastan-Vadiveloo 27191863
Jason Tsalikis 25892120

## Project summary

TRNDii is a group buying website focused on innovative new products. TRNDii has ‘ii tokens’ which users can gain and use. When they want to purchase something, unlike regular online shopping, users will pay to commit to buying a product. After paying, users may spend any amount of ii tokens. Once a predetermined number of committed buyers is reached, the product will then be bought from suppliers at a bulk, discounted price. The savings generated from the bulk discount will be redistributed to the buyers based on how many ii tokens they spent and how many ii tokens were spent in total. Anyone who chooses to not spend ii tokens have a chance at winning a free product.

## Velocity

In our third iteration, we focused mainly on features related to the user purchasing workflow. In this 2 week period, we completed 7 user stories worth a total of 16.5 story points, giving us a velocity of 13.5 points/iteration.

## Overall Arch and Class diagram

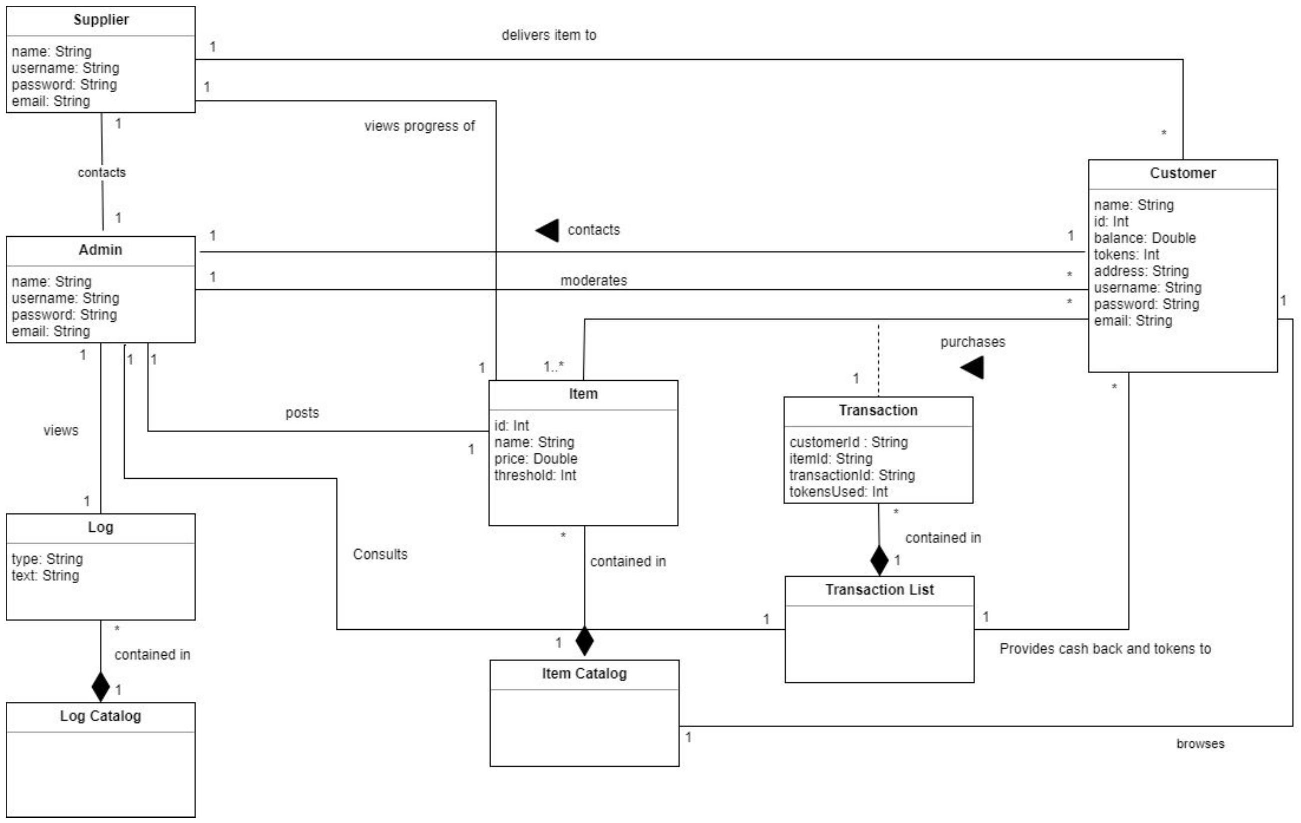

Diagram 1: Conceptual Class Diagram

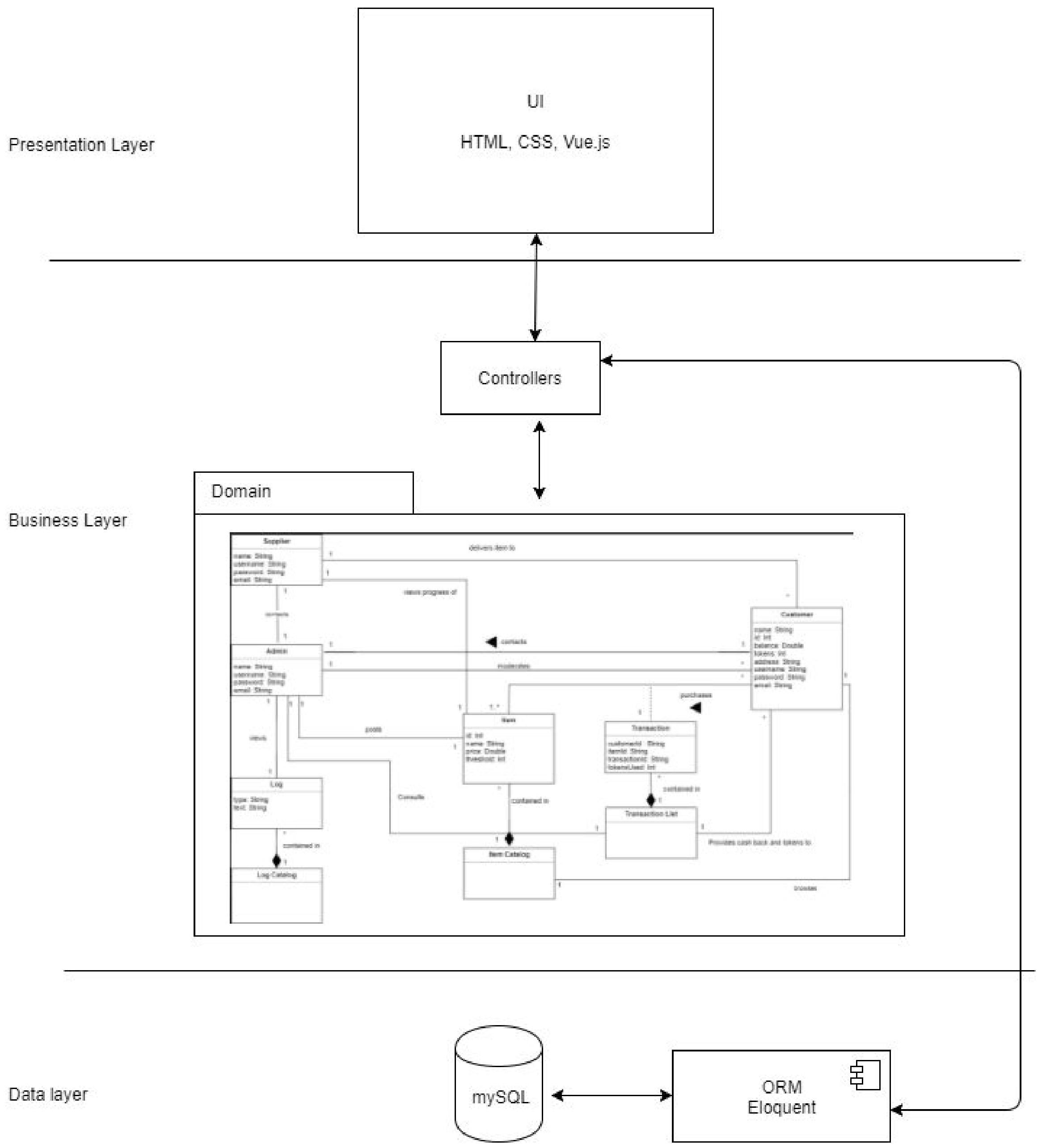

Diagram 2: Architectural Diagram

## Plan up to next release

[Iteration 4, Release 1]

## Infrastructure

### Bootstrap
Bootstrap is one of our front-end design frameworks of choice as it comes in a package with Laravel. Additionally, it is the most popular CSS front-end framework, meaning there are lots of resources and documentation on it. It is responsive which is excellent as our stakeholder emphasized that the website should work well on desktop as well as on phones. Bootstrap also supports all major browsers.

Other frameworks include Foundation and Semantic UI. Foundation is not too different from Bootstrap. It also works on a grid system and is very responsive, but the community around it is smaller and we do not really need to use its extra functionalities. Semantic UI is also similar to the previous two frameworks, but its tags are supposed to be more semantic, making it easier for programmers to code. However, it is large and apparently buggy.

### Laravel
We have decided to use Laravel because it is the most popular framework for PHP, our back-end language of choice, and it strongly enforces the MVC pattern. Laravel has extensive documentation and makes it simple to implement many important features such as authentication or logging.  It is also easy to setup and start coding with. Additionally, Laravel is well suited for testing as there is built-in support for testing with PHPUnit.

Other php back-end libraries are Symfony and CodeIgniter. Symfony is known to be a mature, stable framework that can make the development of web apps very secure and maintainable. However, it is known for having a steep learning curve and we do not want to be spending most of our time figuring out a complex framework. CodeIgniter is a lightweight and easy to learn framework. However, it is now slightly outdated and is no longer officially supported.

### Dusk
We are using Laravel Dusk to perform browser tests. This testing API uses ChromeDriver in order to perform tests therefore we don’t need to install additional plugins or softwares. A large quantity of documentation is available online to help us build our browser tests. The learning curve for Dusk is very low due to the simplicity in the algorithms of the tests.

Another popular automated web testing tool is Selenium. While both of these tools utilize ChromeDriver, the fact that Dusk is integrated into Laravel makes it much more simple to setup and quick to use, and reduces the need for other installations.

## MySQL

We are using MySQL as our database management system. This is because it is very well supported and it has  a lot of GUI managing tools. It is also easy to learn and is very fast. It is also what most of our team has experience with, meaning we do not have to learn a new tool.

Other database management systems are Oracle Database, SQLite and PostgreSQL. Oracle Database needs to be paid for, which is unsuitable for our team. SQLite is an embedded database which has no networking capabilities. It also has issues with concurrency, making it inappropriate for our project. PostgreSQL is a completely open source alternative to MySQL, but it is less popular and thus difficult to get support or to google as many questions for it. PostgreSQL can also run slower for read-heavy operations.

## Vue.js

Vue.js is our front-end library of choice. It integrates well with Laravel; it is provided in a package while downloading Laravel. Vue can also be easily integrated with other front-end libraries, making it very versatile. Other benefits of Vue are the facts that it is quick and lightweight, ideal for our website as we do not want users to wait for pages to load. Vue is also simple to learn, which is ideal.

Other popular frameworks for the front-end are Angular, React and Ember. These all have a steep or steeper learning curve than Vue. We would have to learn JSX for React and Typescript for Angular. Ember has issues processing quick changes, which can be for the user experience. Additionally, it is big and heavy.

## Stripe

Stripe is our payment service of choice. It allows our website to accept payments without credit card information reaching our server, which alleviates many security concerns associated with a website that accepts payments. It provides a very simple dashboard and API to handle transactions. Stripe also provides excellent documentation which allows the learning curve to be much less of a hassle.

Alternatives are Braintree and Paypal. The main reason we decided not to go with Paypal is that it limits the form of payment, as users will not be able to use credit cards. However, we may implement Paypal later on as an alternative form of payment. Our research indicated that Braintree is a bit more of a hassle to implement as the documentation and API is not as clear, and also does not provide features such as discounts.

**Other Libraries**

The main reason we've selected the following libraries to handle things such as task scheduling, data persistence, dependency injections and logging is due to the fact that they are integrated with Laravel, and Laravel has considerable documentation on how to utilize the following libraries. This will make implementation much more simple, and will avoid significant overhead due to potential compatibility issues:

**Laravel Command Scheduler**
Laravel's integrated command scheduler will be used to schedule tasks that need to be scheduled on the server.

**HTTP Session**
Laravel's HTTP Session will be used for data persistence.

**Service Container**
Laravel's service container will handle class dependency management and dependency injections.

**Cache**
The cache.php file provided by Laravel will be used as our caching mechanism in order to cache our backend as needed.

**Monolog**
The Monolog library will be used for logging. We've decided to use this library as it is integrated with Laravel.

**SwiftMailer**
The SwiftMailer library, integrated with Laravel's mailable classes, will be used to develop and send email messages through our website. Our email testing environment will be through mailtrap.io, which allows us to send fake emails and validate them through a development inbox.

## Name Conventions

For the backend:
PSR-1 - PHP basic coding standards
PSR-2
PSR-4

For the frontend:
Javascript conventions
HTML conventions
CSS conventions

For the server:
SQL naming conventions