Trinity WayFinders

WayFinder Application

Project Development Plan

Revision History

Version	Created Date	Author	Comments
1.0	09-Feb-2019	Mark Grenann	

Authors List

Sl. No.	Name	Role
1.	Mark Grenann	Author
2.	Arun Thundyill Saseendran	Author

Table of Contents

1. Objective

The objective of this document is to explain the project development plan for Trinity WayFinders application - an application for multimodal transportation suggestion application with sustainability in mind, providing all the details of the project development plan in detail. The overview of the document is provided in section 2.

2. Development Plan Overview

The development plan details the iteration plan, the plan for pair programming, and the coding standards that will be followed by the project team during the development.

3. Iteration Plan

The iteration plan is developed targeting a weekly delivery of the product such that at then end of each week a working product with better features than the previous week is available.

Week Number	Deliverable	Component	Comments
W4			
	Basic Map	GUI	
	Source & Destination Input	GUI	Test on Multiple Devices
	HL API	Routing	
	LL API	Routing	
	Containerization	Infra	
	Load Balancer	Infra	
	Build & Deploy	Infra	
	Authentication	Security	
	Kubernetes Setup	Infra	
W5			
	Build Jobs and CI Pipeline	Infra	
	Signup and Login	GUI	
	User's Current Location	GUI	Test On Multiple Devices to get current user location info

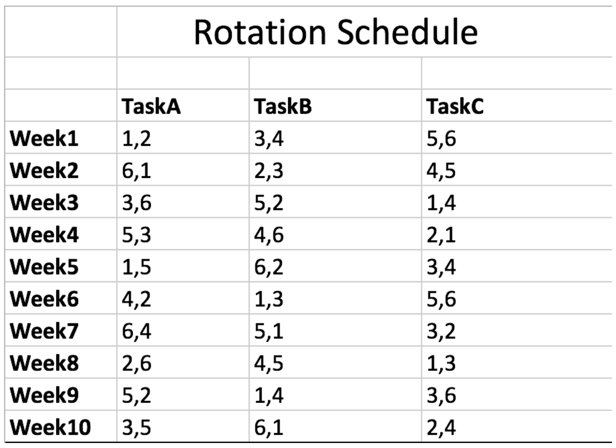
	LL Routing API - Algorithm	Routing	
	Configuration Service + Library	Infra	
	Message Queue + Library	Infra	
	Postgres Setup	Infra	
	MongoDB Setup	Infra	
	Consul	Infra	
W6			
	LL Routing API - Algorithm	Routing	
	Location to Map	GUI	
	User Profile Page with Goal and Transportation Pref	GUI	
	HL API for User Profile Management	User Management	
	Environment Metrics - CO2 & Pollution	Env Metrics	
	Simulation Engine - Pull and Push	Simulation	
	Edge - Connection Logic and Registration	Edge	
	HL API for Edge Registry		
Reading Week			
	Reading Week		
W7			
	Edge - CDN	Edge	
	Edge Manager Service	Edge	
	Routing - Edge, Incentives	Routing	
	Rewards Service LL API	Rewards	
	Rewards HL API	Rewards	

	Extend Profile for Rewards	GUI	
	Connection Interruption Handler	GUI	
	Abstract Connection Handler	GUI	
W8			
	Simulation and Failover	Simulation and DR	
	Environment Metrics - Weather & Traffic	Env Metrics	
	Backend Infra Setup - Compose | Kompose etc	Infra and Deploymen t	
	Administration CLI / API	Admin	
	Edge Caching	Edge	
W9			
	UAT, Bug Fixes, Improvisations		
W10			
	Demo Preparation		

4. Pair Programming

We schedule our time into ten weeks period after lectures end. As we have six members in our team, automatically we have three pairs. Each week one member stays for the same task and the other one swap to other pairs, which makes sure the whole team get a chance to work on different task, every team member get to work with other team members as pair as well. The rotation schedule is attached below as table 1.

Table 1. Rotation Schedule

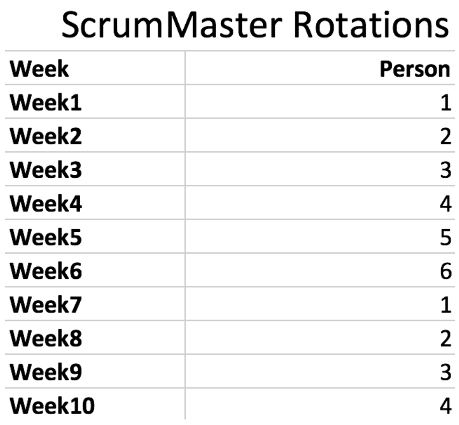


Rotation Schedule

	TaskA	TaskB	TaskC
Week1	1,2	3,4	5,6
Week2	6,1	2,3	4,5
Week3	3,6	5,2	1,4
Week4	5,3	4,6	2,1
Week5	1,5	6,2	3,4
Week6	4,2	1,3	5,6
Week7	6,4	5,1	3,2
Week8	2,6	4,5	1,3
Week9	5,2	1,4	3,6
Week10	3,5	6,1	2,4

Also shown in table 2, we have one person act as scrum master every week.

Table2. ScrumMaster Rotation



ScrumMaster Rotations

Week	Person
Week1	1
Week2	2
Week3	3
Week4	4
Week5	5
Week6	6
Week7	1
Week8	2
Week9	3
Week10	4

5. Coding Standards

The chapter gives a summary of general coding standards that will be used for this project.

In addition, it drills deep to the front-end language Angular 6 and the backend language Java 8 and provide specific instructions and directs to the formatters, lints that can be used to ensure that the coding principles are followed.

5.1 The rule of thumb

Follow the generic principles outlined here when writing each line of code, whichever language it may be. Make use of the formatter and lint where ever possible to ensure the standards.

It is the responsibility of the developer to make sure that the coding standard is met. Also, it is the responsibility of the reviewers to ensure that any code that does not meet the coding standards is blocked from getting into the master.

5.2 Clean Code Principles Summary

- Credits: Clean Code - A Handbook of Agile Software Craftsmanship by Robert C. Martin
- Credits for Summary: https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29

The code is clean if it can be understood easily – by everyone on the team. Clean code can be read and enhanced by a developer other than its original author. With understandability comes readability, changeability, extensibility and maintainability.

5.2.1 General Rules

1. Follow standard conventions.
2. Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.
3. Boy scout rule. Leave the campground cleaner than you found it.
4. Always find the root cause. Always look for the root cause of a problem.

5.2.2 Design Rules

1. Keep configurable data at high levels.
2. Prefer polymorphism to if/else or switch/case.
3. Separate multi-threading code.
4. Prevent over-configurability.
5. Use dependency injection.
6. Follow Law of Demeter. A class should know only its direct dependencies.

5.2.3 Understandability Tips

1. Be consistent. If you do something a certain way, do all similar things in the same way.
2. Use explanatory variables.
3. Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place.
4. Prefer dedicated value objects to a primitive type.
5. Avoid logical dependency. Don't write methods which work correctly depending on something else in the same class.
6. Avoid negative conditionals.

5.2.4 Name Rules

1. Choose descriptive and unambiguous names.
2. Make meaningful distinction.
3. Use pronounceable names.
4. Use searchable names.
5. Replace magic numbers with named constants.
6. Avoid encodings. Don't append prefixes or type information.

5.2.5 Functions Rules

1. Small.
2. Do one thing.
3. Use descriptive names.
4. Prefer fewer arguments.
5. Have no side effects.
6. Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag.

5.2.6 Comments Rules

1. Always try to explain yourself in code.
2. Don't be redundant.
3. Don't add obvious noise.
4. Don't use closing brace comments.
5. Don't comment out code. Just remove.
6. Use as explanation of intent.
7. Use as clarification of code.
8. Use as warning of consequences.

5.2.7 Source Code Structure

1. Separate concepts vertically.
2. The related code should appear vertically dense.
3. Declare variables close to their usage.
4. Dependent functions should be close.
5. Similar functions should be close.
6. Place functions in the downward direction.
7. Keep lines short.
8. Don't use horizontal alignment.
9. Use white space to associate related things and disassociate weakly related.
10. Don't break indentation.

5.2.8 Objects and Data Structures

1. Hide internal structure.
2. Prefer data structures.
3. Avoid hybrids structures (half object and half data).
4. Should be small.
5. Do one thing.
6. A small number of instance variables.
7. The base class should know nothing about their derivatives.
8. Better to have many functions than to pass some code into a function to select a behavior.
9. Prefer non-static methods to static methods.

5.2.9 Tests

1. One assert per test.
2. Readable.
3. Fast.
4. Independent.
5. Repeatable.

6. The Unit Tests should not have dependencies of network or static assets. Mock then as required.

5.2.10 Code Smells

1. Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes.
2. Fragility. The software breaks in many places due to a single change.
3. Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort.
4. Needless Complexity.
5. Needless Repetition.
6. Opacity. The code is hard to understand.

5.3 Java

1. Use the formatter available in the location here. Please refer to the following link to import the formatter into eclipse and use Ctrl + Shift + F to autoformat your code.
2. Also make sure to import the clean-up profile available in the location . Please refer to the following link. Once the code implementation is done, use Source --> Cleanup to run the cleanup wizard with the imported cleanup profile.
3. Make sure to submit your code to SonarQube and ensure that there are no critical warning and no/very less major issues. The documentation for submitting your code to SonarQube. The instruction for SonarQube submission are available in the wiki.

5.4 Angular 6

1. Use ATOM beautify plugin and enable the beautify on save option. Please use the instruction available here

6. Assumptions, Constraints and Dependencies

- The development plan is created assuming that a team of 6 people is available for the entire duration of the project working at least 20 hours a week using eXtreme Programming Methodology.
- There are no major changes in the basic architecture of the application during the course of the project
- It is assumed that the members of the development team are able to adapt to the technological needs of the project without a deep learning time.