**Productivity Apex, Inc.**

Enhancement of Cross Town Improvement Project (C-TIP) Drayage Optimization Proof of Concept Application  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Projects Included In Solution**

|  |  |
| --- | --- |
| **Core** |  |
| PAI.FRATIS.SFL.Common | *- Common interfaces used throughout solution*  *Defines common elements that are composed of objects defined in either the System or System.Runtime.Serializaiton namespaces and a common IEntity interface which underpins a common means for identifying similar objects.* |
| PAI.FRATIS.SFL.Data | *- Data Access Tier: Entity Framework implementation*  *Defines three namespaces, PAI.FRATIS.SFL.Data, PAI.FRATIS.SFL.Data.Mappings, and PAI.FRATIS.SFL.Data.Migrations. PAI.FRATIS.SFL.Data comprises the EntityFramework implementation of the domain objects used by the Drayage Route Optimizer, the Mappings namespace defines any unique mappings necessary from the persistence repository objects to the domain objects. The Migrations namespace is used to create or modify seed data and contains method hooks for implementing run once data conversion methods.* |
| PAI.FRATIS.Domain | *- Business / Domain Objects*  *Comprises several subnamespaces: Congifuration, Equipment, Geography, , Logging, Messaging, Orders, Planning, Times, and Users*  *The Configuration namespace contains configuration and preference objects; the Equipment namespace defines typically amortizable physical goods represented in the optimizer; Geography defines objects that represent either a physical location and situational effects that are geographically bound, such as traffic. Logging provides facilities for outputting debugging and information messages to the file system or screen; Messaging provides an Alert object which can route a message or event from a user to another; Orders defines the objects that are necessary to instantiate an order to process through the optimization algorithm, such as drivers, jobs, jobs status updates, and which actions are available to undertake during a stop; Planning segregates those elements uses to set up and generate an optimization plan for a number of drivers with some number of jobs; Times supplies an access point for fine tuning actions by the day of the week; and Users defines the objects necessary for user management.* |
| PAI.FRATIS.SFL.Infrastructure | *- Project Infrastructure, Ninject Inversion of Control (IoC)*  *Infrastructure segregates the Infrastructure; Infrastructure.Data; Infrastructure.Engine; and Infrastructure.Threading objects that are used to implement the Dependency Injection pattern and support multi-threaded generation of the optimal route.* |
|  |  |
| **Web** |  |
| PAI.FRATIS.SFL.Web.Framework | *Where most of the applications/websites settings is configured. Important Classes, Interfaces, and ViewModels needed by the Web project. Dependency registration, AutoMapper Profiles, and some helper classes to help move around in the Web app.* |
| PAI.FRATIS.SFL.Web | *The main Web application that starts up. Every html page is inside of the View folder. Contains every endpoint inside of the controllers folder. This project acts as the main bridge between the html and the services/optimization.  Contains the ‘client’ folder which contains the MTO Notification Portal and uses angular. To be able to correctly use, you must compile it using npm, bower, and gulp. Afterwards you can access it by placing /mto after the address of the application.* |
|  |  |
| **Services** |  |
| PAI.FRATIS.SFL.Services | *- Services for Domain Objects, Persistence*  *Provides an access point for application access of persistence repositories during execution of the optimization algorithm.* |
|  |  |
| **RoutingEngine** |  |
| PAI.Gis.Routing.Engine | *Configures the traffic engine and calculates distances/time with traffic on.* |
| LaTrafficGenerator/I35RouteGenerator | *Generates a traffic data file to be used by the Routing.Engine* |
| **Optimization Algorithm** |  |
| PAI.Drayage.Optimization | *- Optimization Algorithm*  *The optimization algorithm, wherein jobs and drivers are matched on user provided selection criteria. Typically this is minimizes either time or distance traveled while servicing as many jobs as possible.* |
| PAI.FRATIS.SFL.Optimization.Adapter | *- Intermediary between domain objects and algorithm*  *Access point for further configuration and adjustment of the optimization algorithm or supporting object after initialization and during run-time.* |
| PAI.Drayage.Optimization.Reporting | *- Reporting and statistical service for optimization algorithm solutions*  *A collection of objects and methods that will examine generated routes and schedules to extract meaningful data over the generated route solutions.* |
|  |  |
| **Tests** |  |
| PAI.FRATIS.SFL.DataServices.Tests | *Integration tests for testing database interaction with the PAI.FRATIS.SFL.Services project.* |
| PAI.FRATIS.SFL.Integration.Tests | *Integration tests for data source connections* |
| PAI.FRATIS.SFL.Tests | *Unit tests to ensure the correct behavior of the component objects of the project whole.* |

**The mobile project is not part of the Web/Optimizer solution. The mobile has its own solution and projects**

|  |  |
| --- | --- |
| **Mobile** |  |
| Presentation | This folder and the project inside detail how the mobile view will be shown depending on the device. Mainly a generic all view, Android view, iOS view, and a WinPhone view. |
| PAI.RP.Data.Portable | Configures the repository on the phone to be able to store and retrieve data that has already been collected through rest services or through the actual mobile driver. |
| PAI.RP.Services.Persistence.Portable | A bridge between data and the repository. Mainly used to compress and decompress for storage. |
| PAI.RP.Services.Rest.Portable | The main project that sets up the Restful calls to the main application to retrieve information needed for the mobile project. From here it will be saved internally or used in the services. |
| PAI.lRP.Services.Portable | This is the where all of the business logic goes. Retrieval of the restful data, manipulation, and storage of it. |

The Traffic Service is a separate application from the main project

|  |  |
| --- | --- |
| **TRAFIC SERVICE** |  |
| FratisTraffic | - Tool that generates a historical delay table for routes that have traffic information. The tool works by querying the RIITS database for historical traffic data. It then proceeds to scrub the data and snap it to actual roadways via OSRM. Once all of the data is organized and snapped to the roadways, the delays are then calculated and the data saved to a binary file that can be loaded by the optimization engine.  Note:  - The tool requires OSRM web API 5.4.2 and the RIITS Oracle DB to operate. |

**Summary**

The Drayage Optimizer will produce a solution set that represents the best determined sequence of stops based upon **Locations**, **Drivers**, **Jobs** and **Routes** provided to the algorithm.

The solution is built leveraging v4.5 of the .NET Framework. The optimization class library has a small footprint, and should run successfully on any computer capable of supporting .NET v4.5.

**Inline Code Comments**

Class and property summaries, as well as inline code comments are provided for key logic within this solution. Accompanying unit and integration tests.

**Drayage Optimizer Dependencies**

This optimization algorithm, itself, has no external dependencies aside from the utilization of a Dependency Injector (ex: Ninject, Unity, Castle Windsor) to facilitate Inversion of Control. This process has been demonstrated in the included sample executable with open-source solution “Ninject”.

The additional layers can be implemented on top of the optimization algorithm to readily allow for object persistence, traffic and driving condition analysis via external services.

**Drayage Optimizer Objects**

Within the PAI.Drayage.Optimization project, the Model namespace contains all of the objects that are required in order to build a solution. These key objects will be explained later in this document. At a very high level, the tool requires **Drivers**, **Jobs**, **RouteStops**, and **Locations** in order to produce an optimized solution.

**Database Schema**

The Optimization Algorithm does not require a database in order to properly build solutions. As demonstrated in the sample executable, object can be manually initialized in order to produce a solution without the dependency of a database. If desired, object persistence can easily be achieved using any .NET supported database type and ORM, based upon project requirements and desired implementations.

Within the PAI.FRATIS.SFL.Data project of this solution is an implementation of the Microsoft Entity Framework repository pattern. This can be used to persist domain objects to a database, which can later be queried and mapped to the optimization algorithm to produce a solution from saved data.

**Overview of Drayage Optimizer Key Business Objects**

**(PAI.Drayage.Optimization.Model.Location)  
Location:** a representation of latitude and longitude points that relate to a given location. The Location object is used to represent the address of each of the route stops. The location is identified by the DisplayName property, and the Latitude and Longitude properties are used to represent the coordinates. Each Driver is also assigned a StartingLocation.

**(PAI.Drayage.Optimization.Model.Orders.Driver)  
Driver:** represents a driver that is able to complete a Job. Essential properties are DisplayName (name/nickname), AvailableDrivingHours (maximum TimeSpan for driving in a given day), AvailableDutyHours (maximum TimeSpan for allowable work hours within a day), EarliestStartTime (TimeSpan representing the earliest a Driver can work), and StartingLocation (representing the driver’s starting point).

**(PAI.Drayage.Optimization.Model.Orders.Job)  
Job:** represents a series of load / unloads to be performed by a Driver to satisfy the delivery requirements for a client order. DisplayName identifies the Job, while RouteStops is a collection representing each “stop” of the job (minimum of 2 RouteStops must be provided for each job [pickup and drop-off]).

**(PAI.Drayage.Optimization.Model.Orders.RouteStop)  
RouteStop:** represents each stop / leg of a Job. Complemented by a StopAction which determines the specific action to be performed at each stop. RouteStop objects also have properties for Location (the coordinates of the load/unload), StopDelay (optional – representing the maximum anticipated duration of the stop while on-location), WindowStart and WindowEnd (the earliest and latest time window that a load/unload can be performed at this location).

**(PAI.Drayage.Optimization.Model.Orders.StopAction)  
StopAction:** The StopAction is no longer required in this release of the application. I can simply be ignored or set to N/A when creating a RouteStop.  
 **LocationDistance:** represents the actual travel distance and travel time between two locations based upon current conditions reported by an external mapping / traffic provider. This record can store travel times for every hour of the day within the **Hours** property. These travel times can be used by the algorithm to provide the most accurate driving estimations for all hours of a given day.

**Domain Objects**

The **PAI.FRATIS.SFL.Domain** project contains the domain / business objects that may be used for object persistence. A corresponding **DataService** exists for each of these domain objects in the **PAI.FRATIS.SFL.Services** project. Inline summaries and comments of key properties are provided.

**Optimization Algorithm Adapter**

Since domain objects cannot be provided to the optimization algorithm directly (rather, PAI.Drayage.Optimizer.Model objects must be instantiated), an intermediary project / adapter solution was created in order to facilitate the passing of domain objects to the Drayage Optimizer. Using the **PlanGenerator** service within PAI.Drayage.Optimizer.Adapter project, domain objects can readily be passed to the optimizer. The **PlanGenerator** has methods that accept both PAI.FRATIS.Domain objects, as well as PAI.Drayage.Optimizer.Model objects – so this can be the entry point for consuming the algorithm for either use case. The domain objects will be mapped to the optimization model using the Omu.ValueInjecter **MapperService**.