# OpenTripPlanner (Multimodal Trip Planning)

* provide passenger information and transportation network analysis services
* finds itineraries combining transit, pedestrian, bicycle, and car segments through networks built from widely available, open standard OpenStreetMap and GTFS data

# General Transit Feed Specification (GTFS)

* Making Public Transit Data Universally Accessible
* https://gtfs.org/applications/
* static component -> contains schedule, fare, and geographic transit information
* (agency, stops, routes, trips)
* real-time component -> contains arrival predictions, vehicle positions and service advisories
* A GTFS Realtime feed lets transit agencies provide consumers with realtime information about disruptions to their service (stations closed, lines not operating, important delays, etc.) location of their vehicles, and expected arrival times.

## Ref

* https://www.sidewalklabs.com/blog/gtfs-the-promise-of-data-driven-public-transit/
* https://medium.com/omnimodal/1-billion-reasons-and-counting-to-open-up-that-real-time-transit-data-b7a16b7a8649

## Commands

* **Build ->** mvn clean package -DskipTests
* **Run ->** java -Xmx4G -jar otp-1.4.0-shaded.jar --build /home/user/otp --inMemory
* **Build Graph ->** java -Xmx4G -jar target/otp-1.4.0-shaded.jar --build /home/user/otp
* "/home/user/otp" will contain the pbj(OSM) and gtfs zip files.
* **Run with Graph ->** java -jar target/otp-1.4.0-shaded.jar --server --port 8080 --basePath /home/user/otp/ --graphs /home/user/otp/graphs/ --router pdx

## MU

* It’s a trip planner with combination of multiple transport system (Walking, bicycle, bike, car) using the OpenStreetMap and GTFS data.

# General Bikeshare Feed Specification (GBFS)

* Provide the status of the system at this moment
* Do not provide information whose primary purpose is historical
* Data need to be provided in the JSON file,
* **Files**: gbfs, gbfs\_versions, system\_information, station\_information, station\_status, free\_bike\_status, etc...
* JSON format needs to be followed for all files. (last\_updated, TTL, version, data)
* Versioning and Localization are available.
* Deep Links (beta), Android, IOS, WEB Link
* Possible Future Enhancements: need a way to distinguish between multiple bike types at a station if/when hybrid systems using e-bikes become available
* **License:** Creative Commons Attribution 3.0 License

## Doc & Ref

* https://www.bcycle.com/gbfs
* https://github.com/NABSA/gbfs/blob/master/gbfs.md
* <https://github.com/NABSA/gbfs/blob/master/systems.csv>
* **GBFS Configuration in OTP:** http://docs.opentripplanner.org/en/latest/Configuration/#real-time-data

## Ex

* https://gbfs.bcycle.com/bcycle\_madison/gbfs.json
* https://gbfs.nextbike.net/maps/gbfs/v1/nextbike\_ic/gbfs.json

# Multi-Modal Journey Planning (MMJP) – Hybrid Approach

* Earliest Arrival Problem (EAP) -> Source, destination and departure time
* Range Problem (RP) -> Source, destination and range of time (6 - 8)
* Multi-Criteria Problem (MCP)
* Dijkstra's algorithm is applied to find the closest public network node S (stop or station) to the starting point and the closest node T to the ending point

S--->Ns--->N1--->N2--->Nt--->T

S: Starting Point

T: Target Point

Ns: Starting Point nearest public network

Nt: Target Point nearest public network

N1...n: Nodes

* Ns and Nt will be found by the Dijkstra algorithm.
* In-between nodes of Ns and Nt will be solved by a mathematical model.
* Based on the OSM and GTFS static data, OTP will generate possible routes with multiple transits.
* From the multiple routes, the nearest starting and ending point will be found by Dijkstra algorithm.
* Scan nodes near Source and Target until you hop on a stop of the Public Transportation network.
* And the in-between public transit modal (nodes) will be built by a mathematical model.
* Mixed Integer Linear Program (MILP)
* Greenhouse Gas Emissions (GHG)

# Dijkstra's algorithm

* For finding the shortest paths between nodes in a graph
* Application of the shortest path algorithm is network routing protocols
* Dijkstra follows the greedy method (Relaxed greedy algorithms?)).
* Bellman-Ford is faster than Dijkstra and follows dynamic programming.

## Advantages

* Finding the Shortest Path.
* Distance between the locations refers to edges.
* Used in IP routing to find Open shortest Path First.
* Used in the telephone network.

## Disadvantages

* Wastes lot of time while processing.
* Cannot handle negative edges.

## Greedy algorithm

A greedy algorithm is an algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.

We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the subproblem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference between dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage and may reconsider the previous stage's algorithmic path to the solution.

## Ref

* https://cs.nyu.edu/courses/summer07/G22.2340-001/Presentations/Puthuparampil.pdf
* https://github.com/eugenp/tutorials/blob/master/algorithms-miscellaneous-2/src/test/java/com/baeldung/algorithms/DijkstraAlgorithmLongRunningUnitTest.java

# OTP Data Layer (In-Progress)

* Understand the OTP code.
* How obj is generated while running the OTP jar.
* Code can be modified?
  + If yes,
* Can be extended. Affected by future release.
* Choose DB based on the input data.

Ref

* https://github.com/HSLdevcom/digitransit/wiki/Guide-for-building-and-viewing-Digitransit-OTP-graphs-locally
* http://docs.opentripplanner.org/en/latest/Configuration/#routers

# Keycloak (Identity & Access management)

* Open Source. Written in Java
* Single Sign-on/ out
* Identity Brokering (Validation of the identity between different services via OpenID Connect or SAML 2.0 IdPs) and Social Login
* Connection to existing LDAP or Active Directory servers. Also, have support to add the existing data stores. RDBMS also supported.
* Support for OpenID Connect, OAuth 2.0, and SAML
* **License:** Apache License 2.0
* A realm secures and manages security metadata for a set of users, applications, and registered OAuth clients.

## Advantages

* Client adaptors, which does most of the heavy lifting when it comes to token validation, token refresh etc.

## Ref & Links

* <https://www.keycloak.org/docs/latest/securing_apps/>
* GitHub: <https://github.com/keycloak/keycloak>
* <https://medium.com/@sanjayaben/authentication-as-a-service-for-enterprise-applications-828540f74c9b> \*

## Identity management (IDM)

* Is a framework of policies and technologies for ensuring that the proper people in an enterprise have the appropriate access to technology resources.
* identify, authenticate and authorize
* Storing user information, user identity, access level
* Authentication: Verification
* Authorization: Managing authorization information that defines what operations an entity can perform in the context of a specific application
* Interchange: The SAML protocol is a prominent means used to exchange identity information between two identity domains. OpenID Connect is another such protocol.

## Access control

* restriction of access to a place or other resource[1] while access management describes the process
* Locks and login credentials are two analogous mechanisms of access control.

## Authentication

* authentication is the process of verifying that "you are who you say you are"
* verifying a user's identity is often required to allow access to confidential data or systems

## Authorization

* Authorization is the function of specifying access rights/privileges to resources
* authorization is the process of verifying that "you are permitted to do what you are trying to do"

# First Mile

* The distance of to and from each of the transit stations en route

<https://somethingaboutorange.com/first-mile-last-mile-problem/>

# Estimated Time of Arrival (ETA)

* Need real-time data of the transport (GPS will be used).
* real-time transit data
* Providers need to send the transport current location to the data centre (GTFS, GBFS).
* Trip planner Service can fetch the data and calculate the ETA

## Ref

* <https://core.ac.uk/download/pdf/82293981.pdf>

# Crowdsourcing

Crowdsourcing is a joint process development or problem-solving technique that requires help from a network of people, or crowd. This network is usually connected via the Internet or through a specific website

Use of Crowdsourcing

* Check the quality of the application
* Marketing
* Reduce of Time (Existing Solution)
* New Ideas

# SoS

* Access to medical and travel security specialists 24/7 to identify risks and offer practical tips and planning for individual and company-wide travel safety.
  + Storing Medical details of the passengers.
  + Notifying nearest medical shop.
  + Smart device connections with the application
* Informing family or connected person.

# ToDo

What are apps using the Dijkstra algorithm?

If they are using other algorithms, why?

OTP: API, Response, GTFS-RT (Real-Time) connection.

* <https://stackoverflow.com/questions/19889617/open-trip-planner-scalibility-or-an-alternative-library>
* http://graphserver.github.io/graphserver/
* https://groups.google.com/forum/#!topic/opentripplanner-users/4qF4UPhKs2w
* LDAP or active directory
* OpenID Connect support.
* OAuth 2.0 support.
* SAML support.

# Others

## Windows

### Commands

* systeminfo
* tasklist

# Java Code Structure

* config - class which will read from property files
* cache - caching mechanism class files
* constants - constant defined class
* controller - controller class
* exception - exception class
* model - pojos classes will be present
* security - security classes
* service - Impl classes
* util - utility classes
* validation - validators classes
* bootloader - main class

The service implementation should be a Spring bean (it either has to have a @Component or @Service annotation or should be defined in a Spring XML configuration file) so that Spring will find it and register it in the Spring application context.

Then you use dependency injection, through the @Autowired annotation, to inject the implementation of the service into the controller. This means that Spring will look at your controller, it will find the @Autowired annotation on the service member variable and initialize it with a bean that it finds in the application context, which will be the instance of the service implementation class that it has registered earlier. So, after Spring is done, service will refer to the instance of ServiceImpl

- Object Oriented Programming and development of REST API based middleware using Spring Framework.

- JAVA, spring web MVC module, Spring ORM Module, Spring Batch, Spring Security and Spring Boot.

- HTML5, CSS3, JavaScript, jQuery, Angularjs, Nodejs and Hybrid mobile application development.

- Test-Driven Development (TDD) environment using testing tools like Junit, Mockito for JAVA based application and Karma, jasmine for Angular based web application.

- Design middleware REST API using server-side JavaScript NodeJS, Expressjs and Mongo db.

- Design micro services using SOA architecture and deployed on to Pivotal cloud foundry.

- CICD pipe line using Jenkins.

- Sonar Cube, JSLint.

- TDD

- Coding standards

- Security driven coding

- DevOps - Process, Tools used by us

- Fundas of Microservices Architecture

- Event Sourcing Microservices Architecture

- Saga Pattern

- Gradle Build for SpringBoot

- Basic algorithms used for MMJP

---------------------------------------------------------------------------------------

Kubernet

--------

- Cluster -> Services -> Namespaces -> Pods (Process) -> Containers -> Image

- Cheetsheet: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Cluster:

- kubectl cluster-info

Config:

- kubectl config view

Service:

- kubectl get services

Namespace:

- kubectl get namespaces

Pod:

- kubectl get pods -n <<NAMESPACE\_NAME>> -o wide

- kubectl get pod <<POD\_NAME>> -n <<NAMESPACE\_NAME>> -o yaml

- kubectl describe pods <<POD\_NAME>> -n <<NAMESPACE\_NAME>>

- kubectl logs <<POD\_NAME>> -c <<CONTAINER\_NAME>> -n <<NAMESPACE\_NAME>>

- kubectl port-forward <<POD\_NAME>> 5000:6000