Jignesh Patel

Jpatel200@hotmail.com

POLICY COMMISSION CALCULATOR

User Guide

**Contents**

[**Installation Instructions** 2](#_Toc518458072)

[**To run the application:** 2](#_Toc518458073)

[**WebService detail:** 2](#_Toc518458074)

[**To test the WebService:** 2](#_Toc518458075)

[**Project Structure** 3](#_Toc518458076)

[**Code Documentation** 4](#_Toc518458077)

[**Class Diagram** 5](#_Toc518458078)

[**Assumptions and Design Considerations** 6](#_Toc518458079)

[**TO DO** 7](#_Toc518458080)

# **Installation Instructions**

## **To run the application:**

1. Unzip the content from attached “insurance-agent-commission-app.zip” and navigate to the project root directory
2. Run “npm install” from within project root directory to install application dependencies
3. Run “npm test” to run through the unit tests
4. Run “node standalone.js” to run the program in standalone mode. This will use the given data (policy amount, agent hierarchy, commission plan) and computes commission summary.
5. Run “npm start” to run the application as RestFul service hosted on express server.

## **WebService detail:**

**Service endpoint:** localhost:3000/api/v.1.0/policy

**Method:** POST

**Header Parameter: Content-Type**: application/json

## **To test the WebService:**

1. Using REST Client tool ( e.g. Postman )
2. Submit POST request to endpoint “**localhost:3000/api/v.1.0/policy**” using request body:

{

"policy" : {

"commissionPlan" : "B",

"policyAmount": 100000

},

"agentHierarchy" :{

"sellingAgent" : {"agentName" : "Selling Agent", "commission" : 0.02},

"superAgents":[

{ "agentName": "SuperAgent\_A", "commission" : 0.0250},

{ "agentName": "SuperAgent\_B", "commission" : 0.0325},

{ "agentName": "SuperAgent\_C", "commission" : 0.0225}

]

}

}

1. WebService will compute agents commission and responds with commission summary payload

{

"commissions": {

"sellingAgent": 1400,

"SuperAgent\_A": 200,

"SuperAgent\_B": 130,

"SuperAgent\_C": 0

}

}

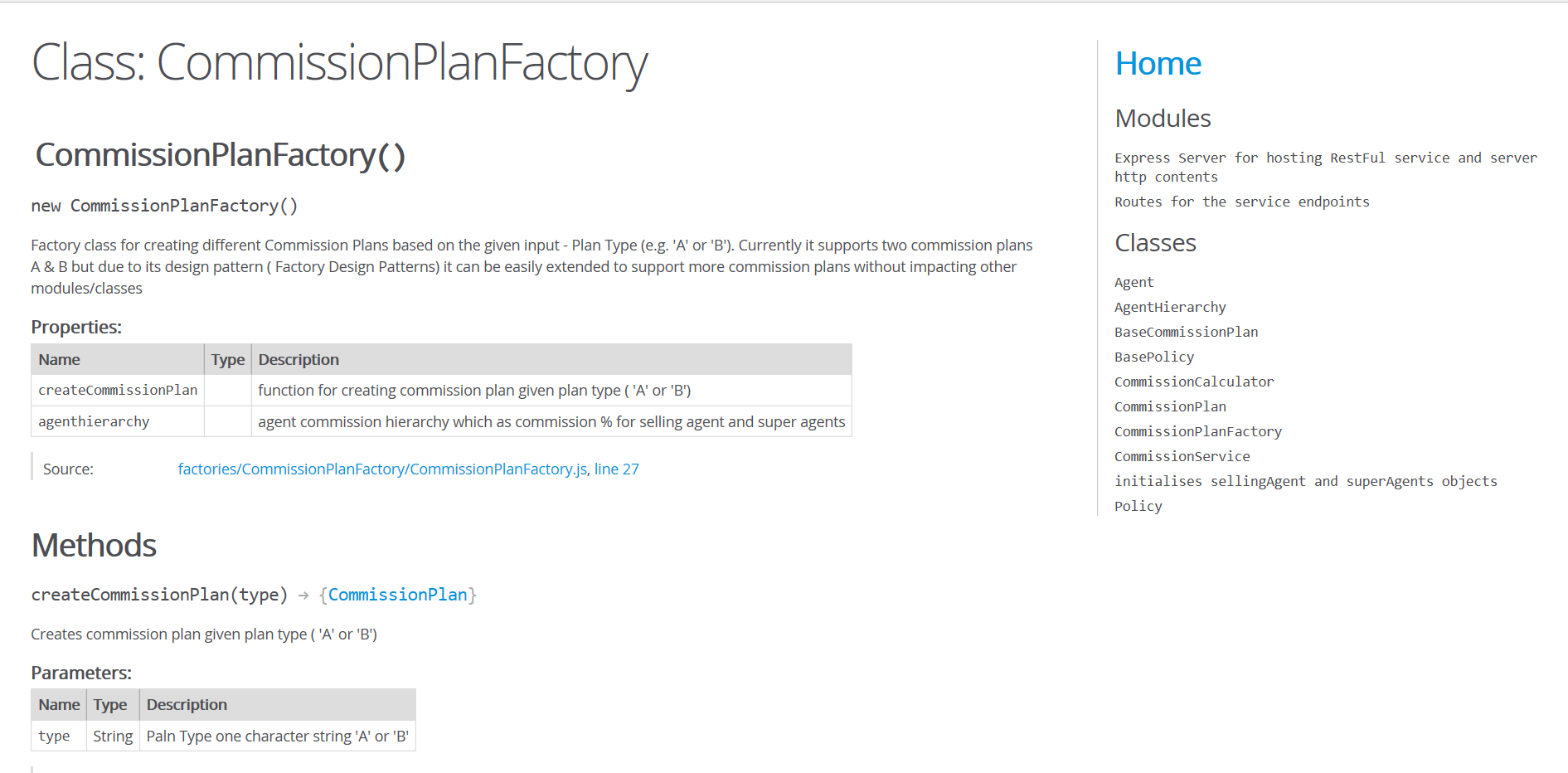
# **Project Structure**

* **Insurance-agent-commission-app**: root directory
* **config**: contains configuration and initialization classes for the application
* **jsdoc**: contains documentation of the project
* **factories**: contains factory classes for creating commission plans
* **models**: contains entity base classes for holding data e.g. Agent, Policy, Agent Hierarchy
* **public**: contains static contents for the project
* **routes**: houses webservice routing detail
* **server**: contains module to run express web server
* **service**: houses services and controller modules
* **tests** : houses all unit tests files for the application
* **app.js** : module contains code to bootstrap webservice
* **package.json** : Houses app of the dependencies and scripts.

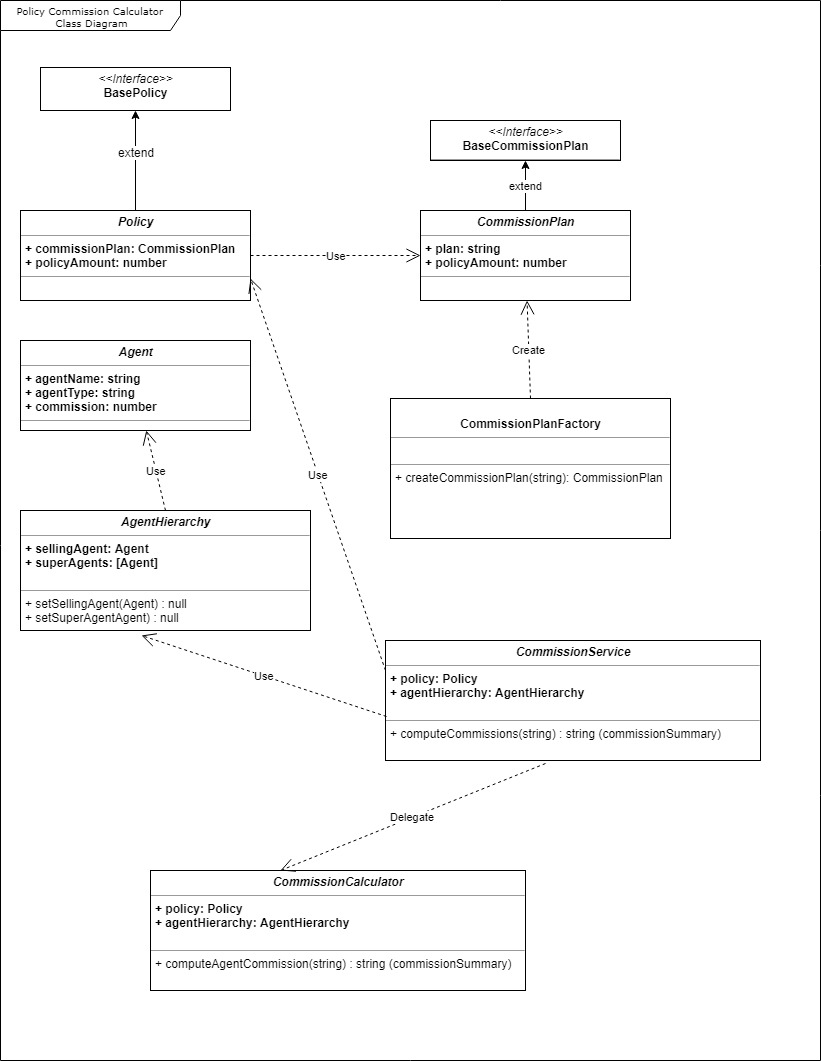
# **Code Documentation**

Application code is documented using JSDOC annotations. The document is generated in the rich html format using JSDOC node module.

The documentation exists within the project structure in the **jsdoc** folder. The revised code documentation can be generated using “npm run jsdoc” script. The script is added into the package.json file to generate the documentation for the project.



# **Class Diagram**



# **Assumptions and Design Considerations**

Below assumptions are made while developing the solution for the policy commission calculator.

1. The company wants to leverage the policy commission calculator logic as a service in other applications running on different platforms. Hence the logic is made available as RESTFul webservice.
2. The company currently has two commission plan A and B but in future company wants to add additional commission plans easily. Design considerations are made to accommodate additional commission plans relatively easily. Factor design pattern is used to abstract the commission plans from the business logic.
3. Versioning of the webservice is required to support future enhancements and backward compatibility. API endpoints URL is designed to support versioning of the API.
4. The company may have business requirement to revise the commission plan. Commission plan configuration is externalized into JSON file so it can be easily changed without impacting the business logic and code.
5. There will be limited number of super agents in the agent hierarchy for e.g will not contain more than 10. Based on this assumption Array is used for storing super agents and their commissions.
6. There will be only one selling agent in the agent hierarchy. Data structure for storing the selling agent is designed
7. Commission should have upper ceiling, for e.g. it should not exceed 20% for policy amount. Assumption is made during developing the application that it should not exceed 20%. Application can be enhanced to externalize the configuration for the commission ceiling.

# **TO DO**

1. Externalize the validation and error messages into resource file
2. Develop more unit test coverage
3. Tighten up constructor and method parameter validations
4. Add in dept code documentation using JSDOC comments
5. Prepare in depth design document by documenting application flow using sequence and flow diagrams
6. Externalize the commission ceiling into config file so it can be changed easily.
7. Profile the application using node built in profiler for identifying bottlenecks, memory leaks and CPU spikes.
8. Conduct the peer review of the application to get feedback on design and code implementation.
9. Explore if any other design patterns can be leveraged with in the application