**TEST STRATEGY**

**Introduction:** Trade Matching is an API based application which is used to create a trade and fetch its detail based on existing entities.

In our application, a party can search a trade based on its trade reference number or all its trades based on its trade status (confirmed, unconfirmed, cancelled, exit).

A party can also update its trade based on its trade reference number and can also cancel its trade.

A trade will exit after completing its maturity date, but it will exist in our database.

**Objectives and tasks:**

**Scope:**

* **Components to be tested:**
* **Models:** Here we will the following points:

1. Schema of the database is created successfully or not.
2. Specifications of the entities in the database schema are defined correctly or not.

* **Services:** It will contain our main business logic and we will test the following points:

1. **Create trade:** The trade creation logic is working properly or not i.e., the data entered by the user is reflected in the database. We will test that our matching logic is working properly or not.
2. **Fetching trade/trades:** Here we will check the expected trade as a response after requesting a trade based on trade reference number or multiple trades by a party based on its status.
3. **Updation of a trade:** This will contain the logic of trade updation. It is of following types:

* When a user wants to update the entities(fields) of a trade.
* When user wants to cancel a trade.
* When a trade exits after its maturity dates.
* **Controller:** It will contain all our APIs for creating, fetching, updating a trade.
* **Repository:** It will be connected to our database and here we will checkthe CRUD operations are working properly or not.
* **Application properties:** It will contain the properties for the database connectivity.

**Testing Strategy:**

* **Unit testing:** In this we will test the smallest functionality or modules of an application. The main objective of unit testing is to isolate written code to test and determine if it is working as intended.

In our application we will perform unit testing on every module of components (controller, services, repository and models).

* **Integration:** In this we will test different modules, units or components of our application as a combine entity.
* **System testing/End to end testing:** Here we will test all the modules or components of our application as a single unit.
* **Acceptance testing:** It is done by the user. Generally, the user check that all the given requirements are being fulfilled or not.

**Hardware Requirements:**

**Environment Requirements:**

* Java
* Spring Tool suits
* Maven
* Swagger
* Junit
* REST Assured

**Test Schedule:** We will perform testing after completion of every module.

**Features to be Tested:**

We will test every module and functionality of GET, PUT, POST, PATCH in controller, services, repository and application properties.

**Features not to be Tested:**

We will not test pom.xml file because we have just imported dependencies.

**Schedules:**

* Test Plan
* Test Cases
* Test summary report

**Tools:**

* **Junit**
* **REST Assured**

**Approvals:**

**Shankar Ganesh**

**Mohit Kumar Khatri**