**App Design for Tic-Tac-Toe**

**I. State**

The state object contains all the information for the tic-tac-toe game.

Its Javascript object below and Java class (StateData.java) is consistent:

stateData = {

"player":CROSS,

"sequenceNum":0,

"stateOfGame":PLAYING,

"errorMessage":0,

"selectedRow":0,

"selectedColumn":0,

"board":[

[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY]

]

};

**II. Back-end**

1. In this project, for RESTful Web service, @Controller and @RequestMapping mechanisms (Spring Web MVC framework) were used. The following dependency in the pom.xml file provide the mechanisms:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

2. DispatcherServlet (defined in web.xml) dispatches request to handler (HomeController.java), with URI mapping ("/submitNewState/") which is defined in @RequestMapping annotation.

@RequestMapping(value = "/submitNewState", method = RequestMethod.***POST***, consumes = "application/json", produces = "application/json")

3. The request has a JSON object parameter. The JSON object is automatically converted to a Java object (StateData.java). The mechanism is implemented by @RequestBody which is annotated parameters for access to the HTTP request body. Parameter values are converted to the declared method argument type using HttpMessageConverters. The following dependency in the pom.xml file provides the mechanisms:

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>${jackson.version}</version>

</dependency>

4. After the automatic data type conversion, the StateData Java object is sent to the TicTacToe.java class and the game process is moved on.

5. After processing the game, a new StateData Java object is created.

6. Response includes a new JSON object (which was converted from the new StateData Java object) and is sent to the ajax call.

**III. Front-End**

1. In the web page, game player can select row and column numbers (in drop down) for her each move in the game.

2. At first, a stateData object is initialized. With each input from the player, the stateData object is updated and the ajax call sends the stateData object as a request data.

3. Once the ajax function gets response, to keep the state information, it saves the JSON information in the window.sessionStorage as follows:

window.sessionStorage.setItem("currentStateData", JSON.stringify(stateData));

In the next move, player sends new move with the stored state.

stateData = window.sessionStorage.getItem("currentStateData")

In this way, the application keeps the state of the game all the way to the end of game.

**IV. Unit Test**

1. Created test code in the TestTicTacToe.java file using junit test.

Implemented functions for testing doMove() and UpdateState() methods

2. The following dependency in the pom.xml file provides the junit:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

**V. Game Play**

1. In the web page, at first, a user (CROSS) can select row and column number from drop-downs and click “Move” button to make a move. After that, another user (NOUGHT) can do the same.

2. If users want to restart any time, they can just click the “Restart” button.

**VI. Future Enhancement**

1. Currently the application declares “DRAW” when there is no winner and all the cells are filled. We can enhance this function to declare draw if no player can "possibly" win, even before all the cells are filled.

2. Allowing to play on the larger board than 3 by 3, and, in that case, the checking condition in hasWon() (TicTacToe.java file) needs to be enhanced too.

3. Dynamic drop-down size in the UI in case playing on the larger board than 3 by 3

4. Login/Signup features

5. User information management feature

6. Design DB tables and set DB connection for recording all the game records