Requirements

You'll create a Spring Boot REST application using **JDBC Template** to access the database.

A Game should have an answer and a status (in progress or finished). While the game is in progress, users should not be able to see the answer. The answer will be a **4-digit number with no duplicate digits**.

Each Round will have a guess, the time of the guess, and the result of the guess in the format **"e:0:p:0"** where "e" stands for exact matches and "p" stands for partial matches.

You will need several REST endpoints for this:

- **"begin"** - POST – Starts a game, generates an answer, and sets the correct status. Should return a 201 CREATED message as well as the created gameId.

- **"guess"** – POST – Makes a guess by passing the guess and gameId in as JSON. The program must calculate the results of the guess and mark the game finished if the guess is correct. It returns the Round object with the results filled in.

- **"game"** – GET – Returns a list of all games. Be sure in-progress games do not display their answer.

- **"game/{gameId}"** - GET – Returns a specific game based on ID. Be sure in-progress games do not display their answer.

- **"rounds/{gameId}"** – GET – Returns a list of rounds for the specified game sorted by time.

You should include a Service layer to manage the game rules, such as generating initial answers for a game and calculating the results of a guess.

All of your public **DAO interface methods should be tested thoroughly**.

My implementation

My implementation tries to follow the MVC pattern except there isn't really a View in this context. I use the **ResponseEntity** object from Spring Boot to ensure that the data can be returned properly and this way, I can collect DTOs from the service layer and return them as they are. The ResponseEntity will also take in a HttpStatus and therefore this can be customised e.g. when the Game is started, it will return a **CREATED 201** status and when a guess is made, it will return a **ACCEPTED 202** status.

NOTE: this implementation is designed so that multiple games can be in progress at any one time and they do not have to be played "in order". Therefore, it does not hold in memory any game or round state unless it is being prepared to be returned as a **ResponseEntity**.

The controller works as a **RestController** and uses URL mappings to methods in the class definition. The controller does not pass on data to a view, it just serializes it to JSON and returns it.

I have written a **GameFinishedException** which is used to either tell the user they cannot guess anymore, or that they have just won the game.

In terms of DTOs, I use **Game** and **Round** objects. The Round contains the guess data including the guess number.

There is a service layer that handles the logic:

- Beginning game
- Generating an answer
- Make a guess and if the guess is valid, then save the round
- Get all games
- Get games by ID.
- Find all rounds by a game ID.
- Censoring a game answer: this will create a new game object if the game has not been finished and then change the correctAnswer to -1 so that the user cannot see it. If the game has finished then it will return the same Game object rather than copying it to a new one.

The whole application uses Spring DI by default and uses Spring JDBC involving the JDBC Template.

I use Spring Boot to test my game as well using a **TestApplicationConfiguration** class.

I have written two test classes: **GameDAOTest**, **RoundDAOTest** to ensure that data access works as we expect. To ensure real data is not harmed, I use a separate application.properties in my projects test folder to **only manipulate a test database**.

Random number generator

My generator for a new answer ensuring that all digits are unique works by guessing a random number of X digits and then checks if it has unique digits - if not, then it re-guesses a random number. It is a brute force approach but still ensures the numbers are pseudo random.

My database design

Game

gameId (PK)	INT
correctAnswer	INT
isFinished	boolean

Round

roundId (PK)	INT
gameId (FK)	INT
roundNumber	INT
roundStarted	TIMESTAMP
result	VARCHAR(45)

Script used to create database tables

```
CREATE TABLE `Game` (
  `gameId` int NOT NULL AUTO_INCREMENT,
  `isFinished` tinyint NOT NULL,
  `correctAnswer` int NOT NULL,
  PRIMARY KEY (`gameId`)
)

CREATE TABLE `Round` (
  `roundId` int NOT NULL AUTO_INCREMENT,
  `gameId` int NOT NULL,
```

```sql
  `roundStarted` timestamp NOT NULL,
  `roundNumber` int NOT NULL,
  `result` varchar(45) NOT NULL,
  PRIMARY KEY (`roundId`),
  KEY `fk_gameId` (`gameId`),
  CONSTRAINT `fk_gameId` FOREIGN KEY (`gameId`) REFERENCES `Game`(`gameId`)
)
```