Repository Layer Tech Stack Report

What they do:

Postgres is an open-source relational database management system, so is needed as part of the project to store tournament, player and team information in tables on the database.

JPA is used to map Java object to tables in a relational database, used because it is less error prone and time consuming that writing SQL queries. JPA allows using databases without using database semantics. JPA is a part of Spring Data.

Spring Data is used to provide a consistent Spring-based programming model for data access while retaining the features of the underlying data store.

The repository is an abstraction of the data layer. It only deals with data access.

How they work:

- Postgres uses a client server model
  - The server process manages the database files by accepting connections from the client and doing actions on their behalf.
  - Can handle multiple concurrent connections
- Spring Data analyses new Repositories
  - Looks at all the methods defined
  - Automatically generates queries from the method names

Postgres Features:

- Data types
  - Array data type functionality means that if a tournament has multiple rule set requirements it can be added as an array in the record instead of making another table that stores tournament id and a ruleset in each record (a linked information table would be better for this).
  - There are also data type for dates and money, which would allow easily storing the tournament dates and the reward for winning the tournament in the database.
- Foreign Keys
  - Means that the database can be relational and can be used for storing which teams are taking part in which tournaments with team_id and tournament_id as foreign keys, this would then allow easily finding which teams are taking part in tournaments. The team_id and tournament_id could be stored in a linked information table for this.
- Aggregate function
  - Can be used to calculate sum, averages, max, min, etc...
  - Could be used for example to find the team with highest win rate.
- Indexes
  - A way to enhance database performance, database server uses this to find specific rows quickly. Indexes also add overhead so shouldn't be overused.
- Order By
  - Used to have the output sort the records in desired order.
  - Could be used to sort teams by win rate.
- Transactions

- Used to update records of a table in a single operation. Concurrent transactions can't see intermediate steps of a transaction, so if there is a failure during the transaction the database is not effected by the intermediate steps.

Spring Data Features:

- Repository and custom object mapping abstractions
  - Spring Data Repositories are and abstraction which reduce the amount of code required for data access layers.
- Dynamically derives queries
  - Uses the repository names to derive the queries.
- Domain base classes which provide basic properties
  - Domain class is the model and represents a persistent entity that is mapped onto a database.
- Transparent auditing
  - Spring data allows keeping track of the time an entity was created or changed and who did it.

Using JPA:

Creating an entity and it's corresponding repository

```
@Entity
class Team {
        @Id @GeneratedValue(strategy = GenerationType.AUTO)
        private Long id;
        private String name;

        // getters and setters for each attribute also required
}

interface TeamRepository extends Repository<Team, Long> {
        Team save(Team team);
        Optional<Team> findById(long id);
}
```

CRUD (Create Read Update Delete) functionality:

the CrudRepository interface provides CRUD functionality for an entity class that is being managed

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
        <S extends T> S save(S entity); // Saves an entity
        Optional<T> findById(ID primaryKey); // Returns entity with given primary key
        Iterable<T> findAll(); // Return all entities
        long count(); // Returns number of entities
        void delete(T entity); // Deletes entity
        boolean existsById(ID primaryKey); // Checks if an entity with the given ID exist
        //....
}
```

Defining Repositories:

Example:

A query follows the structure:
Type<Entity> query(search parameters)

```
interface TeamRepository extends Repository<Team, Long> {
    //Type<Repository> query(search parameters)
        List<Team> findByIdAndName(Long id, String name);

        // Enables the distinct flag for the query
        List<Team> findDistinctTeamsByIdOrName(Long id, String name);
        List<Team> findTeamsDistinctByIdOrName(Long id, String name);

        // Enabling ignoring case (upper/lower case) for an individual property
        List<Team> findByNameIgnoreCase(String name);
        // Enabling ignoring case for all suitable properties
        List<Team> findByIdAndNameAllIgnoreCase(Long id, String name);

        // Enabling static ORDER BY for a query
        List<Team> findByNameOrderByFirstnameAsc(String name);
        List<Team> findByNameOrderByFirstnameDesc(String name);
}
```

Subject Keywords

`find...by` - typically returns repository type, collection, streamable, or a result wrapper. Can be used in combination with additional keywords
`exists...By` - check if exists typically returns a boolean
`count...By` - counts number of matching results and returns as a numeric value
`delete...By` - acts as Delete query either returns void or delete count
`...First<number>...` - limits to first number of results, should be used between the `find` and `by` keywords
`...Disitinct...` - used to return only unique results, should be used between the `find` and `by` keywords

Predicate Keywords

Logical Keyword -> Keyword expression
`AND` -> `And`, `OR` -> `Or`, `AFTER` -> `After`, `CONTAINING` -> `Containing`, `BETWEEN` -> `Between` etc... (there are also alternatives but they all have a version that follows this pattern)

Modifier Keywords

`IgnoreCase` - used for case-insensitive comparison
`AllIgnoreCase` - ignore case for all suitable properties, used anywhere in query method
`OrderBy...` - specify order

Query Method Return Types

`List<T>` - Fetches all results in an iterable, requires a single query. Can be time intensive.
`Stream<T>` - Fetches results in chunks where the size depends on stream consumption, structure requires a cursor and must be closed to avoid leaks.

Paging and Sorting

Simple sorting expression:

```
Sort sort = Sort.by("teamName").ascending().and(Sort.by("teamID").ascending);
```

Limiting Query Results

Limits results with `FirstX` or `TopX` where `X` is an integer if just `First` or `Top` will return only one result

```
List<Player> findFirst10ByWinCountDesc();
```

Projections

Projections act in a similar way to views in a database.

Interface-based Projections

In this example a projection called WinLosses made using an interface makes it so that when retrieving from the entity if used only returns the team name, win count and loss count for the result.

```
interface WinLosses {
        String getTeamName();
        int getWinCount();
        int getLossCount();
}

interface TeamRepository extends Repository<Team, Long> {
        Collection<WinLosses> findByTeamName(String teamName);
}
```

Class-based Projections (DTOs / Data Transfer Objects)

DTOs hold properties for the fields that are going to be retrieved. Used in the same way as interface projections, but without proxying.

DTO projection example:

```
record WinLossesDTO(String teamName, int winCount, int lossCount) {
}
```

Links Used (probably more useful than what I've written):

Postgres Documentation
Accessing Spring Data with JPA
JPA Reference
Spring Data Overview
JPA Query Keywords **Very useful**
More on projections
JPA Query Methods