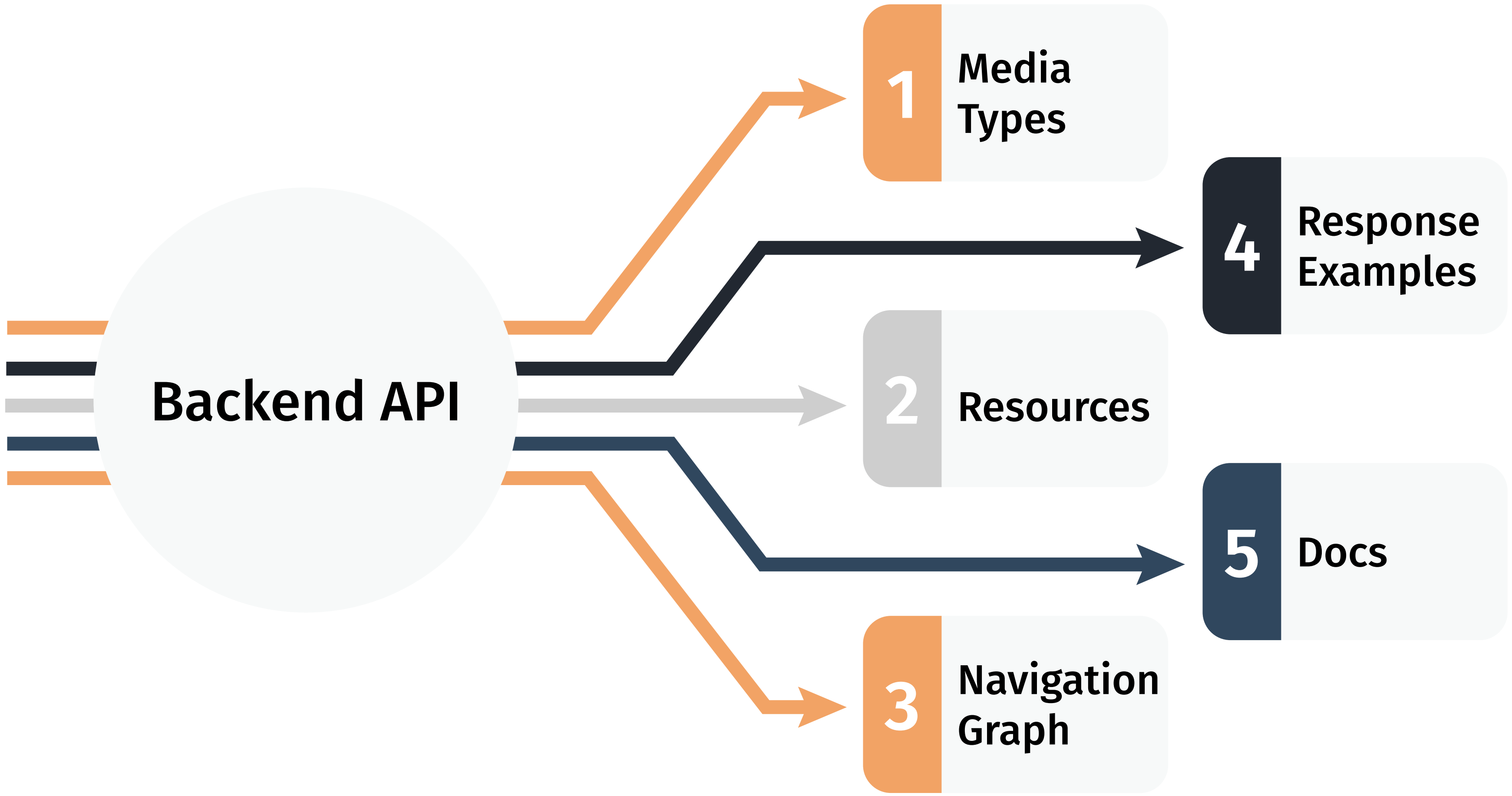
Web Applications Development

Phase 1 - Backend

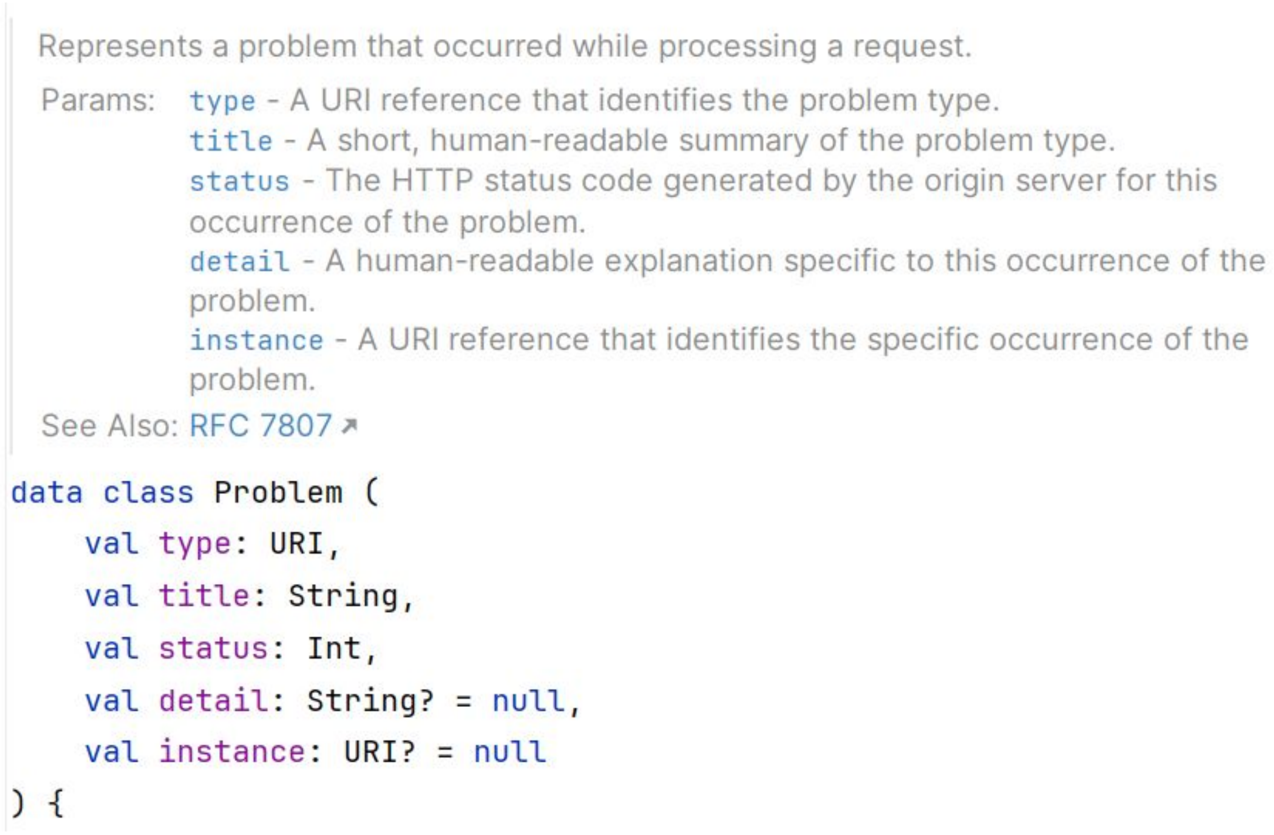

Backend API
1 Media Types
4 Response Examples
2 Resources
5 Docs
3 Navigation Graph

Media Types

application/json

application/
problem+json

Problem

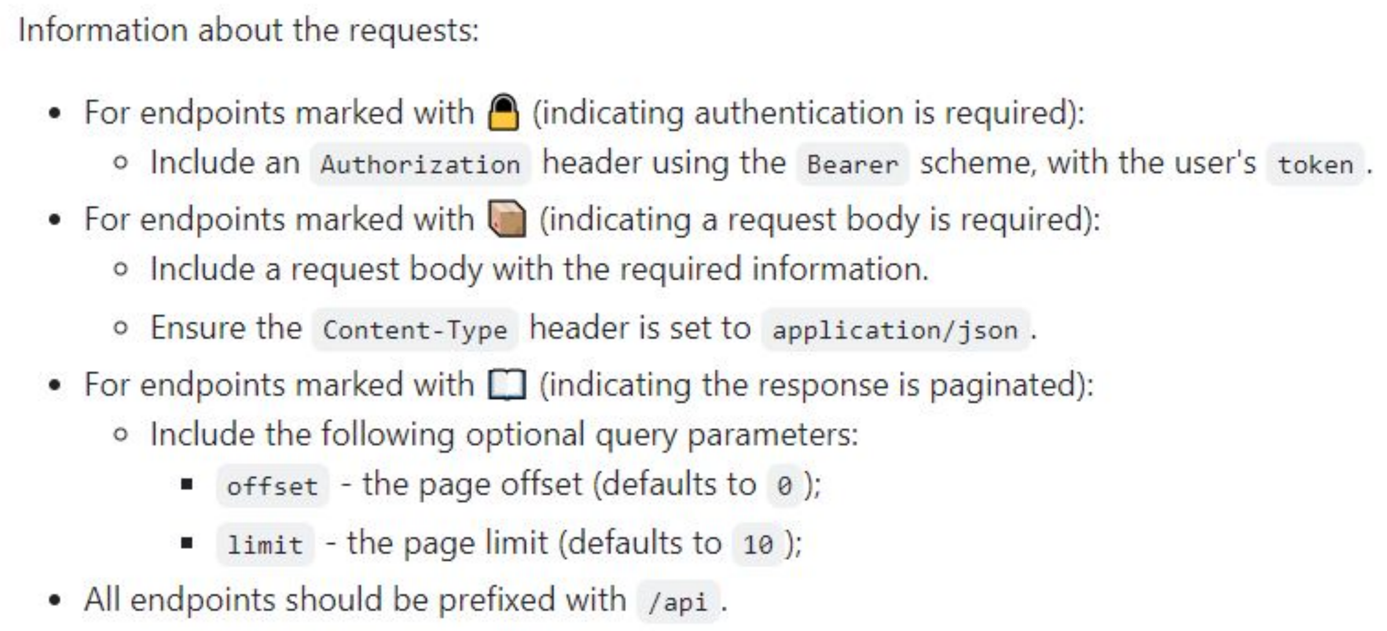


Represents a problem that occurred while processing a request.

Params:	`type` - A URI reference that identifies the problem type.
	`title` - A short, human-readable summary of the problem type.
	`status` - The HTTP status code generated by the origin server for this occurrence of the problem.
	`detail` - A human-readable explanation specific to this occurrence of the problem.
	`instance` - A URI reference that identifies the specific occurrence of the problem.
See Also:	RFC 7807 ↗

```kotlin
data class Problem (
    val type: URI,
    val title: String,
    val status: Int,
    val detail: String? = null,
    val instance: URI? = null
) {
```

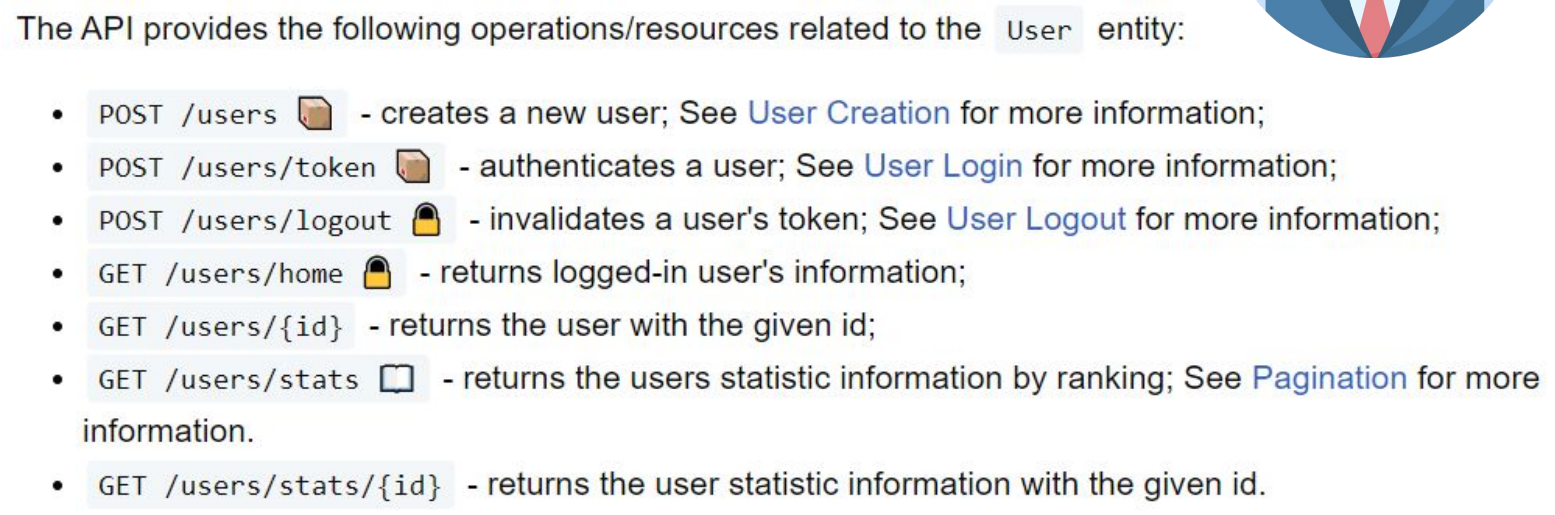
Requests



Information about the requests:

- For endpoints marked with 🔒 (indicating authentication is required):
  - Include an `Authorization` header using the `Bearer` scheme, with the user's `token`.
- For endpoints marked with 📦 (indicating a request body is required):
  - Include a request body with the required information.
  - Ensure the `Content-Type` header is set to `application/json`.
- For endpoints marked with 📖 (indicating the response is paginated):
  - Include the following optional query parameters:
    - `offset` - the page offset (defaults to `0`);
    - `limit` - the page limit (defaults to `10`);
- All endpoints should be prefixed with `/api`.

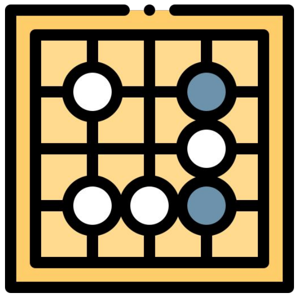
Resources - User

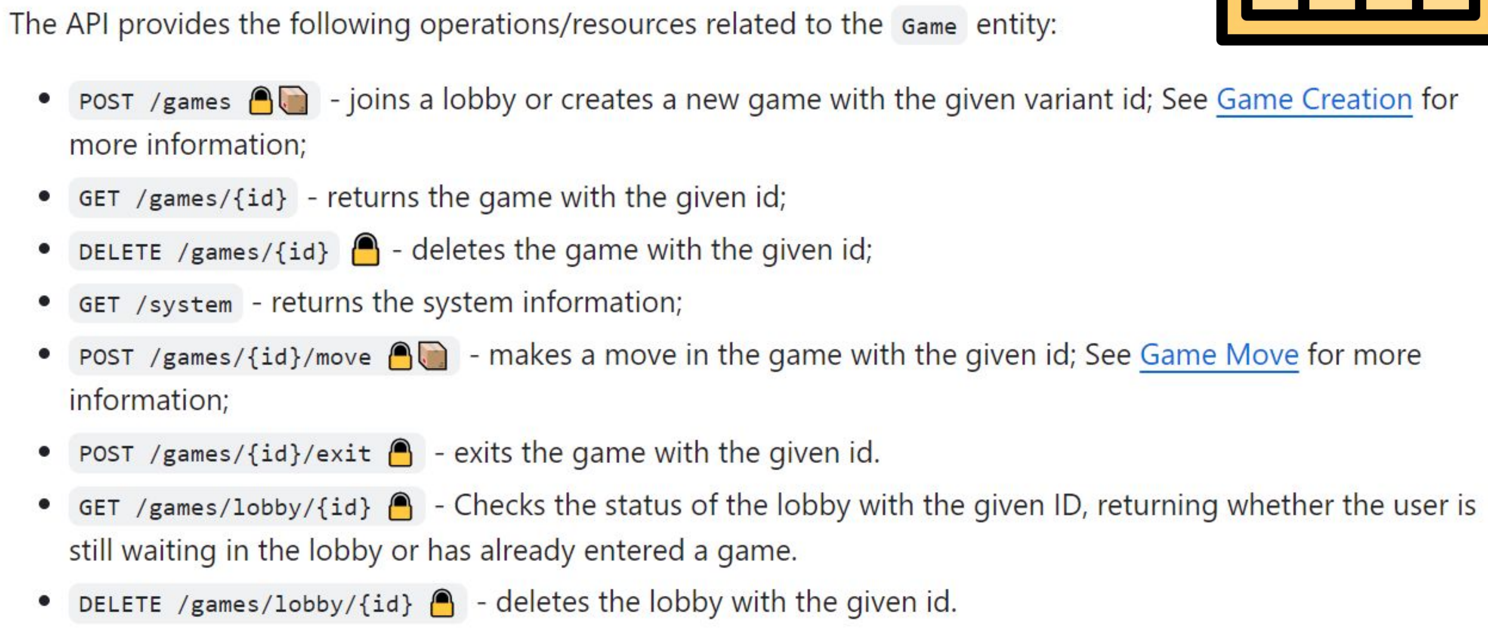


The API provides the following operations/resources related to the `User` entity:

- `POST /users 📦` - creates a new user; See User Creation for more information;
- `POST /users/token 📦` - authenticates a user; See User Login for more information;
- `POST /users/logout 🔒` - invalidates a user's token; See User Logout for more information;
- `GET /users/home 🔒` - returns logged-in user's information;
- `GET /users/{id}` - returns the user with the given id;
- `GET /users/stats 📖` - returns the users statistic information by ranking; See Pagination for more information.
- `GET /users/stats/{id}` - returns the user statistic information with the given id.

Resources - Game

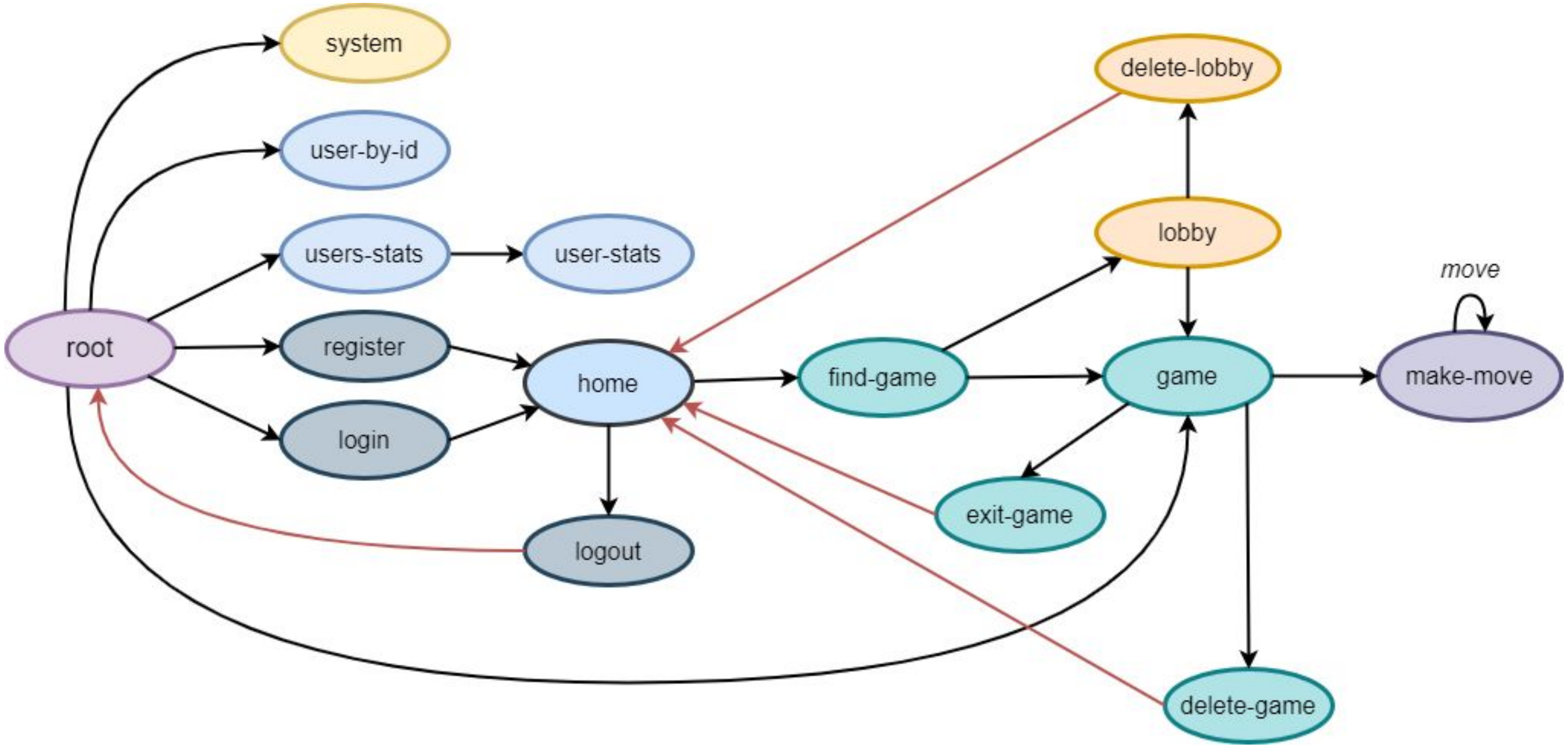




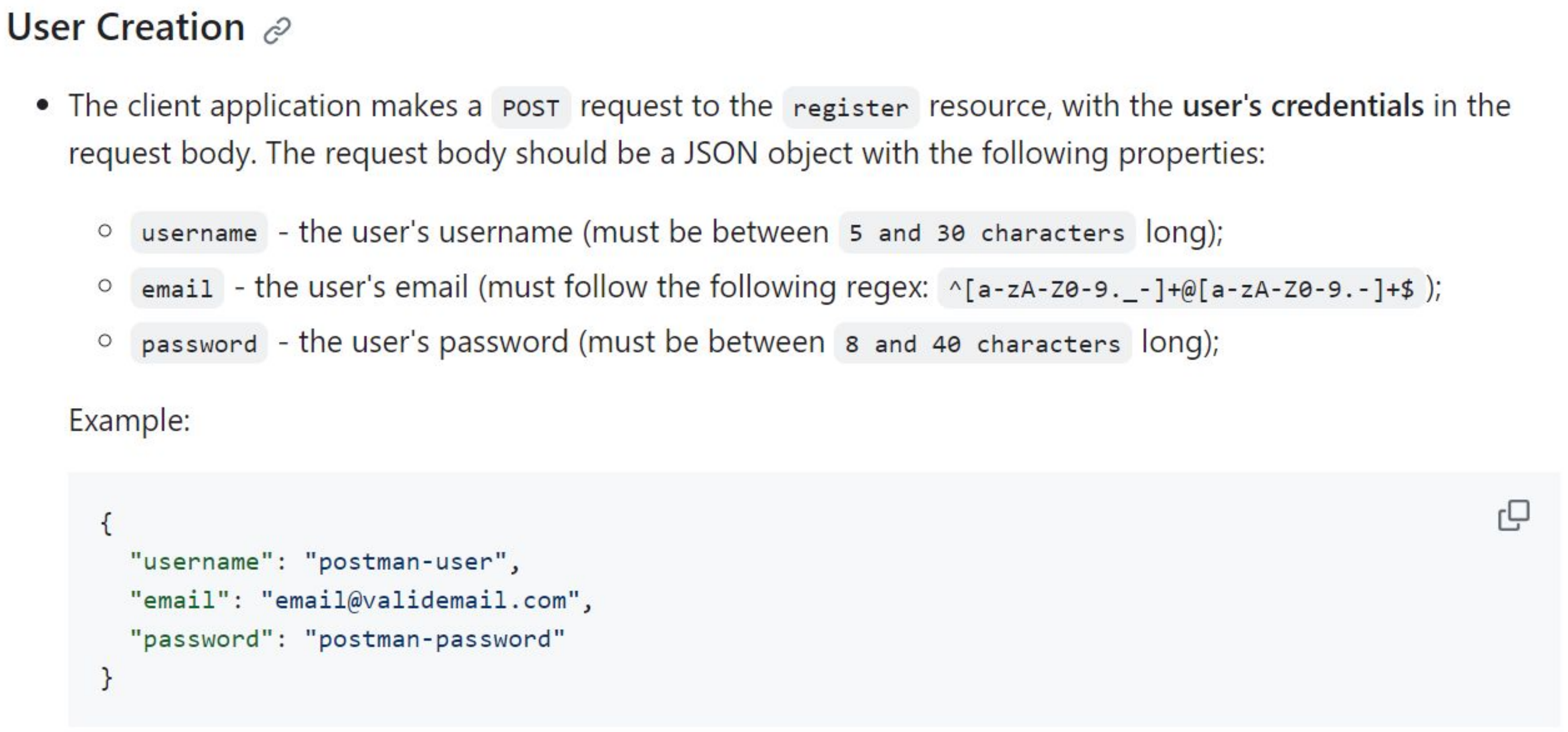
The API provides the following operations/resources related to the `Game` entity:

- `POST /games` 🔒📦 - joins a lobby or creates a new game with the given variant id; See Game Creation for more information;
- `GET /games/{id}` - returns the game with the given id;
- `DELETE /games/{id}` 🔒 - deletes the game with the given id;
- `GET /system` - returns the system information;
- `POST /games/{id}/move` 🔒📦 - makes a move in the game with the given id; See Game Move for more information;
- `POST /games/{id}/exit` 🔒 - exits the game with the given id.
- `GET /games/lobby/{id}` 🔒 - Checks the status of the lobby with the given ID, returning whether the user is still waiting in the lobby or has already entered a game.
- `DELETE /games/lobby/{id}` 🔒 - deletes the lobby with the given id.

Navigation Graph



Response Examples



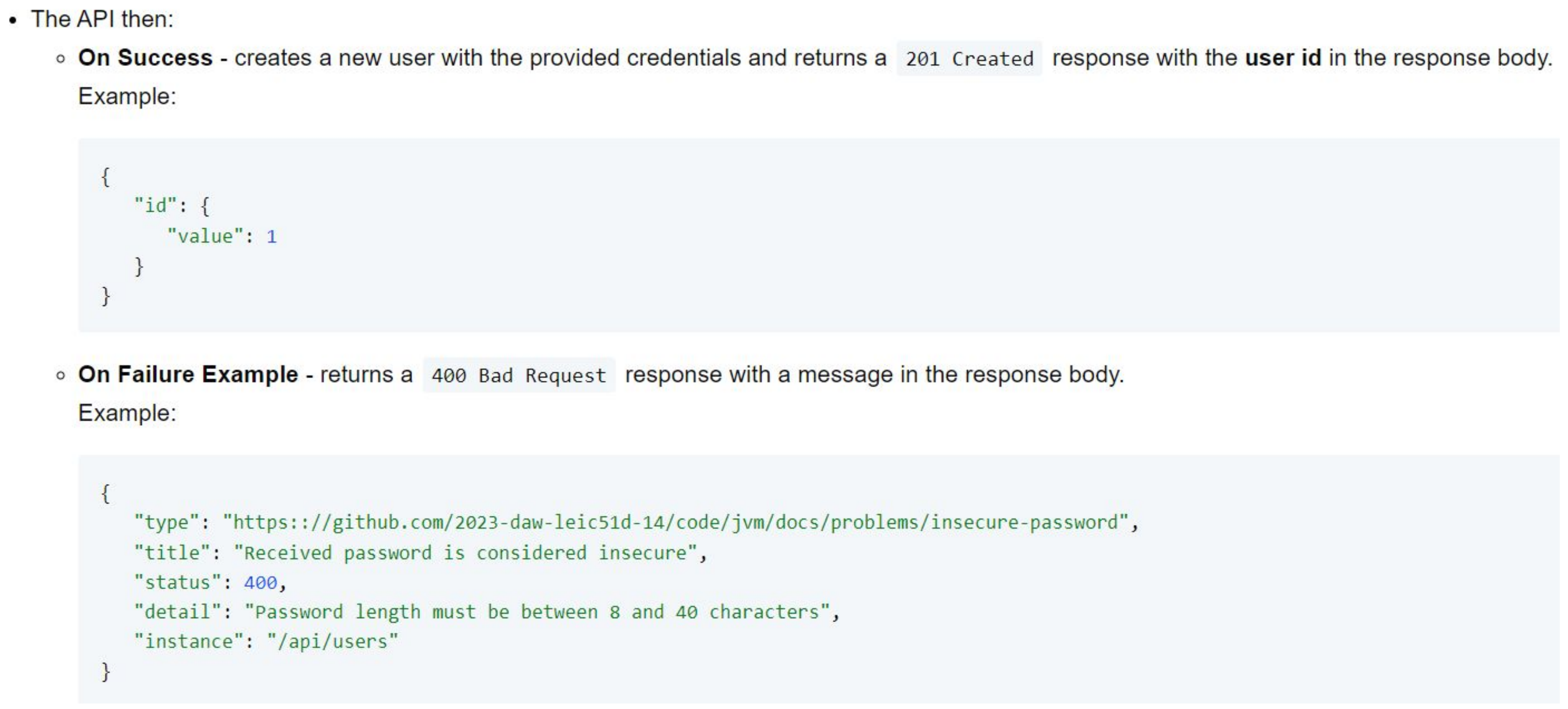
User Creation

- The client application makes a `POST` request to the `register` resource, with the **user's credentials** in the request body. The request body should be a JSON object with the following properties:
  - `username` - the user's username (must be between `5 and 30 characters` long);
  - `email` - the user's email (must follow the following regex: `^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+$`);
  - `password` - the user's password (must be between `8 and 40 characters` long);

  Example:

```
{
  "username": "postman-user",
  "email": "email@validemail.com",
  "password": "postman-password"
}
```

Response Examples



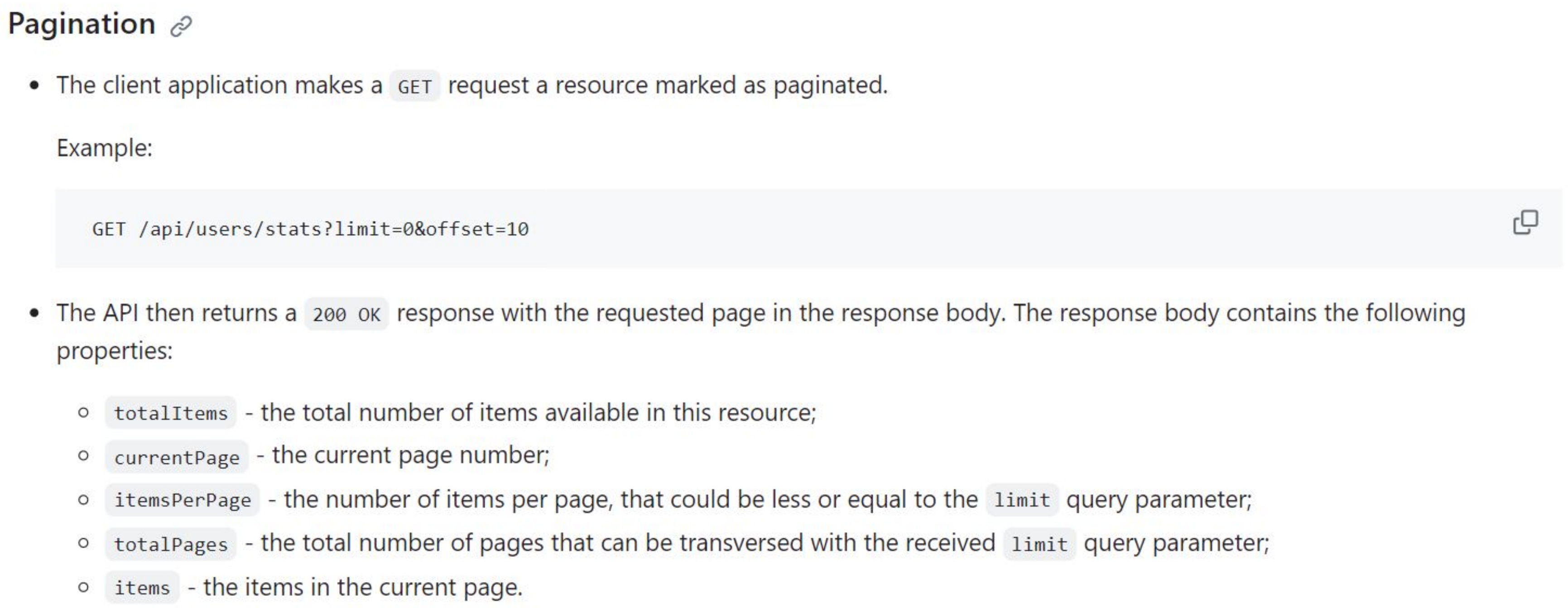
- The API then:
  - **On Success** - creates a new user with the provided credentials and returns a `201 Created` response with the **user id** in the response body.
    Example:

    ```
    {
       "id": {
          "value": 1
       }
    }
    ```

  - **On Failure Example** - returns a `400 Bad Request` response with a message in the response body.
    Example:

    ```
    {
       "type": "https://github.com/2023-daw-leic51d-14/code/jvm/docs/problems/insecure-password",
       "title": "Received password is considered insecure",
       "status": 400,
       "detail": "Password length must be between 8 and 40 characters",
       "instance": "/api/users"
    }
    ```

Response Examples



Pagination

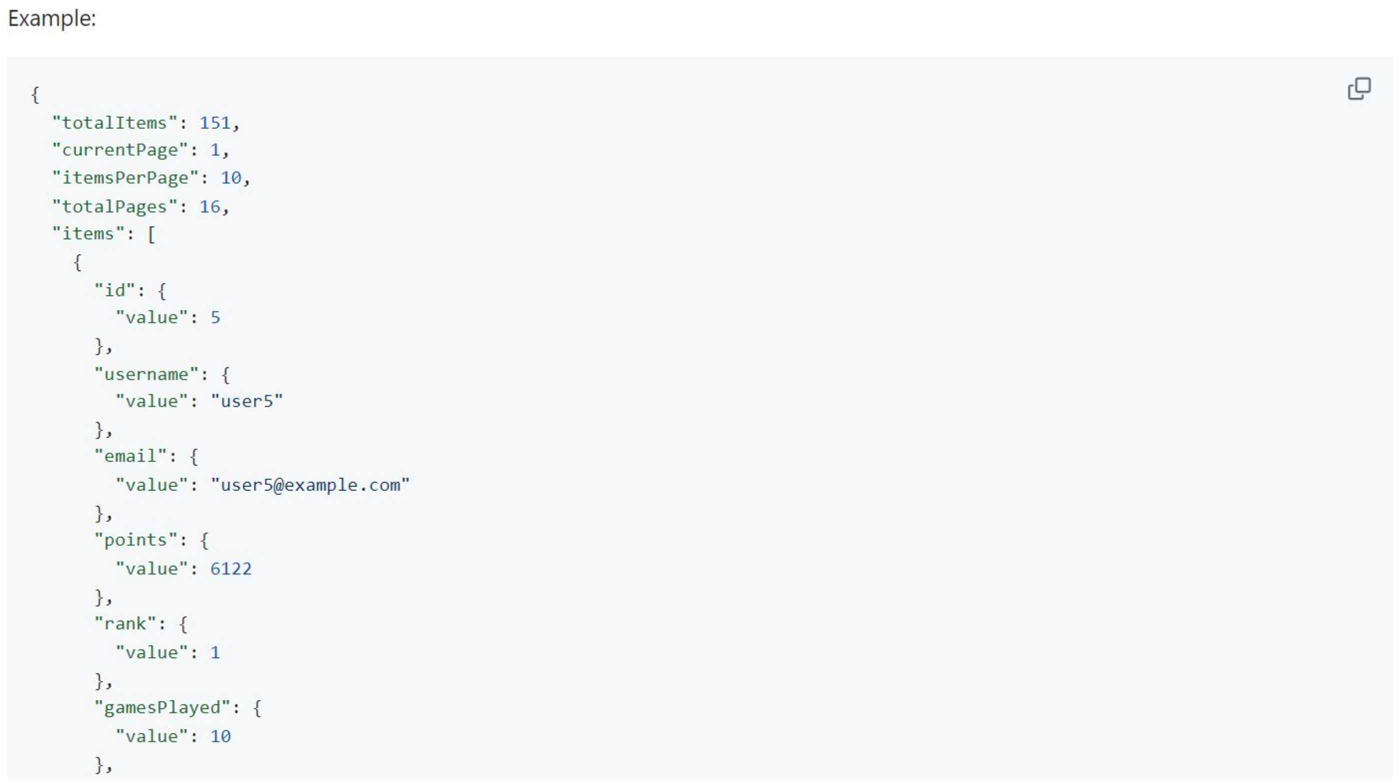
- The client application makes a `GET` request a resource marked as paginated.

  Example:

  ```
  GET /api/users/stats?limit=0&offset=10
  ```

- The API then returns a `200 OK` response with the requested page in the response body. The response body contains the following properties:
  - `totalItems` - the total number of items available in this resource;
  - `currentPage` - the current page number;
  - `itemsPerPage` - the number of items per page, that could be less or equal to the `limit` query parameter;
  - `totalPages` - the total number of pages that can be transversed with the received `limit` query parameter;
  - `items` - the items in the current page.

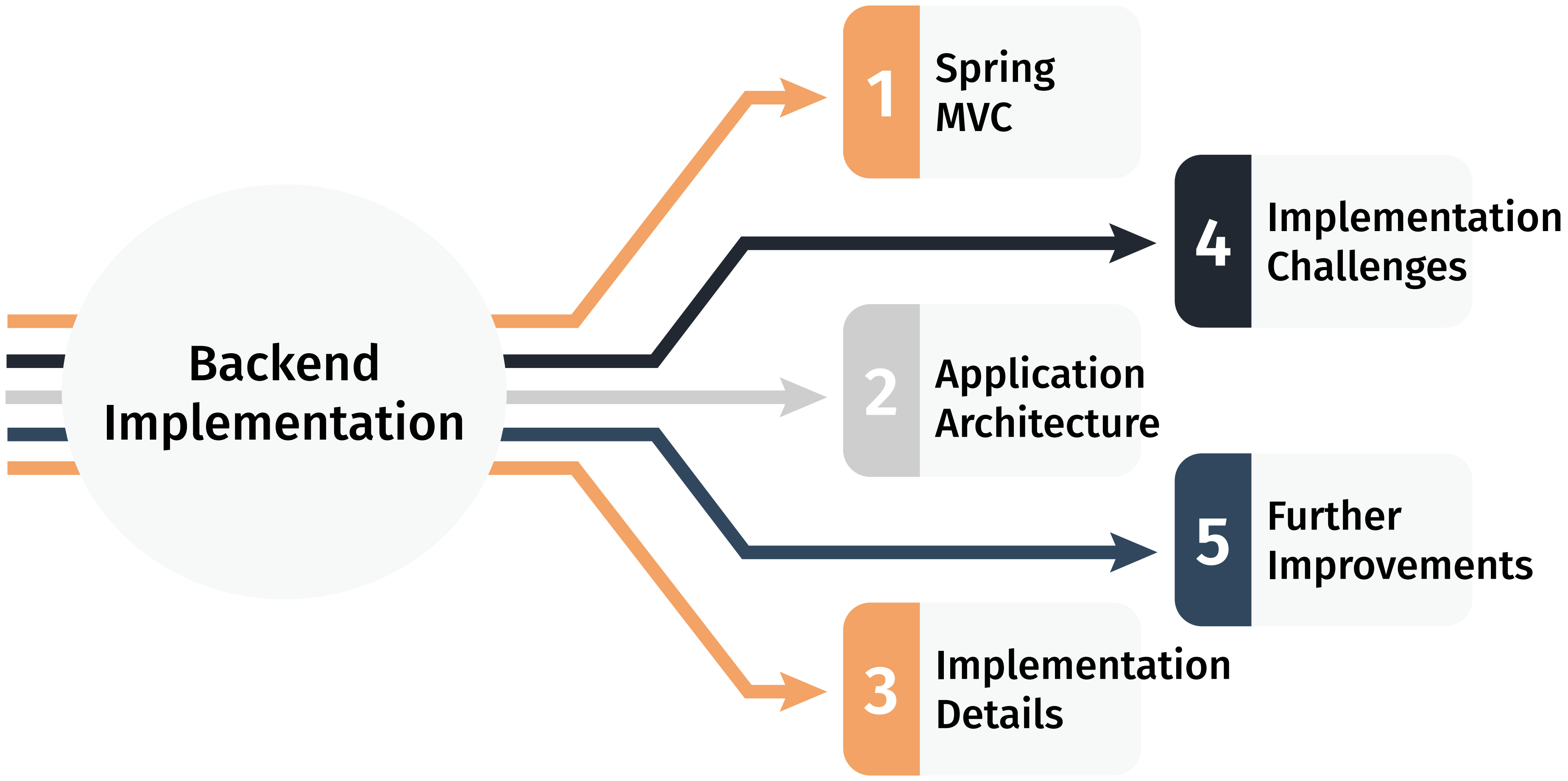
Response Examples

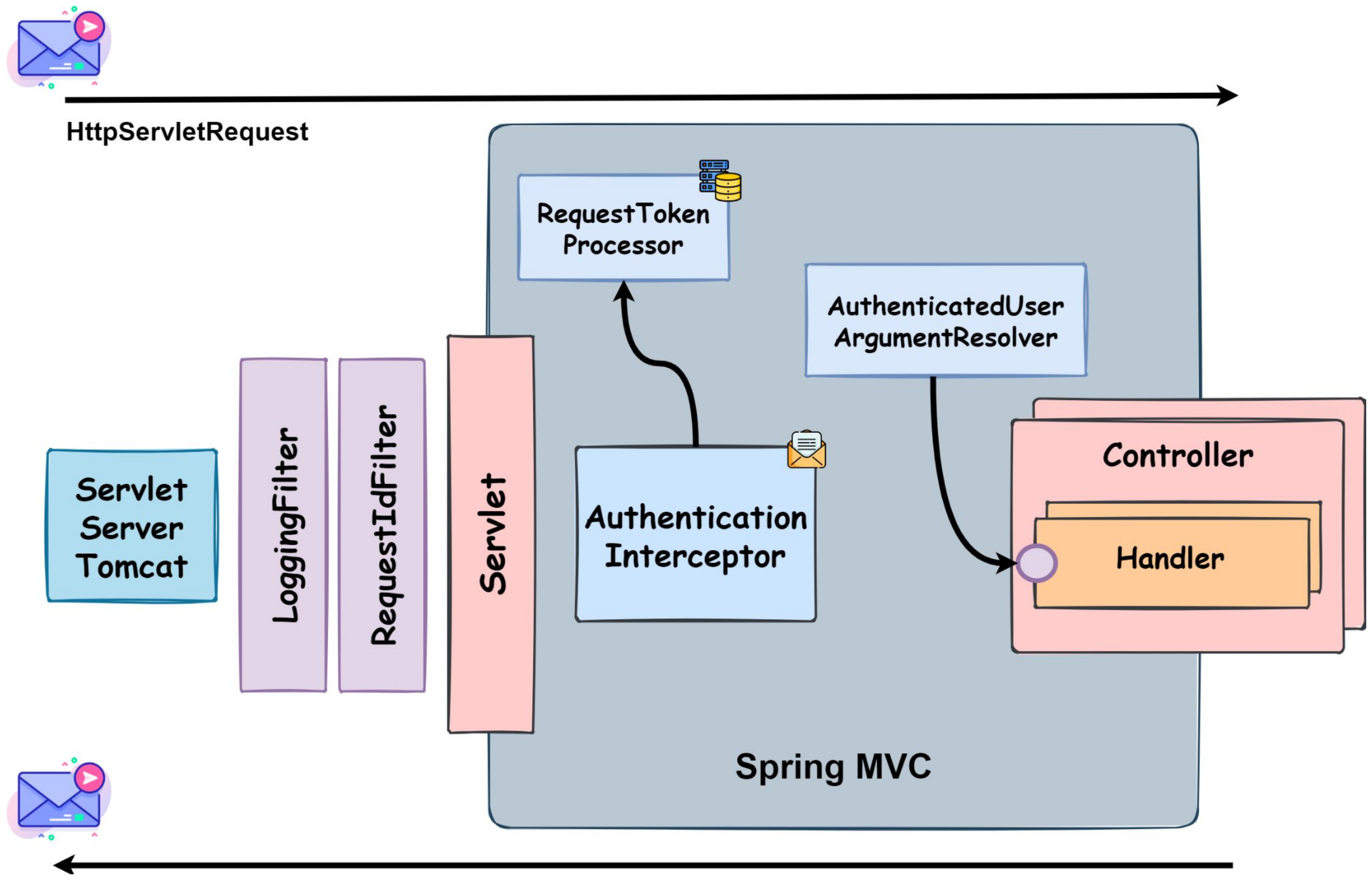


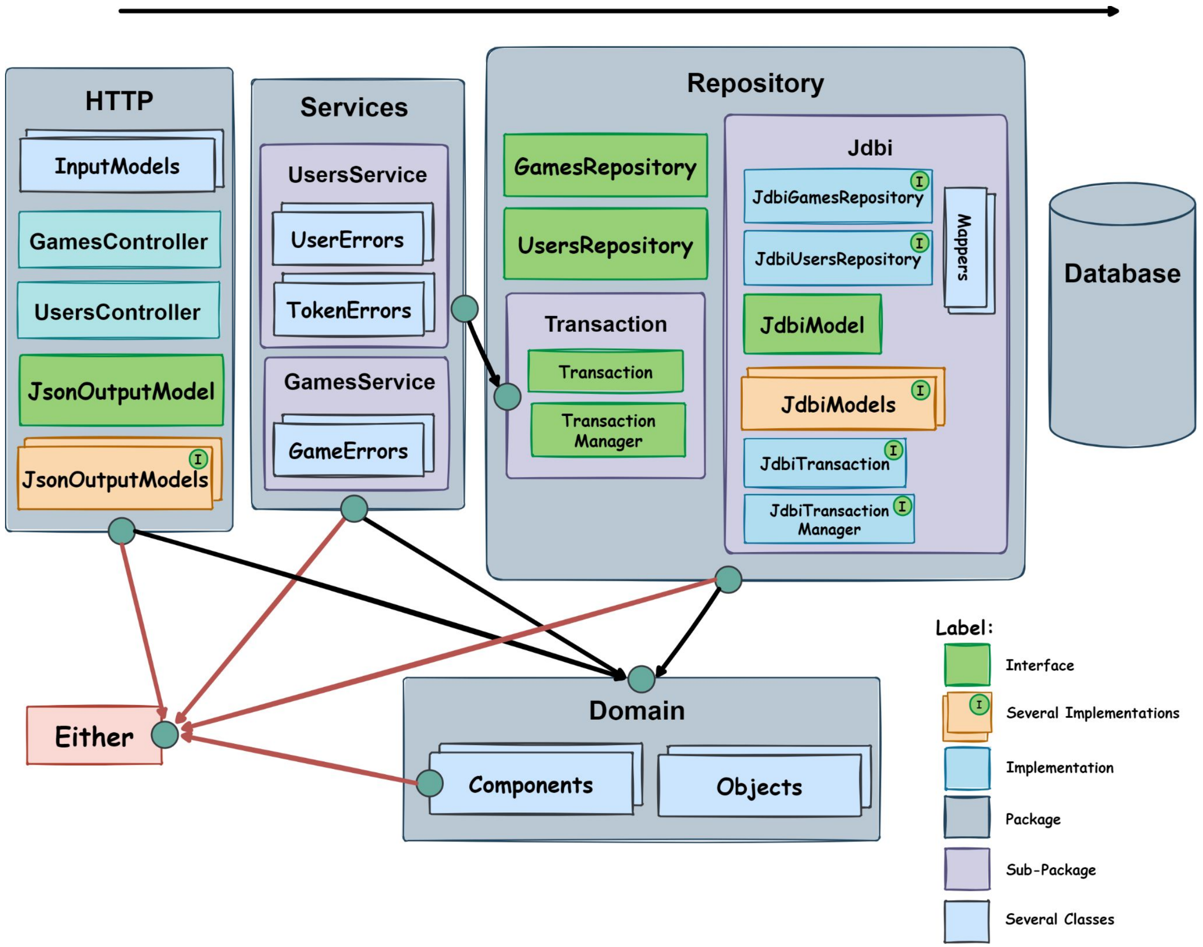
Example:

```
{
  "totalItems": 151,
  "currentPage": 1,
  "itemsPerPage": 10,
  "totalPages": 16,
  "items": [
    {
      "id": {
        "value": 5
      },
      "username": {
        "value": "user5"
      },
      "email": {
        "value": "user5@example.com"
      },
      "points": {
        "value": 6122
      },
      "rank": {
        "value": 1
      },
      "gamesPlayed": {
        "value": 10
      },
```

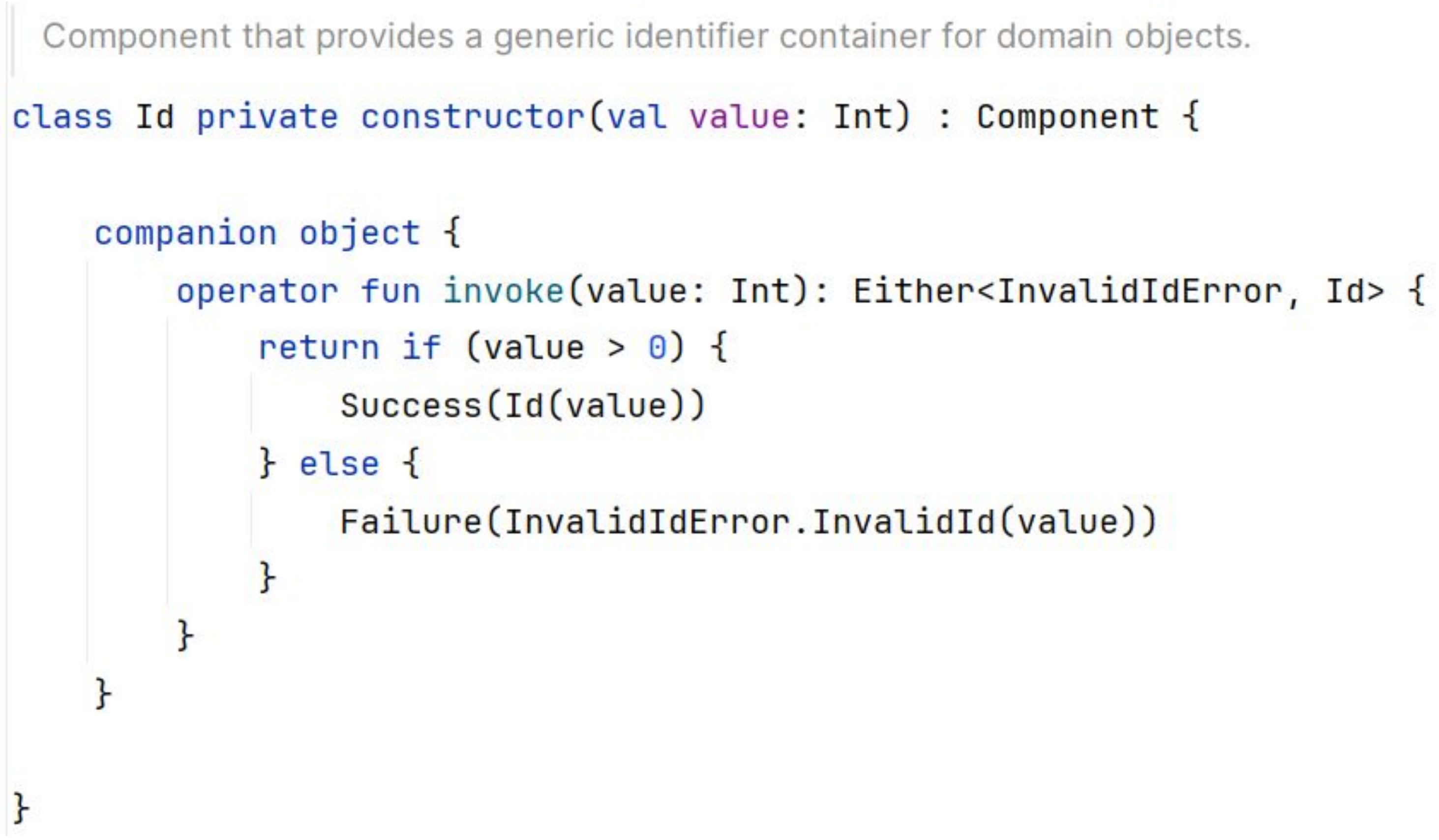
Show Backend API Documentation


Backend Implementation
1 Spring MVC
2 Application Architecture
3 Implementation Details
4 Implementation Challenges
5 Further Improvements


HttpServletRequest
RequestToken Processor
AuthenticatedUser ArgumentResolver
Servlet Server Tomcat
LoggingFilter
RequestIdFilter
Servlet
Authentication Interceptor
Controller
Handler
Spring MVC


HTTP
InputModels
GamesController
UsersController
JsonOutputModel
JsonOutputModels
Services
UsersService
UserErrors
TokenErrors
GamesService
GameErrors
Repository
GamesRepository
UsersRepository
Transaction
Transaction
Transaction Manager
Jdbi
JdbiGamesRepository
JdbiUsersRepository
Mappers
JdbiModel
JdbiModels
JdbiTransaction
JdbiTransaction Manager
Database
Either
Domain
Components
Objects
Label:
Interface
Several Implementations
Implementation
Package
Sub-Package
Several Classes

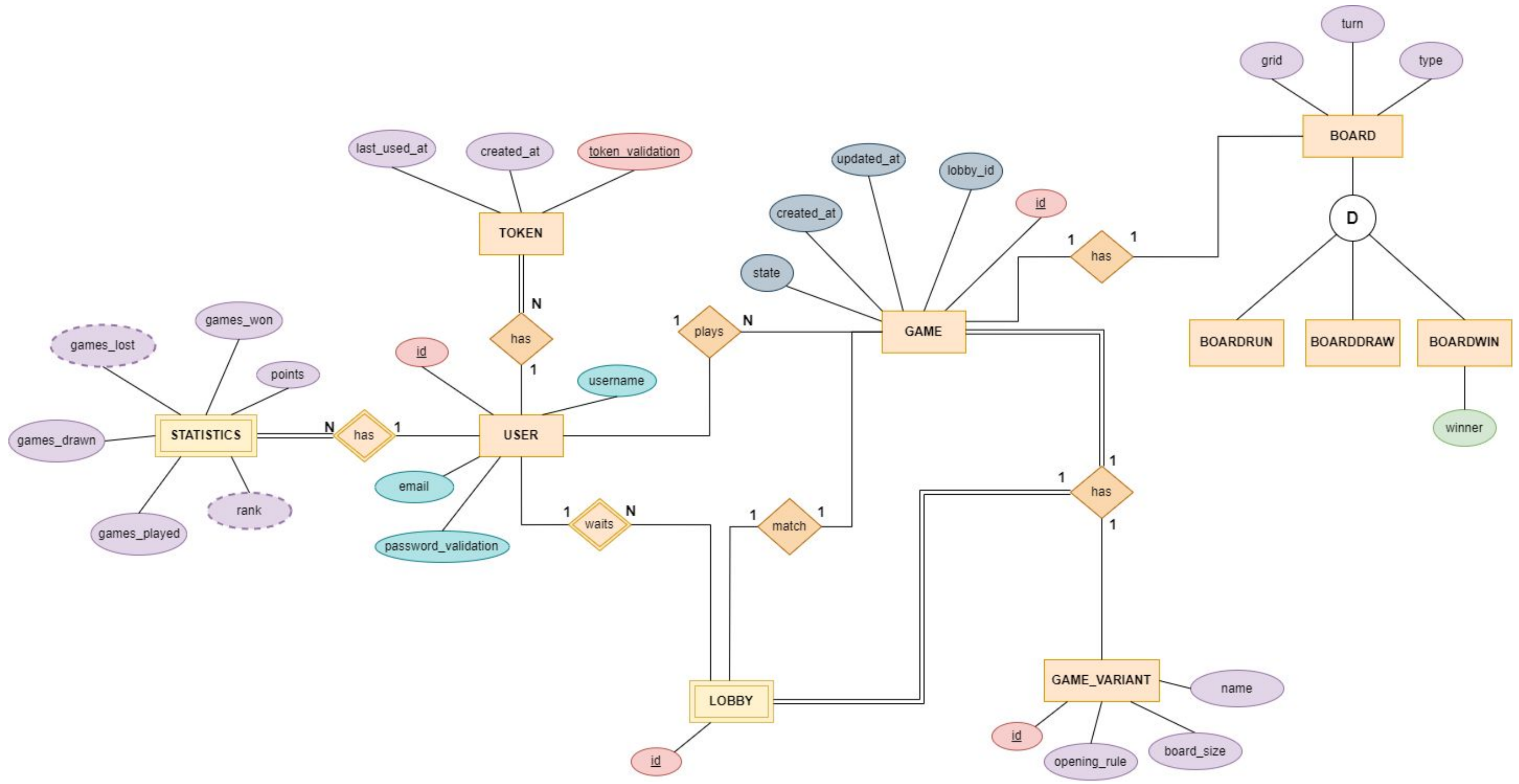
Components



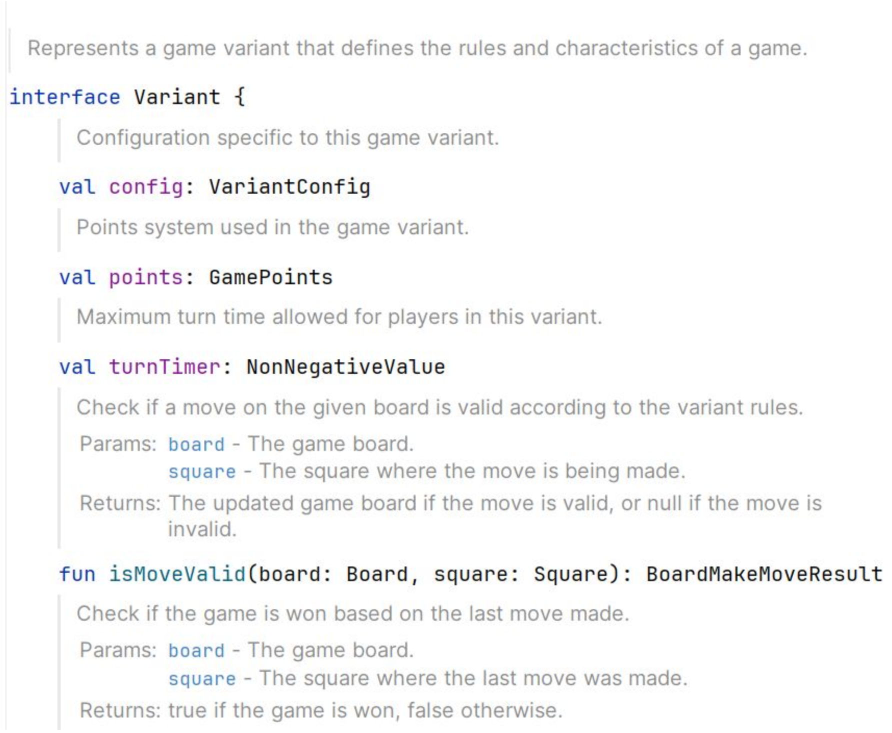
> Component that provides a generic identifier container for domain objects.

```kotlin
class Id private constructor(val value: Int) : Component {

    companion object {
        operator fun invoke(value: Int): Either<InvalidIdError, Id> {
            return if (value > 0) {
                Success(Id(value))
            } else {
                Failure(InvalidIdError.InvalidId(value))
            }
        }
    }

}
```

Extended Entity-Relation Diagram



Implementation Details



Represents a game variant that defines the rules and characteristics of a game.

```
interface Variant {
```

Configuration specific to this game variant.

```
    val config: VariantConfig
```

Points system used in the game variant.

```
    val points: GamePoints
```

Maximum turn time allowed for players in this variant.

```
    val turnTimer: NonNegativeValue
```

Check if a move on the given board is valid according to the variant rules.

Params: `board` - The game board.
`square` - The square where the move is being made.

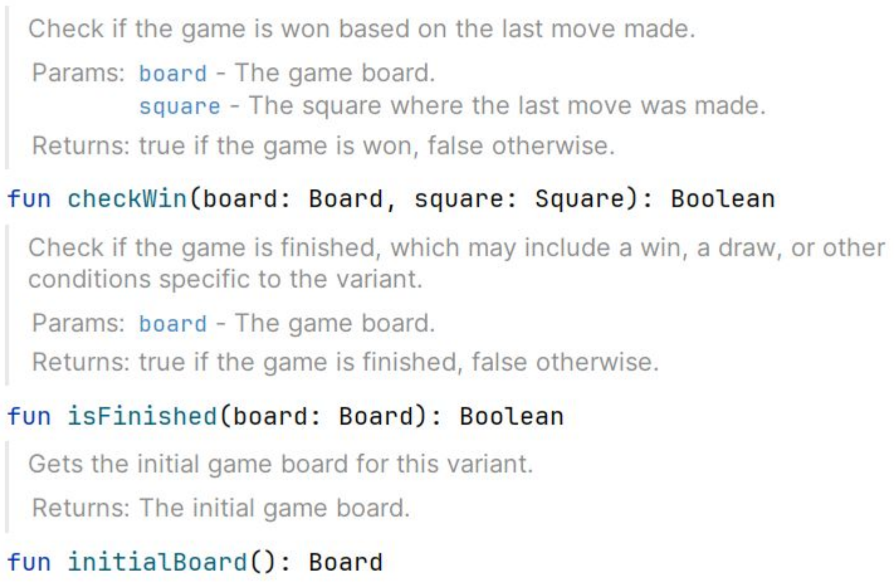
Returns: The updated game board if the move is valid, or null if the move is invalid.

```
    fun isMoveValid(board: Board, square: Square): BoardMakeMoveResult
```

Check if the game is won based on the last move made.

Params: `board` - The game board.
`square` - The square where the last move was made.

Returns: true if the game is won, false otherwise.



Check if the game is won based on the last move made.

Params: `board` - The game board.
`square` - The square where the last move was made.

Returns: true if the game is won, false otherwise.

```
fun checkWin(board: Board, square: Square): Boolean
```

Check if the game is finished, which may include a win, a draw, or other conditions specific to the variant.

Params: `board` - The game board.

Returns: true if the game is finished, false otherwise.

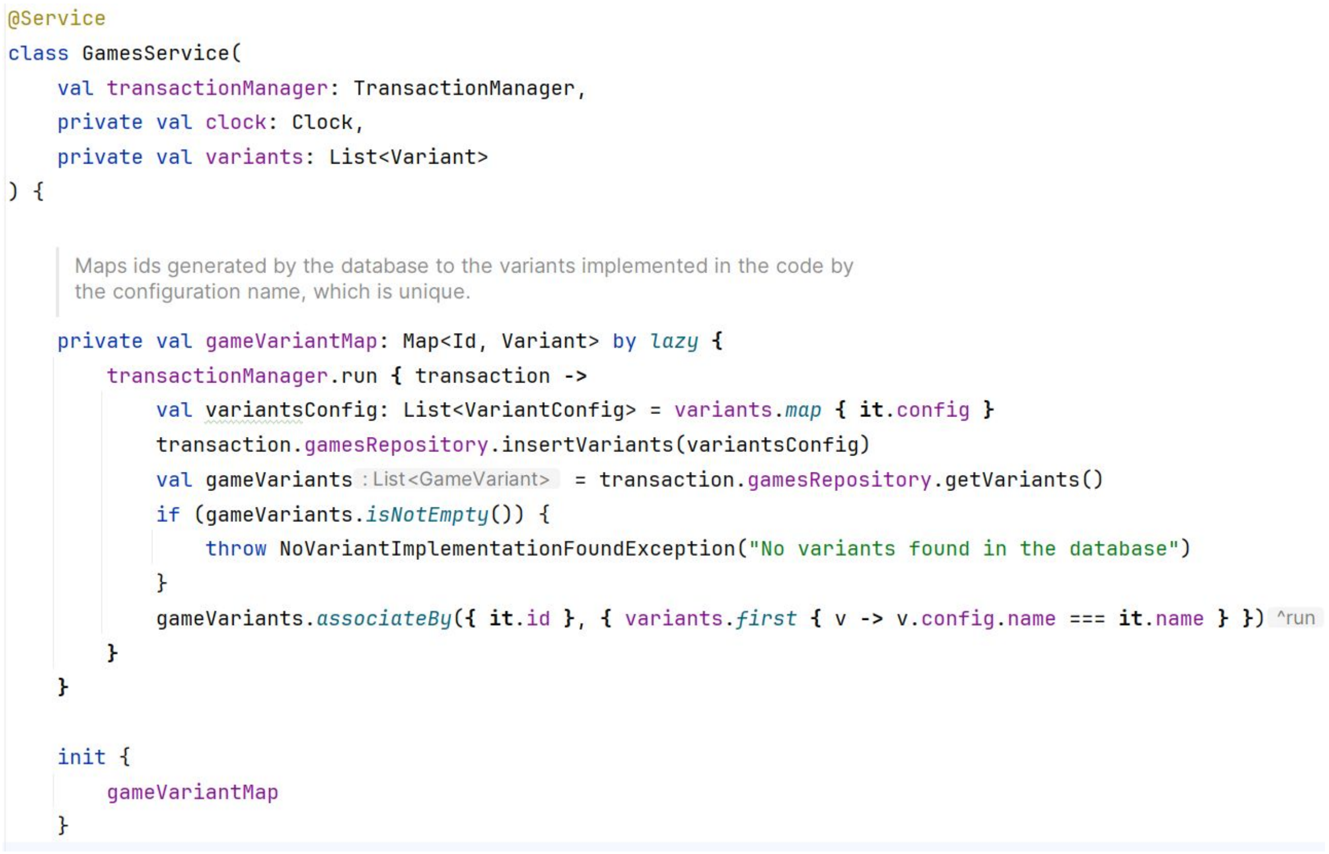
```
fun isFinished(board: Board): Boolean
```

Gets the initial game board for this variant.

Returns: The initial game board.

```
fun initialBoard(): Board
```

Implementation Details



```
@Service
class GamesService(
    val transactionManager: TransactionManager,
    private val clock: Clock,
    private val variants: List<Variant>
) {

     Maps ids generated by the database to the variants implemented in the code by
     the configuration name, which is unique.

    private val gameVariantMap: Map<Id, Variant> by lazy {
        transactionManager.run { transaction ->
            val variantsConfig: List<VariantConfig> = variants.map { it.config }
            transaction.gamesRepository.insertVariants(variantsConfig)
            val gameVariants : List<GameVariant>  = transaction.gamesRepository.getVariants()
            if (gameVariants.isNotEmpty()) {
                throw NoVariantImplementationFoundException("No variants found in the database")
            }
            gameVariants.associateBy({ it.id }, { variants.first { v -> v.config.name === it.name } }) ^run
        }
    }

    init {
        gameVariantMap
    }
```

Validation

Spring Validation

Using **@Valid** and related annotations

Components

Through **Either** imbued constructors

Database Schema

Check **constraints**

Implementation Challenges

Database Design

Concurrency

Abstracting Code

Further Improvements

- Siren Media Type
- More Game Variant Implementations
- Revisit and Write More Tests
- More Operations

Postman Demo

0.1.1