```java
/**
 * Rest-Controller zum Registrieren und für den Login bzw. das Erzeugen eines
 * JWT.
 */
@RestController
@RequestMapping("/api/auth/v1")
public class AuthController {

    /**
     * Erzeugt bei erfolgreicher Authentifizierung ein JWT.
     *
     * @param authentication Spring Security Objekt mit Username und Passwort
     * @return JWT
     */
    @GetMapping("/token")
    public ResponseEntity<String> token(Authentication authentication) {
    }

    /**
     * Registriert einen neuen User.
     *
     * @param userCredentialsDto request enthält Username, Mail (optional),
Password
     */
    @PostMapping("/register")
    public ResponseEntity<Void> register(@Valid @RequestBody UserCredentialsDto
userCredentialsDto) {
    }

    /**
     * Gibt die Details zu einem User zurück.
     *
     * @param principal enthält die UserId und wird durch Spring Security im
     *                  authentication Prozess erzeugt.
     * @return Details zum User
     */
    @GetMapping("/details")
    public ResponseEntity<UserDetailsDto> details(Principal principal) {
    }
}

/**
 * Rest-Controller zum Erstellen, Betreten und Auflisten von Lobbies.
 */
@RestController
@RequestMapping("/api/lobby/v1")
public class LobbyController {

    private final LobbyService lobbyService;

    public LobbyController(LobbyService lobbyService) {
        this.lobbyService = lobbyService;
    }

    /**
```

```
54      * Erstellt eine neue Lobby.
55      *
56      * @param principal enthält die UserId und wird durch Spring Security im
57      *                  authentication Prozess erzeugt.
58      * @param request   enthält den Namen der zu erstellenden Lobby.
59      * @return neu erstellte Lobby.
60      */
61     @PostMapping("/create")
62     public ResponseEntity<Lobby> create(Principal principal, @RequestBody
    CreateRequest request) {
63     }
64
65     /**
66      * Verbinden mit einer Lobby.
67      *
68      * @param principal enthält die UserId und wird durch Spring Security im
69      *                  authentication Prozess erzeugt.
70      * @param request   enthält die LobbyId, mit der sich der User verbinden
    möchte.
71      * @return Lobby die der User beigetreten ist.
72      */
73     @PostMapping("/connect")
74     public ResponseEntity<Lobby> connect(Principal principal, @RequestBody
    ConnectRequest request) {
75     }
76
77     /**
78      * Eine Lobby verlassen.
79      *
80      * @param principal enthält die UserId und wird durch Spring Security im
81      *                  authentication Prozess erzeugt.
82      * @param request   enthält die LobbyId, die der User verlassen möchte.
83      */
84     @PostMapping("/disconnect")
85     public ResponseEntity<Void> disconnect(Principal principal, @RequestBody
    DisconnectRequest request) {
86     }
87
88     /**
89      * Gibt eine angefragte Lobby zurück.
90      *
91      * @param request enthält die LobbyId
92      * @return Lobby
93      */
94     @GetMapping("/get")
95     public ResponseEntity<Lobby> get(@RequestBody LobbyRequest request) {
96     }
97
98     /**
99      * Gibt alle bestehenden Lobbies zurück.
100     *
101     * @return Liste mit allen bestehenden Lobbies.
102     */
103    @GetMapping("/all")
104    public ResponseEntity<List<Lobby>> all() {
105    }
```

```
106 }
107
108 /**
109  * Rest-Controller für das Quiz
110  */
111 @RestController
112 @RequestMapping("/v1")
113 public class QuizController {
114
115     /**
116      * Erstellt eine Quiz-Session.
117      *
118      * @param request enthält die LobbyId und die PlayerId's.
119      * @return Quiz
120      */
121     @PostMapping("/create")
122     public ResponseEntity<Quiz> create(@RequestBody CreateRequest request) {
123     }
124
125     /**
126      * Gib eine angefragte Quiz-Session zurück.
127      *
128      * @param request enthält die QuizId
129      * @return Quiz
130      */
131     @GetMapping("/get")
132     public ResponseEntity<Quiz> get(@RequestBody QuizRequest request) {
133     }
134 }
```

WebSockets

Lobby

- Endpunkt zum Verbinden

```
ws://<host>/lobby-websocket
```

- Endpunkt zum Abonnieren für neue Lobbies

```
ws://<host>/topic/new-lobby
```

  - Message on Subscribe: Json Array mit allen Lobbies
  - Message

```
{
    "id": "cc884343-578b-4c95-9ef3-541d311c8682",
    "name": "TestLobby",
    "players": [
        {
            "userId": "7cb353a2-c35d-4560-958a-a6ec6ceae50b"
        }
    ]
}
```

- Endpunkt zum Abonnieren für Änderungen in einer Lobby

```
ws://<host>/topic/lobby/<lobby-UUID>
```

  - Message on Subscribe: aktueller State der abonnierten Lobby
  - Message

```
{
    "id": "cc884343-578b-4c95-9ef3-541d311c8682",
    "name": "TestLobby",
    "players": [
        {
            "userId": "7cb353a2-c35d-4560-958a-a6ec6ceae50b"
            "status": "ready"
```

```
        },
        {
            "userId": "1db739fe-6054-4567-b443-2e8e38822068"
            "status": "wait"
        }
    ]
}
```

- Endpunkt zum Mitteilen, dass der Spieler bereit ist, oder nicht mehr bereit ist

```
ws://<host>/app/lobby/<lobby-UUID>/status
```

  - Message

```
{
    "status": "ready"
}

{
    "status": "wait"
}
```

- Endpunkt zum Abonnieren, der über den Start oder Abbruch eines Spiels informiert

```
ws://<host>/topic/lobby/<lobby-UUID>/start
```

  - Message

```
{
    "status": "start"
}

{
    "status": "abort"
}
```

Quiz

- Endpunkt zum Verbinden

```
ws://<host>/quiz-websocket
```

- Endpunkt zum Senden einer Antwort

```
ws://<host>/app/quiz/<quiz-UUID>/answer
```

  - Message

```
{
    "answer": "2"
}
```

- Endpunkt zum Abonnieren für Änderungen im Spiel

```
ws://<host>/topic/quiz/<quiz-UUID>
```

  - Message

```
{
    "id": "cc884343-578b-4c95-9ef3-541d311c8682",
    "round": "2",
    "timeLeft" "10",
    "players": [
        {
            "userId": "7cb353a2-c35d-4560-958a-a6ec6ceae50b"
            "status": "ready"
        },
        {
            "userId": "1db739fe-6054-4567-b443-2e8e38822068"
            "status": "wait"
        }
    ]
}
```