MythBusters Software Design Document

Batuhan Efe Alkış, Doruk Demirci, Gökçen Enli, Buğra Işın, Zeynep Yavuz

June 30, 2025

Software Design Document

Use Case: User Creation with Builder Pattern

MythBusters Team

July 1, 2025

Contents

1 System Overview

The MythBusters application is a gamified web platform aimed at educating users about health myths through interactive gameplay and user engagement. One essential component of this system is the user profile creation process, which is the entry point for personalizing user experiences.

2 System Context

The user creation use case is initiated through a REST API endpoint. When a new user attempts to register, their input is collected via a `ProfileCreateDTO` object. The backend service utilizes the Builder design pattern to build this DTO, assigns default avatars, hashes the password, and stores the data in the database.

3 Key Features and Functionality

- Collect user input (username, email, password) from REST client
- Validate fields using annotations like `@NotBlank`, `@Email`, `@Size`
- Use Builder Pattern to construct `ProfileCreateDTO`
- Automatically assign default avatars for Race, Balloon, and Hangman games
- Set default profile image
- Hash the password securely with `PasswordEncoder`
- Save the completed profile entity to the database

4 Assumptions and Dependencies

- Spring Boot application
- Spring Security for password encoding
- Lombok for boilerplate code generation
- Jakarta Validation API for input validation
- PostgreSQL or compatible relational database
- Default avatars with IDs 1 (Race), 3 (Balloon), and 5 (Hangman) exist in the database

5 Architectural Design

Architecture Style

Layered (MVC)

System Architecture Diagram

`Controller → Service → Repository → Database`

Rationale

Ensures separation of concerns, makes testing and maintenance easier, and clearly defines responsibilities.

6 Component Design

- **Controller:** Handles HTTP POST `/api/profiles`, receives and validates JSON request body.
- **DTO Layer:** Uses `ProfileCreateDTO` with an inner static Builder class for safe and readable construction.
- **Service Layer:** Contains business logic for profile creation.
- **Repository Layer:** Interacts with the database using `ProfileRepository` and `AvatarRepository`.

7 Data Design

Data Entities

`Profile` (id, username, email, passwordHash, profilePhoto, raceGameAvatarID, ballonGameAvatarID, hangGameAvatarID)

Data Validation

- `@NotBlank` for username, email, and password
- `@Email` for email field
- `@Size(min = 6)` for password

Data Flow

`User → Controller → DTO → Service → Entity → Repository → DB`

8 Design Patterns

Builder Pattern

Location: `ProfileCreateDTO` class

Purpose: Allows incremental construction of immutable DTOs from incoming requests.

```
ProfileCreateDTO dto = new ProfileCreateDTO.Builder()
    .username(createDTO.getUsername())
    .email(createDTO.getEmail())
    .password(createDTO.getPassword())
    .build();
```

Listing 1: Builder Usage

Benefits:

- Prevents telescoping constructors
- Enhances code readability
- Ensures only valid objects are built (internal validation)

9 Implementation Notes

During the user registration process, once the validated `ProfileCreateDTO` is constructed using the Builder Pattern, the backend service proceeds to construct the `Profile` entity, enriching it with default values. These include the default profile photo and three different avatar assignments for each game.

The implementation of this logic in the service layer ensures clean separation between data transfer and persistence logic. Below is the full code block in `ProfileService` class responsible for setting up the default profile configuration:

```
// Step 1: Create and populate the Profile entity
Profile profile = new Profile();
profile.setUsername(createDTO.getUsername());
profile.setEmail(createDTO.getEmail());
profile.setPasswordHash(hashedPassword);

// Step 2: Assign default profile photo
profile.setProfilePhoto("https://www.w3schools.com/howto/img_avatar.png
    ");

// Step 3: Assign default avatars for each game type
profile.setRaceGameAvatar(
    avatarRepository.findById(1)
        .orElseThrow(() -> new IllegalArgumentException("Default race
            avatar bulunamad "))
);
profile.setBaloonGameAvatar(
    avatarRepository.findById(3)
        .orElseThrow(() -> new IllegalArgumentException("Default baloon
            avatar bulunamad "))
);
profile.setHangmanGameAvatar(
```

```
    avatarRepository.findById(5)
        .orElseThrow(() -> new IllegalArgumentException("Default
           hangman avatar bulunamad "))
);
```

Listing 2: Default Profile Initialization and Avatar Assignment

This code ensures:

- Every user is initialized with a valid profile image and playable avatars.
- The system avoids `null` values in the `Profile` entity for these fields.
- Runtime failures are handled gracefully using `orElseThrow()`, which provides meaningful error messages if defaults are missing.

10 User Interface Design

Although this use case is primarily a JSON REST API, a minimal front-end interface is provided for demonstration.

(a) Before sign-up (b) Sign-up form (c) After sign-up

Figure 1: UI flow for the registration process

11 External Interfaces

Endpoint:

`POST /api/profiles`

Sample JSON Input:

```
{
  "username": "john_doe",
  "email": "john@example.com",
  "password": "secret123"
}
```

12 Performance Considerations

- Lightweight operation (single DB insert)
- Three avatar fetch queries
- No loops or external API calls

13 Error Handling and Logging

- Validation errors return 400 with standard Spring handling
- Runtime exceptions (e.g., avatar not found) return 500

14 Design for Testability

- Builder allows test DTO creation without JSON
- Service is testable in isolation

15 Deployment and Installation Design

- Spring Boot JAR
- `application.properties` for DB config

16 Change Log

- Initial version created on 30.06.2025
- Focused exclusively on profile creation with Builder pattern

17 Future Work / Open Issues

- Add client-controlled avatar/profile image selection
- Add profile editing
- Add endpoint for available avatars