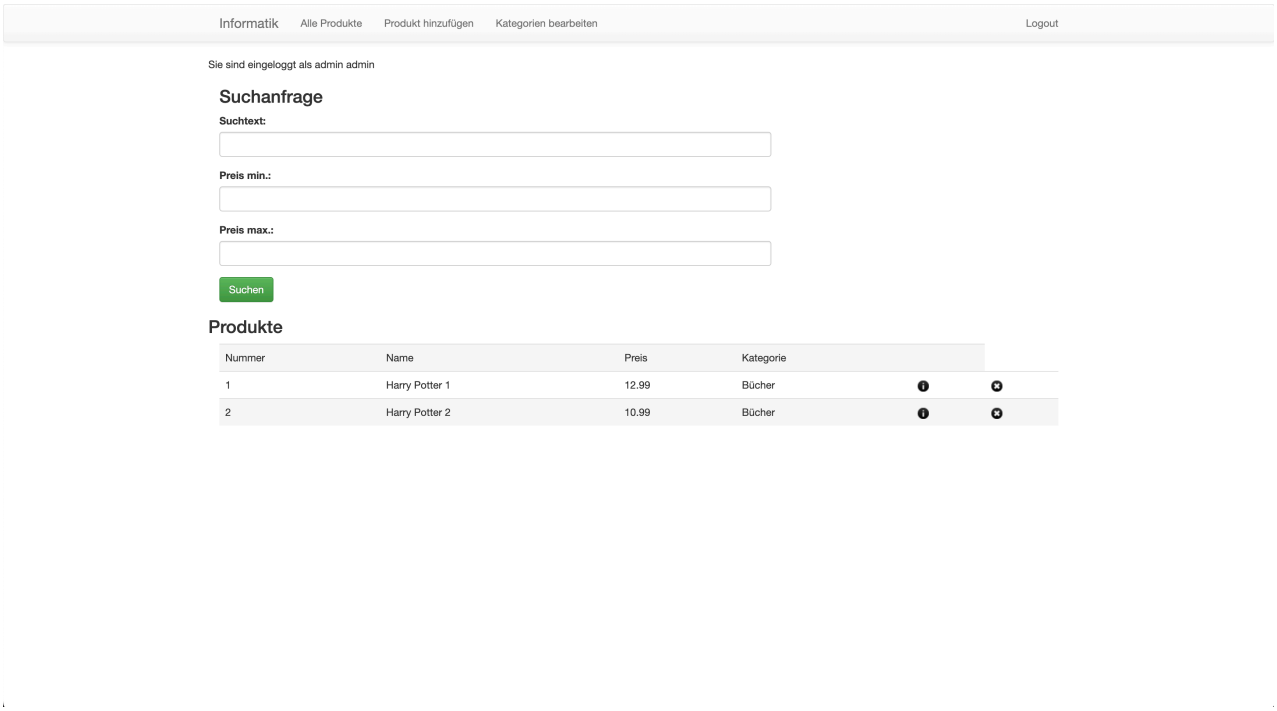
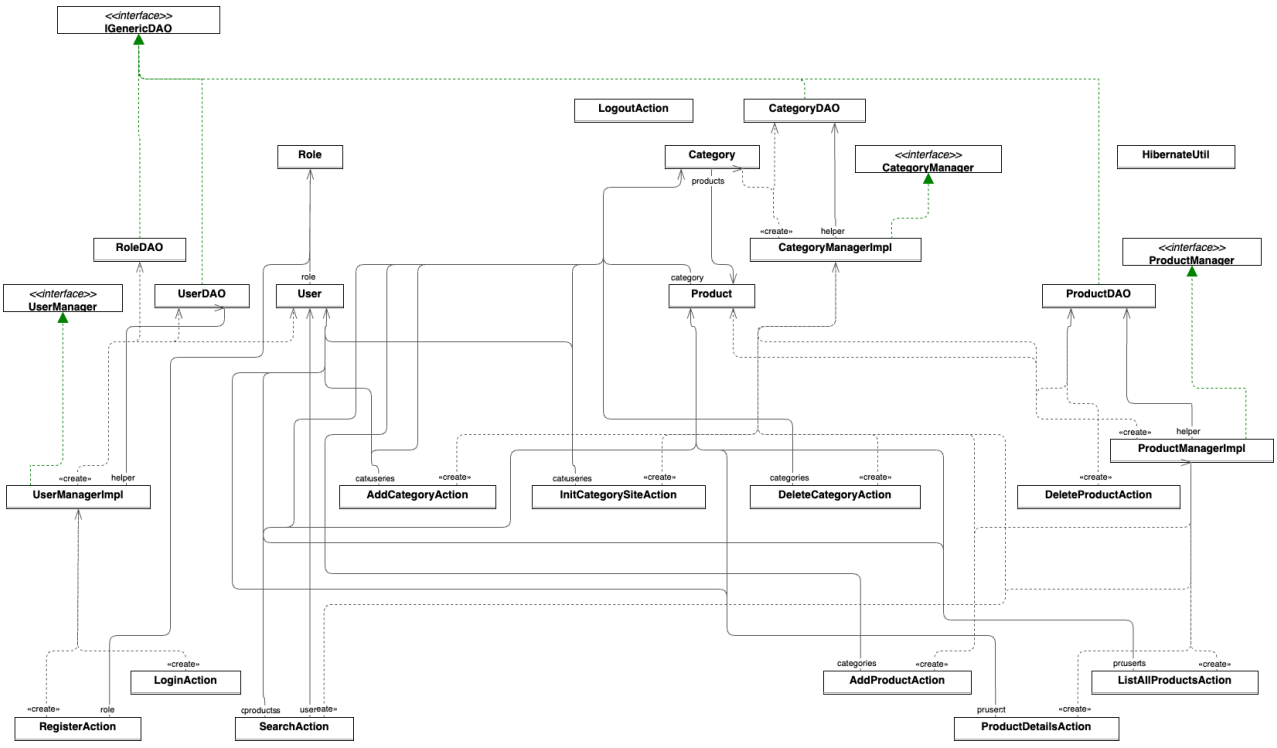
Verteilte Systeme

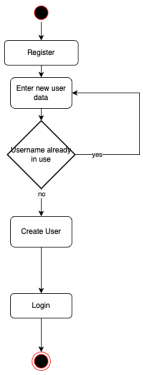
Eingelogged

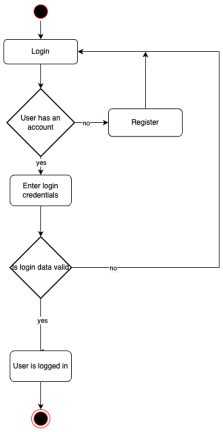


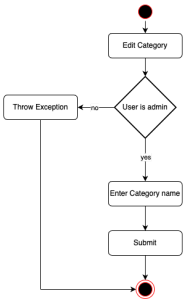
Klassen Diagram

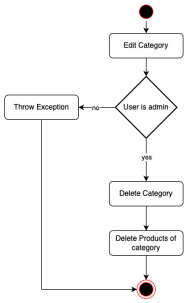


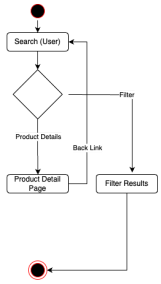
User Flow Diagram

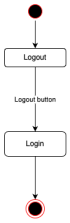


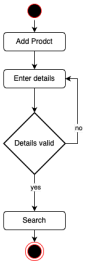


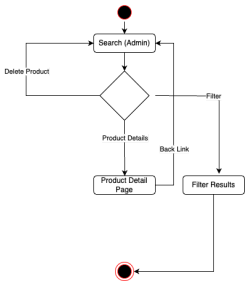




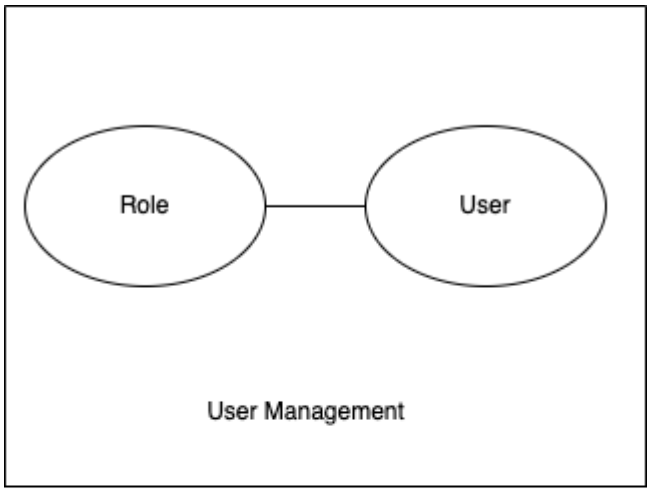


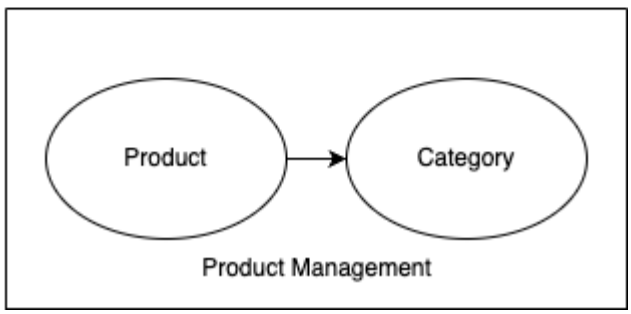






Context Map / Bounded Context





Erklärung für gewählten Bounded Context

Benutzerverwaltung:

Das System wird in Nutzerverwaltung und Produktmanagement geteilt.
Die Benutzerverwaltung ist vom Rest des Systems unabghänig, deswegen wird sie als eigener Bounded-Context abgebildet. Die zugehörigen Entitäten (Rolle und Nutzer) sind stark gekoppelt und bilden eine zusammenhängende Einheit.

Das Produktmangement, ist vom Rest des Systems unabhängig. Das Produktmangement setzt sich aus Product und Category zusammen, da diese beiden Enititäten fachlich eine Einheit bilden.

Aufgabe 2: Implementierung von Microservices

Git Commit: Link (https://github.com/hka-iwi-vislab/hska-vis-legacy/commit/68d401a0cd75562cddfc900ad4c298716fd67f28)

Implementierung mit Spring-Boot und Docker

- Verwendung von Spring-Boot für die Implementierung der Microservices.
- Verwendung von Docker für die Containerisierung der Microservices.

Anforderungen an die Microservices:

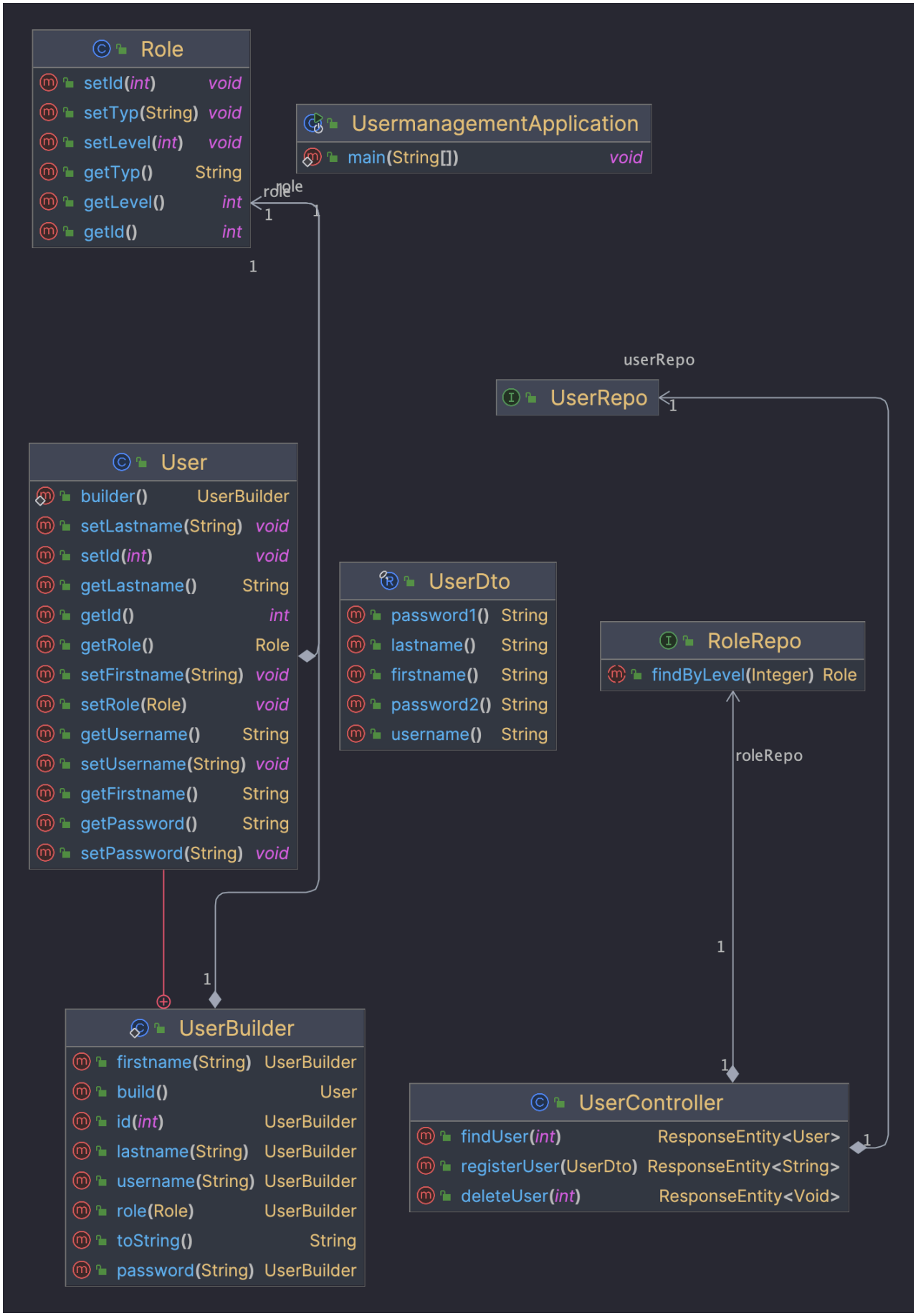
- Jeder Microservice stellt, wenn nötig, eine REST-API bereit.
- Jeder Microservice besitzt eine eigene Datenbank.

Integration in Docker Compose Konfiguration

- Integration der Microservice-Container in die Docker Compose Konfiguration aus Aufgabe 1.
- Verwendung des MySQL Containers aus Aufgabe 1 als einziges DBMS mit mehreren DBs.

Neue UML-Diagramme

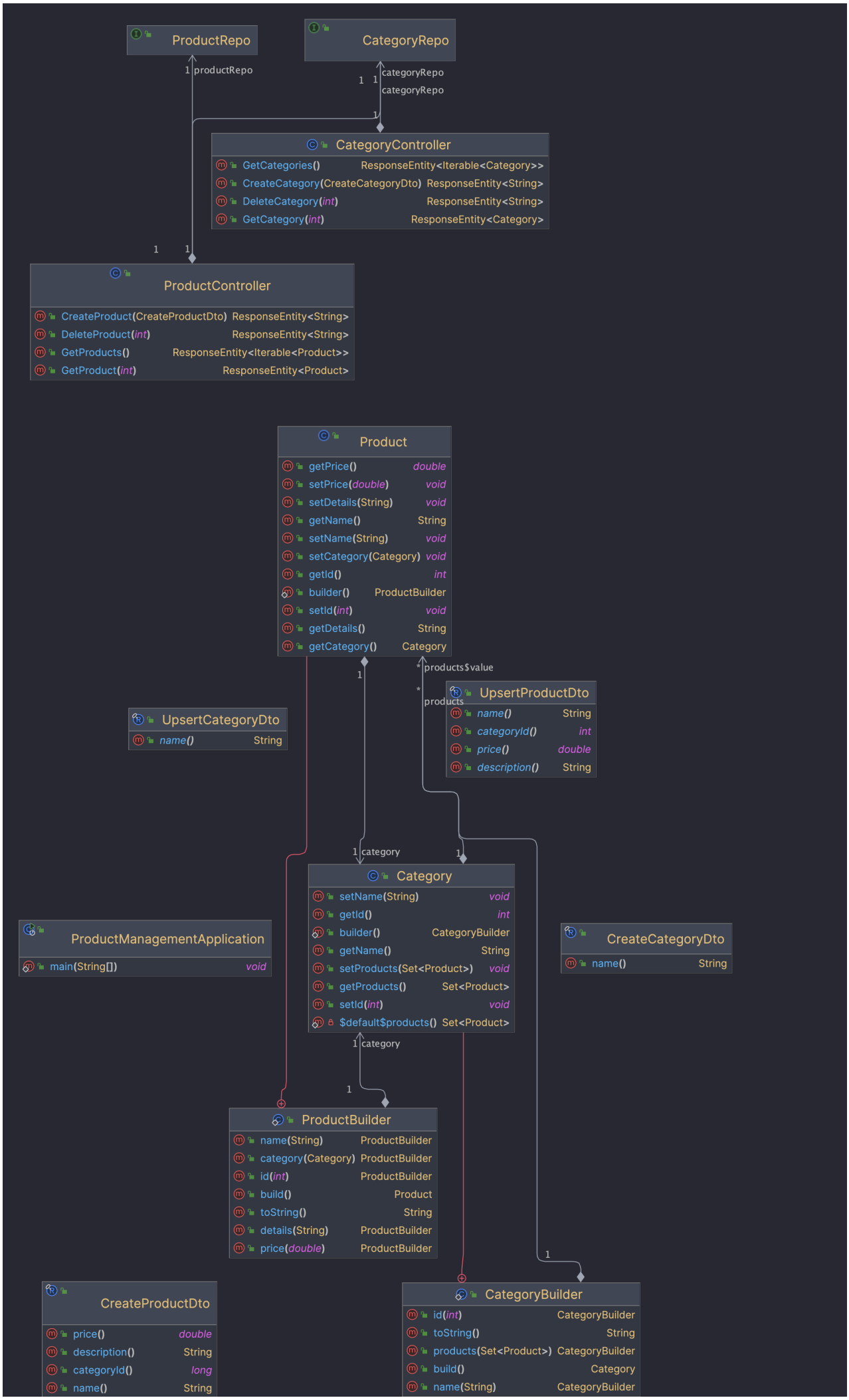
UML-Diagramm für Usermanagement



Beschreibung des Diagramms

Hier ist das UML-Diagramm für das Usermanagement. Die Klassen sind logisch getrennt, und es gibt entsprechende Verknüpfungen zwischen den relevanten Entitäten und Klassen. Die Rolle ist nun nur noch mit dem UserBuilder verknüpft, während der UserDTO abgekapselt ist. Die Klassen RoleRepository und UserController sind miteinander verknüpft, wobei der UserController das UserRepository verwendet. Klassen wie Category wurden aus diesem Kontext entfernt, da sie zum Produktmanagement gehören und nichts mit Benutzern und Rollen zu tun haben.

UML-Diagramm für Produktmanagement


ProductRepo
CategoryRepo
productRepo
categoryRepo
categoryRepo
CategoryController
GetCategories() ResponseEntity<Iterable<Category>>
CreateCategory(CreateCategoryDto) ResponseEntity<String>
DeleteCategory(int) ResponseEntity<String>
GetCategory(int) ResponseEntity<Category>
ProductController
CreateProduct(CreateProductDto) ResponseEntity<String>
DeleteProduct(int) ResponseEntity<String>
GetProducts() ResponseEntity<Iterable<Product>>
GetProduct(int) ResponseEntity<Product>
Product
getPrice() double
setPrice(double) void
setDetails(String) void
getName() String
setName(String) void
setCategory(Category) void
getId() int
builder() ProductBuilder
setId(int) void
getDetails() String
getCategory() Category
products$value
products
UpsertProductDto
name() String
categoryId() int
price() double
description() String
UpsertCategoryDto
name() String
category
Category
setName(String) void
getId() int
builder() CategoryBuilder
getName() String
setProducts(Set<Product>) void
getProducts() Set<Product>
setId(int) void
$default$products() Set<Product>
ProductManagementApplication
main(String[]) void
CreateCategoryDto
name() String
ProductBuilder
name(String) ProductBuilder
category(Category) ProductBuilder
id(int) ProductBuilder
build() Product
toString() String
details(String) ProductBuilder
price(double) ProductBuilder
CreateProductDto
price() double
description() String
categoryId() long
name() String
CategoryBuilder
id(int) CategoryBuilder
toString() String
products(Set<Product>) CategoryBuilder
build() Category
name(String) CategoryBuilder

Beschreibung des Diagramms

Hier ist das UML-Diagramm für das Produktmanagement. Die Klassen sind logisch getrennt, und es gibt entsprechende Verknüpfungen zwischen den relevanten Entitäten und Klassen. Hier finden sich die abgekapselten Klassen UserCategoryDTO, ProductManagementApplication, CreateProductDTO und CreateCategoryDTO. Sie sind vollständig abgekapselt. Die zusammenhängenden Teile des Codes sind logisch miteinander verbunden, wo dies sinnvoll ist. Das Product hat beispielsweise einen ProductBuilder und eine Category, die den mit der Kategorie verknüpften CategoryBuilder verwendet, während die Kategorie den mit dem Produkt verbundenen ProductBuilder verwendet. Insgesamt ist der Code deutlich besser getrennt.