Quarkus MicroServices

#### Book Service  
Dependencies:

<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-rest</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-rest-jsonb</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-hibernate-orm-panache</artifactId>  
</dependency>  
*<!-- H2 Database -->*<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-jdbc-h2</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-arc</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-hibernate-orm</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-junit5</artifactId>  
 <scope>test</scope>  
</dependency>  
<dependency>  
 <groupId>io.rest-assured</groupId>  
 <artifactId>rest-assured</artifactId>  
 <scope>test</scope>  
</dependency>

Application Properties

quarkus.datasource.db-kind=h2  
quarkus.datasource.jdbc.url=jdbc:h2:mem:testdb;DB\_CLOSE\_DELAY=-1  
quarkus.datasource.username=sa  
quarkus.datasource.password=sa  
quarkus.hibernate-orm.database.generation=drop-and-create  
quarkus.hibernate-orm.log.sql=true  
  
quarkus.datasource.devservices.enabled=false

1. `Book.java` (Entity Class)  
```java  
package com.example;  
  
import io.quarkus.hibernate.orm.panache.PanacheEntity;  
import jakarta.persistence.Entity;  
  
@Entity  
public class Book extends PanacheEntity {  
 public String title;  
 public String author;  
  
 public Book() {}  
  
 public Book(String title, String author) {  
 this.title = title;  
 this.author = author;  
 }  
}  
```  
\*\*Explanation\*\*:  
- Defines the `Book` entity with `title` and `author` as properties.  
  
2. `BookResource.java` (REST Endpoint Class)  
```java  
package com.example;  
  
import jakarta.transaction.Transactional;  
import jakarta.ws.rs.\*;  
import jakarta.ws.rs.core.MediaType;  
import java.util.List;  
  
@Path("/books")  
@Produces(MediaType.APPLICATION\_JSON)  
@Consumes(MediaType.APPLICATION\_JSON)  
public class BookResource {  
  
 @GET  
 public List<Book> getAllBooks() {  
 return Book.listAll();  
 }  
  
 @POST  
 @Transactional  
 public void addBook(Book book) {  
 book.persist();  
 }  
  
 @GET  
 @Path("/{id}")  
 public Book getBookById(@PathParam("id") Long id) {  
 return Book.findById(id);  
 }  
  
 @GET  
 @Path("/search")  
 public List<Book> searchByAuthor(@QueryParam("author") String author) {  
 return Book.list("author", author);  
 }  
}  
```  
\*\*Explanation\*\*:  
- \*\*Methods\*\*:  
 - `getAllBooks()`: Retrieves all books.  
 - `addBook(Book book)`: Adds a new book.  
 - `getBookById(Long id)`: Fetches a book by ID.  
 - `searchByAuthor(String author)`: Searches books by author.  
  
### 4. application.properties for Both Services  
  
\*\*Author Service (`author-service/src/main/resources/application.properties`)\*\*:  
```properties  
quarkus.http.port=8081  
quarkus.rest-client.book-service.url=http://localhost:8082  
```  
  
\*\*Book Service (`book-service/src/main/resources/application.properties`)\*\*:  
```properties  
quarkus.http.port=8082  
```  
  
### 5. Dependencies (`pom.xml`)  
  
\*\*Common Dependencies\*\*:  
- `quarkus-resteasy-reactive`  
- `quarkus-resteasy-reactive-jackson`  
- `quarkus-hibernate-orm-panache`  
- `quarkus-rest-client`  
  
### 6. Postman Calls  
  
\*\*Author Service\*\*:  
- \*\*Get all authors\*\*:  
 ```  
 GET http://localhost:8081/authors  
 ```  
- \*\*Add a new author\*\*:  
 ```  
 POST http://localhost:8081/authors  
 Body (JSON):  
 {  
 "name": "Author Name",  
 "country": "Country Name"  
 }  
 ```  
- \*\*Get books for an author\*\*:  
 ```  
 GET http://localhost:8081/authors/{id}/books  
 ```  
  
\*\*Book Service\*\*:  
- \*\*Get all books\*\*:  
 ```  
 GET http://localhost:8082/books  
 ```  
- \*\*Add a new book\*\*:  
 ```  
 POST http://localhost:8082/books  
 Body (JSON):  
 {  
 "title": "Book Title",  
 "author": "Author Name"  
 }  
 ```  
- \*\*Search books by author\*\*:  
 ```  
 GET http://localhost:8082/books/search?author=Author Name  
 ```

### 1. Overview of the Project  
The project consists of two services:  
- \*\*Author Service\*\*: Manages authors and provides endpoints for CRUD operations and retrieving books written by an author.  
- \*\*Book Service\*\*: Manages books and provides endpoints for CRUD operations and searching books by author.  
  
### 2. Project Structure  
```  
project-root/  
│  
├── author-service/  
│ ├── src/  
│ │ ├── main/  
│ │ │ ├── java/com/example/  
│ │ │ │ ├── Author.java  
│ │ │ │ ├── AuthorResource.java  
│ │ │ │ ├── BookServiceClient.java  
│ │ │ │ └── RestConfig.java  
│ │ │ ├── resources/  
│ │ │ │ └── application.properties  
│ │ └── pom.xml  
│  
└── book-service/  
 ├── src/  
 │ ├── main/  
 │ │ ├── java/com/example/  
 │ │ │ ├── Book.java  
 │ │ │ └── BookResource.java  
 │ │ ├── resources/  
 │ │ │ └── application.properties  
 │ └── pom.xml  
```  
  
### 3. Explanation of Code and Classes  
  
#### Author Service  
  
1. `Author.java` (Entity Class)  
```java  
package com.example;  
  
import io.quarkus.hibernate.orm.panache.PanacheEntity;  
import jakarta.persistence.Entity;  
  
@Entity  
public class Author extends PanacheEntity {  
 public String name;  
 public String country;  
  
 public Author() {} // Default constructor for JPA  
  
 public Author(String name, String country) {  
 this.name = name;  
 this.country = country;  
 }  
}  
```  
\*\*Explanation\*\*:  
- \*\*`@Entity`\*\*: Marks this class as a JPA entity.  
- \*\*`PanacheEntity`\*\*: Extends this base class to simplify entity operations.  
- \*\*Attributes\*\*: `name` and `country` represent the author's details.  
  
2. `AuthorResource.java` (REST Endpoint Class)  
```java  
package com.example;  
  
import jakarta.transaction.Transactional;  
import jakarta.ws.rs.\*;  
import jakarta.ws.rs.core.MediaType;  
import jakarta.ws.rs.core.Response;  
import org.eclipse.microprofile.rest.client.inject.RestClient;  
  
import java.util.List;  
  
@Path("/authors")  
@Produces(MediaType.APPLICATION\_JSON)  
@Consumes(MediaType.APPLICATION\_JSON)  
public class AuthorResource {  
  
 @RestClient  
 BookServiceClient bookServiceClient;  
  
 @GET  
 public List<Author> getAllAuthors() {  
 return Author.listAll();  
 }  
  
 @POST  
 @Transactional  
 public Response addAuthor(Author author) {  
 if (author == null || author.name == null || author.country == null) {  
 return Response.status(Response.Status.BAD\_REQUEST).entity("Invalid author data").build();  
 }  
 author.persist();  
 return Response.status(Response.Status.CREATED).entity(author).build();  
 }  
  
 @GET  
 @Path("/{id}")  
 public Author getAuthorById(@PathParam("id") Long id) {  
 Author author = Author.findById(id);  
 if (author == null) {  
 throw new WebApplicationException("Author not found", Response.Status.NOT\_FOUND);  
 }  
 return author;  
 }  
  
 @GET  
 @Path("/{id}/books")  
 public List<Book> getBooksForAuthor(@PathParam("id") Long authorId) {  
 Author author = Author.findById(authorId);  
 if (author == null) {  
 throw new WebApplicationException("Author not found", 404);  
 }  
 return bookServiceClient.searchByAuthor(author.name);  
 }  
}  
```  
\*\*Explanation\*\*:  
- \*\*`@Path`\*\*: Defines the base path for the REST endpoint.  
- \*\*`@RestClient`\*\*: Injects the `BookServiceClient` for calling the book service.  
- \*\*Methods\*\*:  
 - `getAllAuthors()`: Retrieves all authors.  
 - `addAuthor(Author author)`: Adds a new author with validation.  
 - `getAuthorById(Long id)`: Fetches an author by ID.  
 - `getBooksForAuthor(Long authorId)`: Calls the `BookServiceClient` to get books for an author.  
  
3. `BookServiceClient.java` (REST Client Interface)  
```java  
package com.example;  
  
import jakarta.ws.rs.GET;  
import jakarta.ws.rs.Path;  
import jakarta.ws.rs.QueryParam;  
import jakarta.ws.rs.Produces;  
import jakarta.ws.rs.core.MediaType;  
import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;  
  
import java.util.List;  
  
@RegisterRestClient(configKey = "book-service")  
@Path("/books")  
public interface BookServiceClient {  
  
 @GET  
 @Path("/search")  
 @Produces(MediaType.APPLICATION\_JSON)  
 List<Book> searchByAuthor(@QueryParam("author") String author);  
}  
```  
\*\*Explanation\*\*:  
- \*\*`@RegisterRestClient`\*\*: Registers this interface as a REST client.  
- \*\*`searchByAuthor`\*\*: Calls the `/books/search` endpoint with an author query parameter.  
  
4. `RestConfig.java` (Configuration Class)  
```java  
package com.example;  
  
import jakarta.ws.rs.ApplicationPath;  
import jakarta.ws.rs.core.Application;  
  
@ApplicationPath("/")  
public class RestConfig extends Application {}  
```  
\*\*Explanation\*\*:  
- \*\*`@ApplicationPath`\*\*: Sets the base URI path for the JAX-RS application.