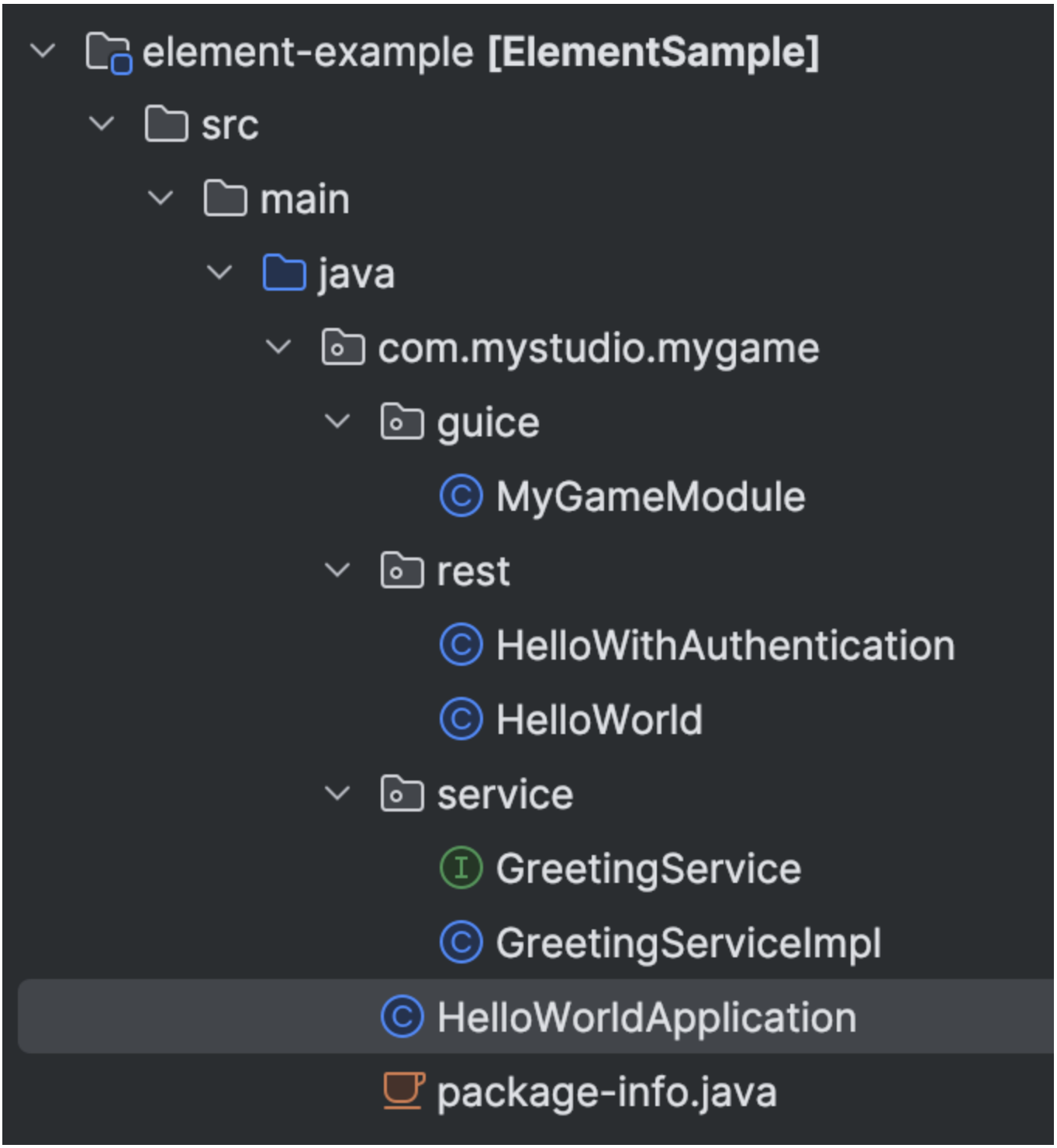
Structuring your Element

Setting Up Your Element Project

See https://github.com/NamazuStudios/element-example for a complete example. It is recommended to use this as a starting point as well.



Setting Up the Application

The Application is required to tell Jakarta which endpoints to load and expose. In the example project, that looks like this:

```
package com.mystudio.mygame;

import com.mystudio.mygame.rest.HelloWorld;
import com.mystudio.mygame.rest.HelloWithAuthentication;
import dev.getelements.elements.sdk.annotation.ElementServiceExport;
import dev.getelements.elements.sdk.annotation.ElementServiceImplementation;
import jakarta.ws.rs.core.Application;

import java.util.Set;

// Swagger OpenAPI JAX-RS resource
import io.swagger.v3.jaxrs2.integration.resources.OpenApiResource;

@ElementServiceImplementation
@ElementServiceExport(Application.class)
public class HelloWorldApplication extends Application {

    /**
     * Here we register all the classes that we want to be included in the Element.
     */
    @Override
    public Set<Class<?>> getClasses() {
        return Set.of(
                //Endpoints
                HelloWorld.class,
                HelloWithAuthentication.class,

                //Required if you want codegen to work for this
                OpenApiResource.class
        );
    }
}
```

Defining an Element with `package-info.java`

At the root of your Element project (e.g. `src/main/java/com/mystudio/mygame`), you need a `package-info.java` file.

This file tells Elements Core:

- **What package is an Element** (`@ElementDefinition`).
- **Which Guice module to use** for dependency injection (`@GuiceElementModule`).
- **Which other packages to allow injection from** (`@ElementDependency`).

Example: `package-info.java`

```
// Required annotation for an Element. Will recursively search folders from this point to include classes in the Element if recursive is true. Otherwise, you must include additional package-info.java files in child packages.
@ElementDefinition(recursive = true)

// Enables DI via Guice
@GuiceElementModule(ExampleModule.class)

// Allows injecting DAO layer from Elements Core
@ElementDependency("dev.getelements.elements.sdk.dao")

// Allows injecting Service layer from Elements Core
@ElementDependency("dev.getelements.elements.sdk.service")

package com.mystudio.mygame;

import com.mystudio.mygame.guice.ExampleModule;
import dev.getelements.elements.sdk.annotation.ElementDefinition;
import dev.getelements.elements.sdk.annotation.ElementDependency;
import dev.getelements.elements.sdk.spi.guice.annotations.GuiceElementModule;
```

This is required for your Element to be recognized and properly wired up by the Elements runtime.
Without this file, your Guice bindings and service injections won't work.

Example: Greeting Service with User Context

Let's extend the greeting example so it uses the **current logged-in user**.
The `UserService` from the SDK can fetch information about the user making the request.

- The user is identified automatically based on the session token provided in the header: `Elements-SessionSecret: <session-token>`

- Elements Core resolves the session, and `UserService.getCurrentUser()` will give you the current user.

Service (Interface)

```
package com.mystudio.mygame.service;

import dev.getelements.elements.sdk.annotation.ElementServiceExport;

@ElementServiceExport
public interface GreetingService {

    /**
     * Attempts to fetch the current user for the session header and return an appropriate
greeting
     * @return The greeting based on if a logged-in user is found
     */
    String getGreeting();
}
```

Note

It is important to add the `@ElementServiceExport` annotation to make this service discoverable within the Element.

Service (uses `UserService`)

```
package com.mystudio.mygame.service;

import dev.getelements.elements.sdk.model.user.User;
import dev.getelements.elements.sdk.service.user.UserService;
import jakarta.inject.Inject;

public class GreetingServiceImpl implements GreetingService {

    private UserService userService;
```

```
    public UserService getUserService() {
        return userService;
    }

    @Inject
    public void setUserService(final UserService userService) {
        this.userService = userService;
    }

    @Override
    public String getGreeting() {

        final User currentUser = userService.getCurrentUser();
        final boolean isLoggedIn = !User.Level.UNPRIVILEGED.equals(currentUser.getLevel());
        final String name = isLoggedIn ? currentUser.getName() : "Guest";

        return "Hello, " + name + "!";
    }
}
```

In this code:

- `UserService` is injected by Guice.
- `getCurrentUser()` returns the `User` associated with the request's `Elements-SessionSecret` header.
- If no user is logged in, we fall back to `"Guest"`.

Note

Because we set the dev.getelements.elements.auth.enabled attribute to "true" in the HelloWorldApplication, the UserService will be automatically injected with the current user. This will apply an authentication filter to every request and every service that is used in this application.

Module (binds the service)

```
package com.mystudio.mygame.guice;

import com.google.inject.AbstractModule;
```

```
import com.mystudio.mygame.service.GreetingService;
import com.mystudio.mygame.service.GreetingServiceImpl;


public class ExampleModule extends AbstractModule {

  @Override
  protected void configure() {

    bind(GreetingService.class).to(GreetingServiceImpl.class);

    // Needed in conjunction with the @ElementServiceExport annotation
    // to make this discoverable by the service locator
    expose(GreetingService.class);
  }

}
```

Endpoint (fetches the service)

```
package com.mystudio.mygame.endpoint;

import com.mystudio.mygame.service.GreetingService;
import dev.getelements.elements.sdk.spi.ElementSupplier;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Tag(name = "MyGame")
@Path("/greet")
public class GreetingEndpoint {

    private final Element element = ElementSupplier
            .getElementLocal(GreetingEndpoint.class)
            .get();

    private final GreetingService greetingService = element
```

```
            .getServiceLocator()
            .getInstance(GreetingService.class);

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Operation(
    summary = "Gets a greeting",
    description = "Checks if the session token in the header corresponds
    to at least a USER level user and returns a greeting with their name
    if so, or Guest if not."
  )
  public String greet() {
    return greetingService.getGreeting();
  }
}
```

Example Request

```
GET /greet
Elements-SessionSecret: eyJhbGciOiJIUzI1...
```

Response (if user logged in)

```
"Hello, Alice!"
```

Response (if not logged in)

```
"Hello, Guest!"
```

Summary

- `package-info.java` defines your Element and its dependencies.
- Services can inject SDK-provided classes like `UserService`.
- The **current user** is resolved automatically from the `Elements-SessionSecret` header.

- Endpoints should fetch services using `ElementSupplier.getElementLocal(...).get().getServiceLocator().getInstance(...)`.