Amarildo Kondi Follow
Nov 10, 2020 · 5 min read · Listen

Save

Clean Architecture with Spring

Clean Architecture with multimodules written in Java and the web layer build with the help of Spring.

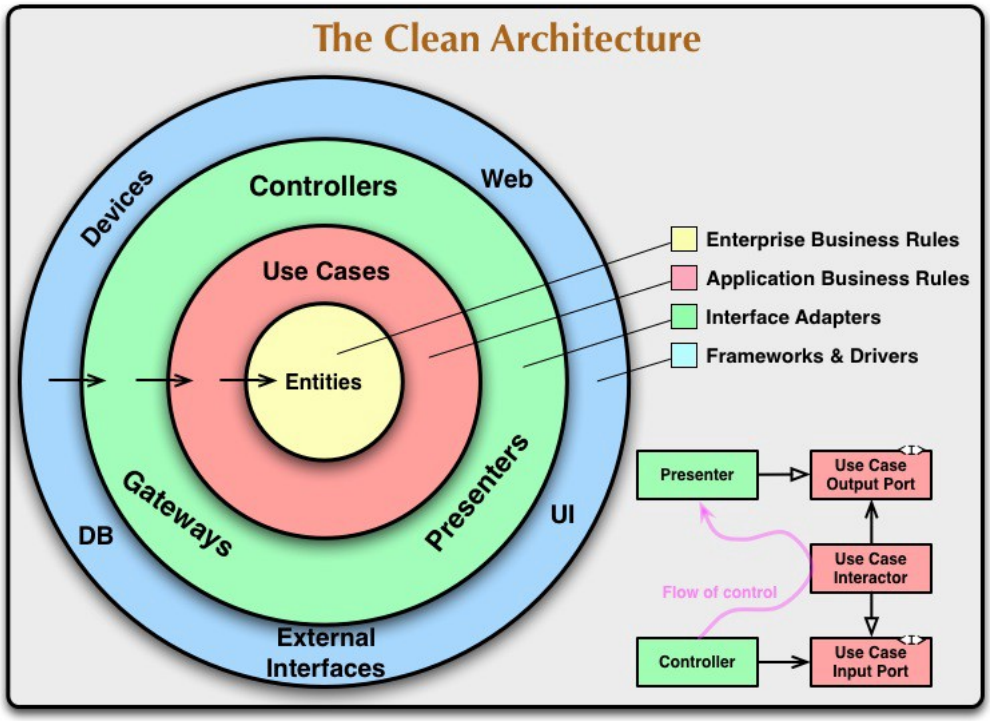
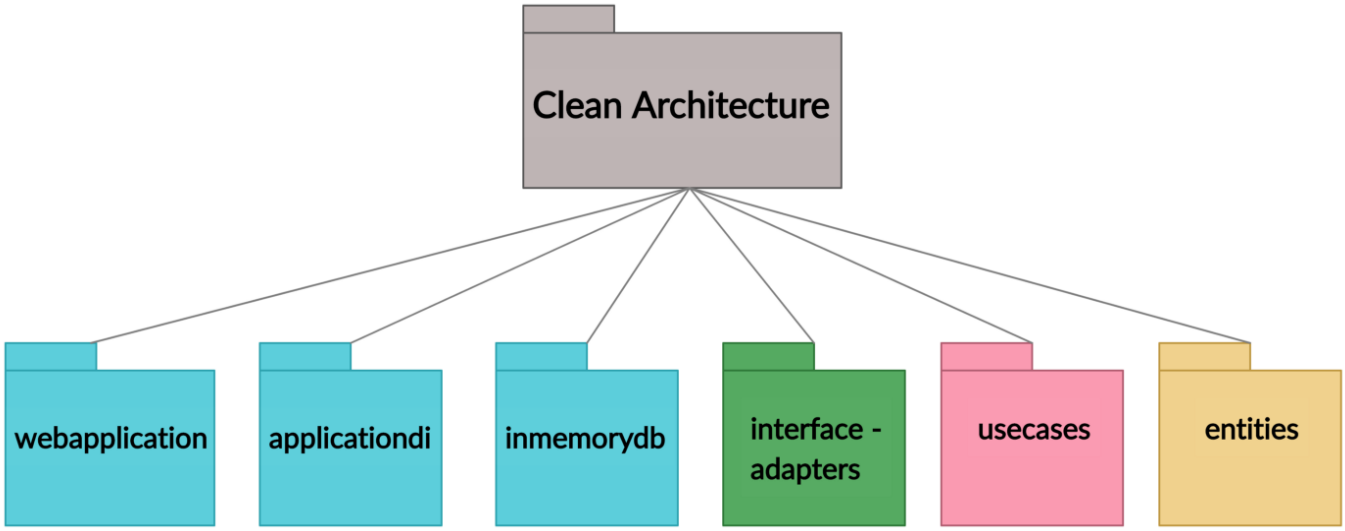


Image source by Clean Architecture Website (Unlce Bob)

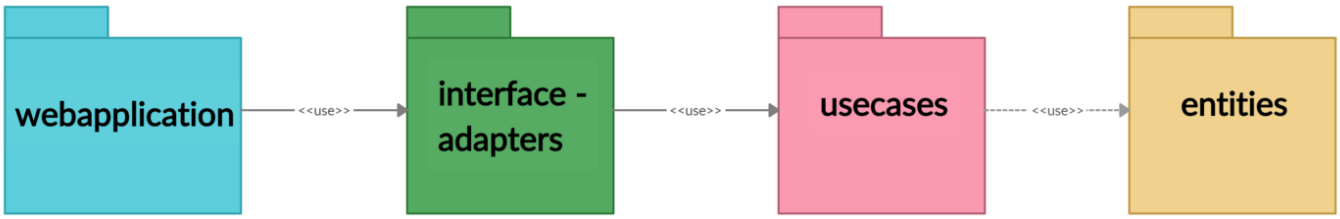
Clean Architecture main directory contains all the modules which realize the working of application.



- **usecases** module is part of ***Application Business Rules layer.***
- **entities** module is part of ***Enterprise Business Rules layer.***

Dependency direction of the modules from the outermost to the center.

Outside layer depends on the layer below it. The inner layer has no dependency on the layer above and no information for any class or method from it. Changes on the outer layer should not cause changes to the inner layer .

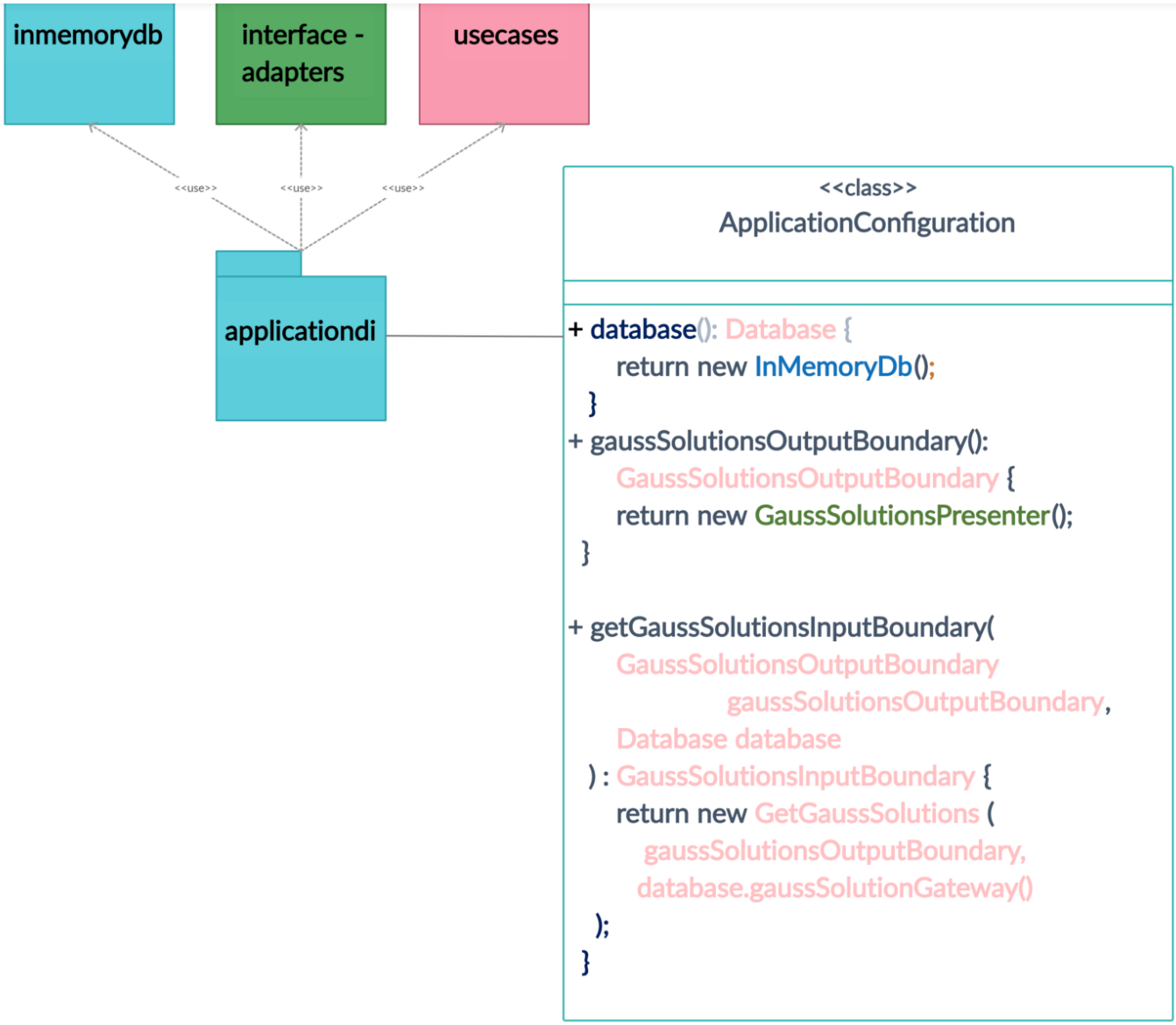


applicationdi module (application dependency injection module)

applicationdi module uses:

- **inmemorydb**
- **usecases**
- **interfaceadpters**

modules to inject the dependencies that are needed in this application.



webapplication module

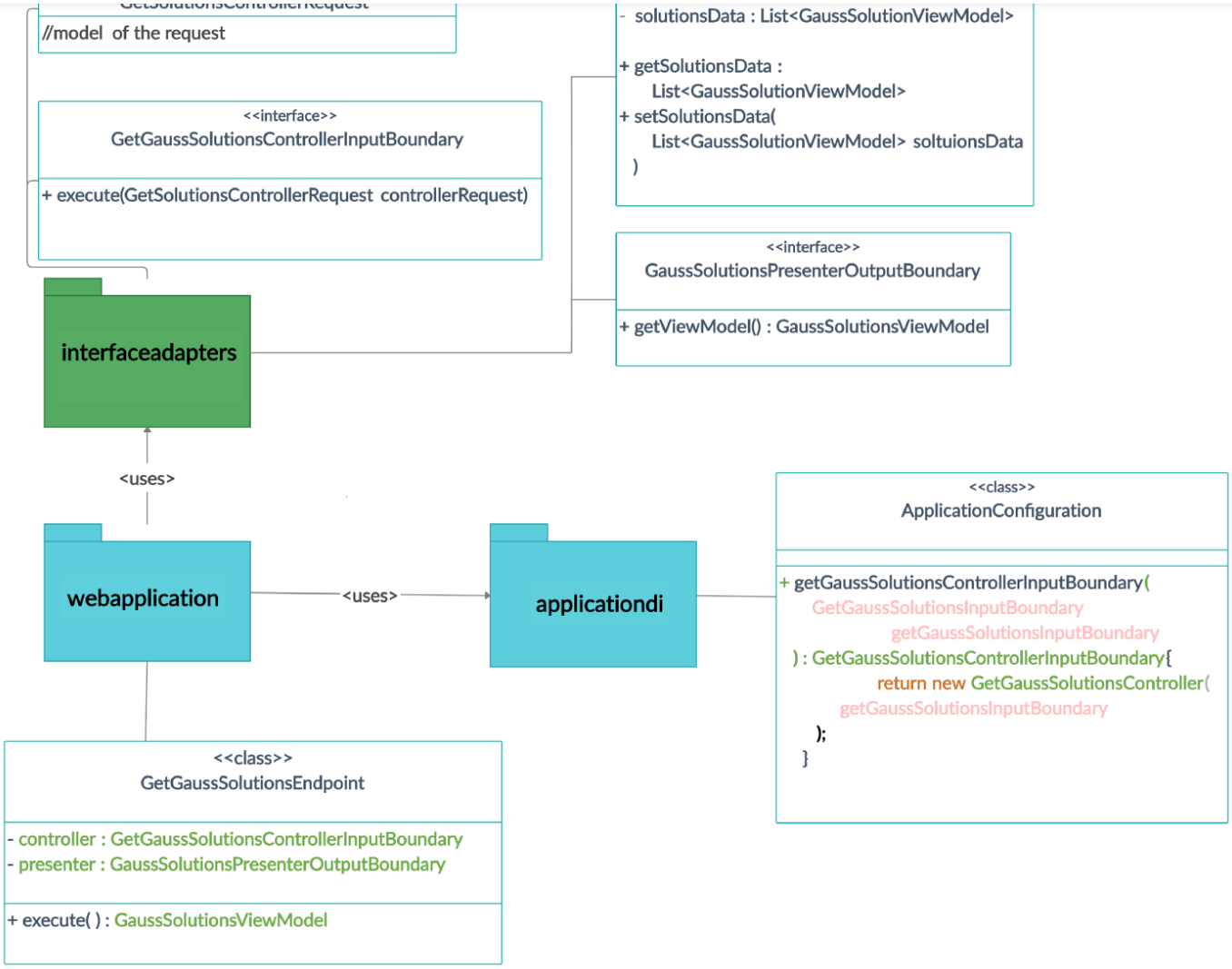
webapplication module is part of **framework and drivers layer**.

Framework and drivers layer is the outermost layer.

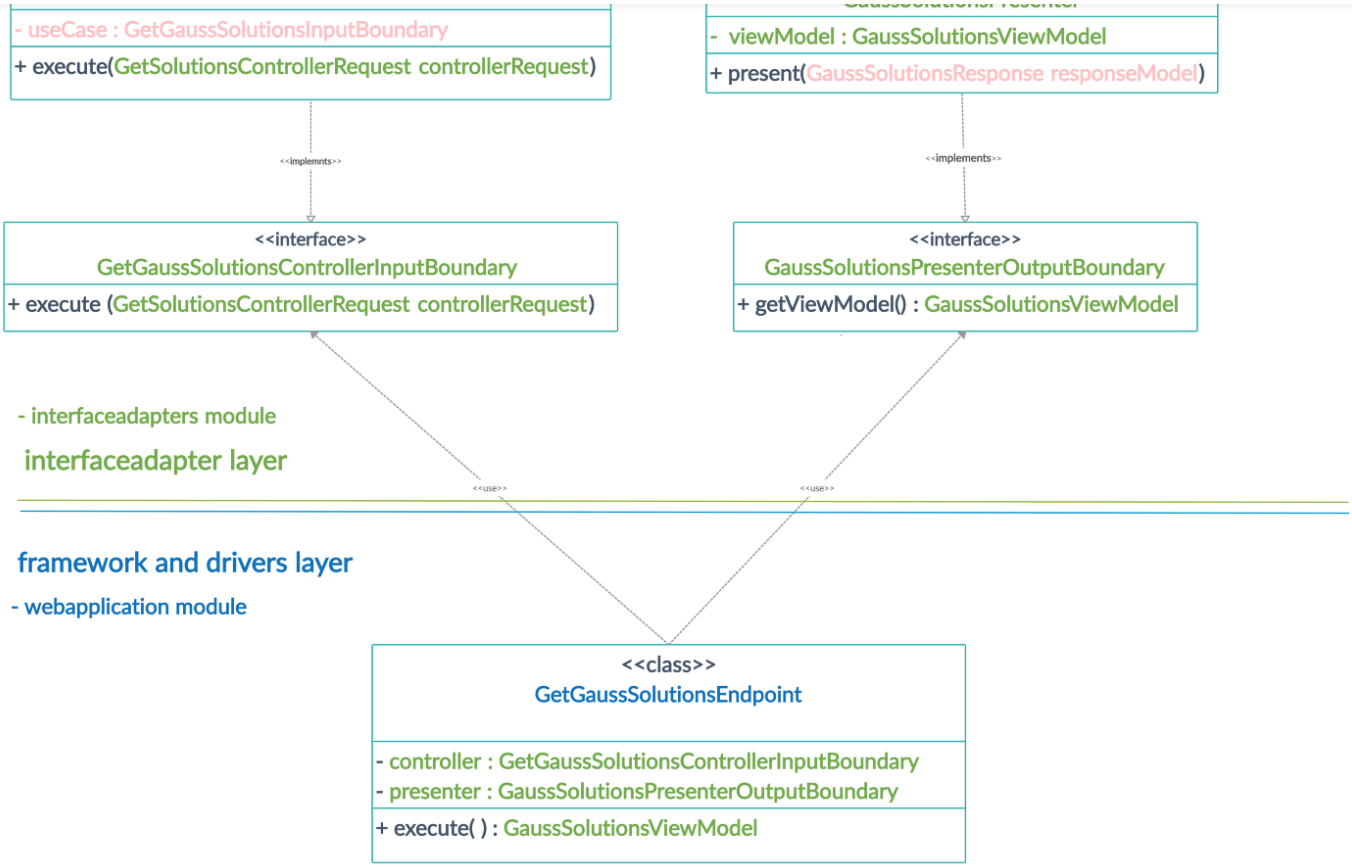
- **webapplication** module depends on the layer below it: **interface adapters**.
- **webapplication** module uses **Spring** framework. The **endpoints** which are in this layer with help of **Spring annotations** build together the entrypoints for the input.

Example schema of one case of **Gauss elimination** for linear equations with **Clean Architecture**.

This schema shows the **interaction** between **weabapplication** module and **interfaceadapters** module.



Class diagram shows the **dependencies** of **GetGaussSolutionsEndpoint.class** in **webapplication** module from the **interfaceadapters** module.



GetGaussSolutionsEndpoint.class uses two **interfaces** from **interfaceadapters** module to *cross the **boundary** between layers*.

- **GetGaussSolutionsControllerInputBoundary interface** that is **implemented** by **GetGaussSolutionsController** class which **gets** the **input** (GetSolutionsControllerRequest). **GetGaussSolutionsController** will send the **input** to the **usecase** layer to be processed, explained later in details.
- The second interface **GaussSolutionsPresenterOutputBoundary** that is **implemented** by **GaussSolutionsPresenter** class will be used by **GetGaussSolutionsEndpoint.class** to get the **output** (GaussSolutionsViewModel) and present to the web. **GaussSolutionsPresenter receives** the processed **output** from **usecase** layer, explained later in details.

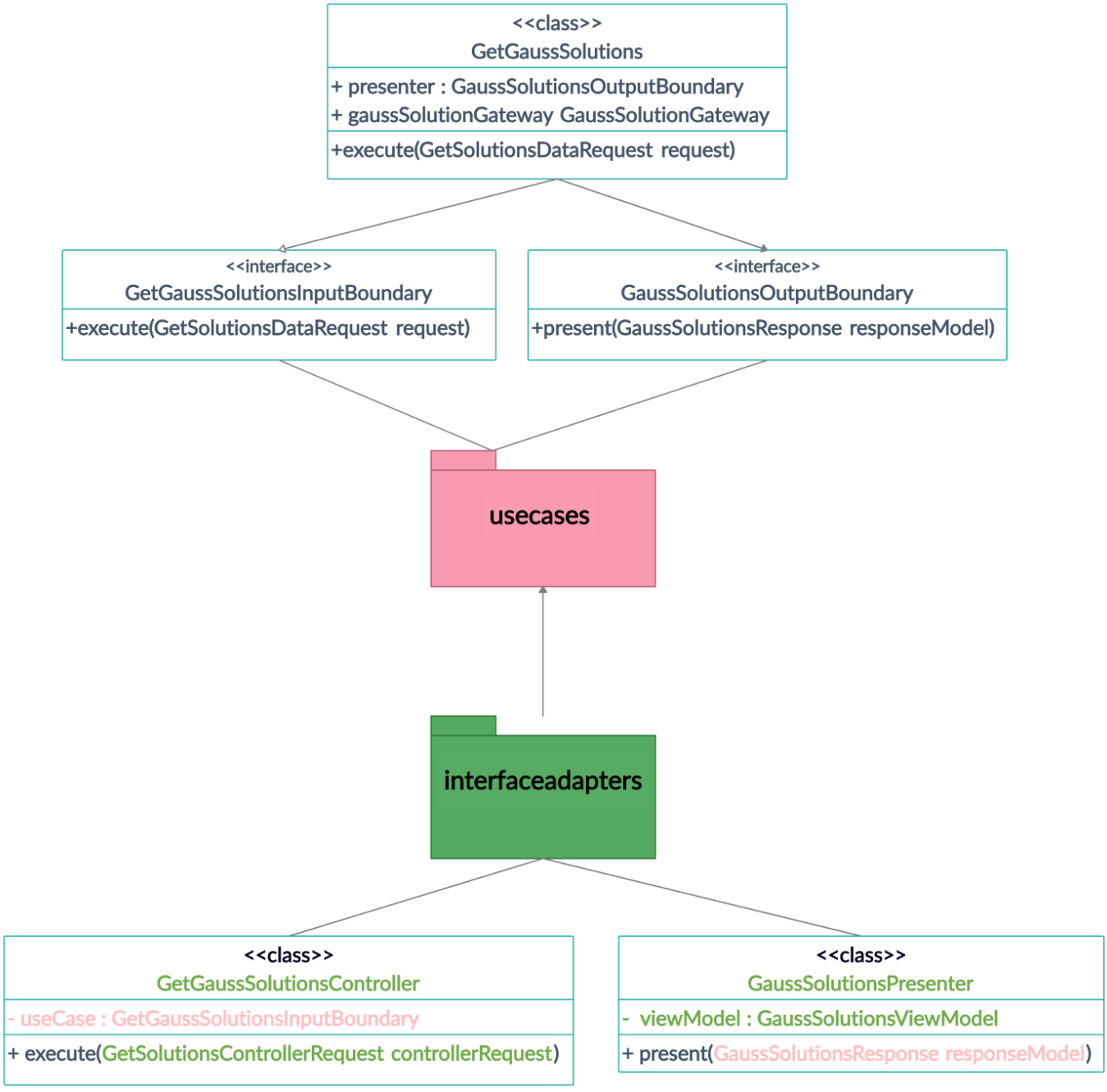
code snippets

```
@RestController()
@RequestMapping("/api/v1/gausssolver")
public class GetGaussSolutionsEndpoint implements BaseEndpoint {

    private final GetGaussSolutionsControllerInputBoundary controller;
    private final GaussSolutionsPresenterOutputBoundary presenter;

    public GetGaussSolutionsEndpoint(
            GetGaussSolutionsControllerInputBoundary controller,
            GaussSolutionsPresenterOutputBoundary presenter) {
        this.controller = controller;
        this.presenter = presenter;
    }

    @GetMapping
    @ApiOperation(value = "Get solutions", response = GaussSolutionsViewModel.class)
    public ResponseEntity execute() {
        controller.execute(new GetSolutionsControllerRequest());

        return ResponseEntity.ok(presenter.getViewModel());
    }
}
```

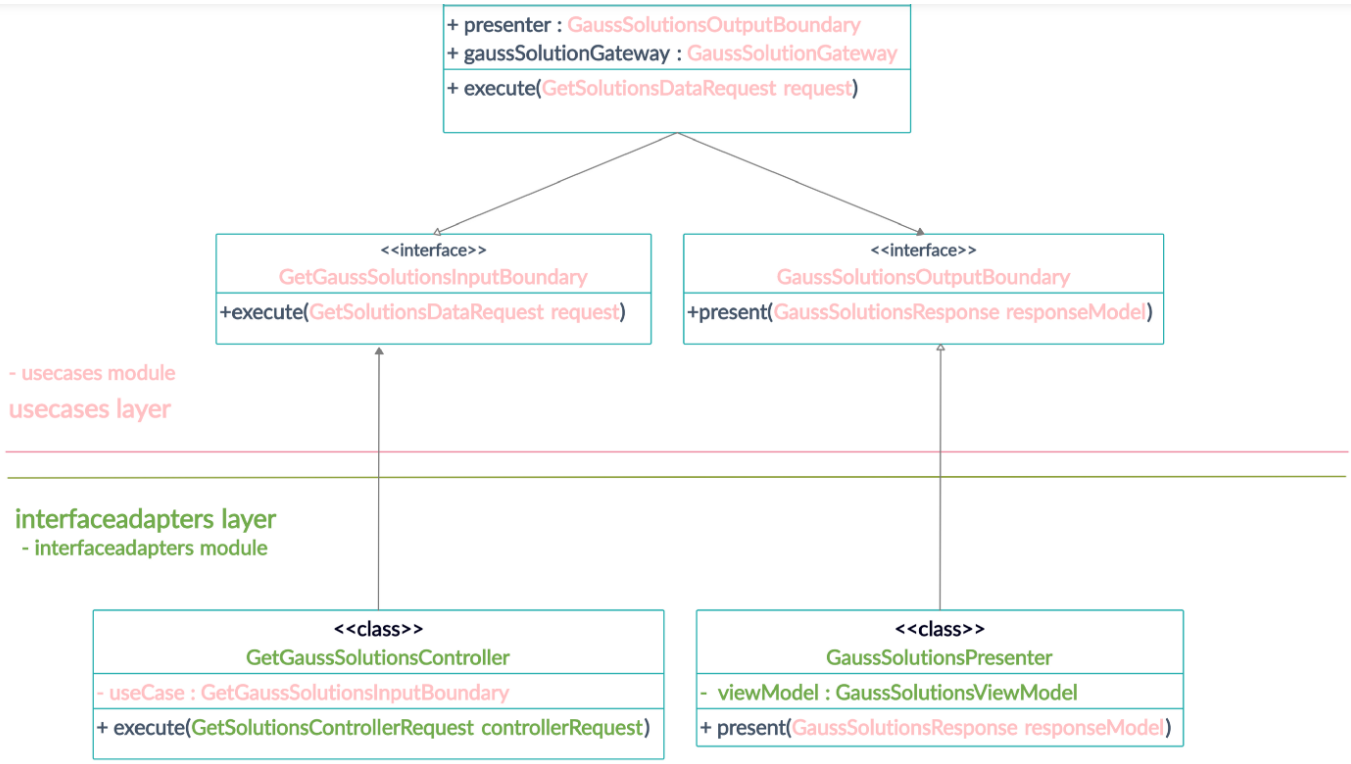
interfaceadapters module

interfaceadapters module **depends** on the the layer below it concretely on **usecases** module.

The schema below shows the **interaction** between **interfaceadapters** module and **usecases**.



Class diagram below shows the **dependencies** of classes in **interfaceadapters** module on the layer above in **usecase** module.



Both the **presenter** and **controller** communicate with the **usecase using** the **interfaces**, which helps to **pass** the **boundary** and don't break the **dependency rules**.

GetGaussSolutionsController.class in the **interfaceadapters** module uses the **interface GetGaussSolutionsInputBoundary** which is in the **usecase** module **to send the request** to the **GetGaussSolutions.class**.

GetGaussSolutions.class is the **usecase** of getting the gauss solutions. **GetGaussSolutions.class** implements **GetGaussSolutionsInputBoundary interface** and **get the request** from **GetGaussSolutionsController.class**.

On the same time **GetGaussSolutions.class** uses **GaussSolutionsOutputBoundary** interface.

- **GaussSolutionsOutputBoundary** interface is **implemented** by **GaussSolutionsPresenter.class** in **interfaceadapters** module.
- **GetGaussSolutions.class** presents the **response** to **GaussSolutionsPresenter.class** with the help of **GaussSolutionsOutputBoundary** interface.

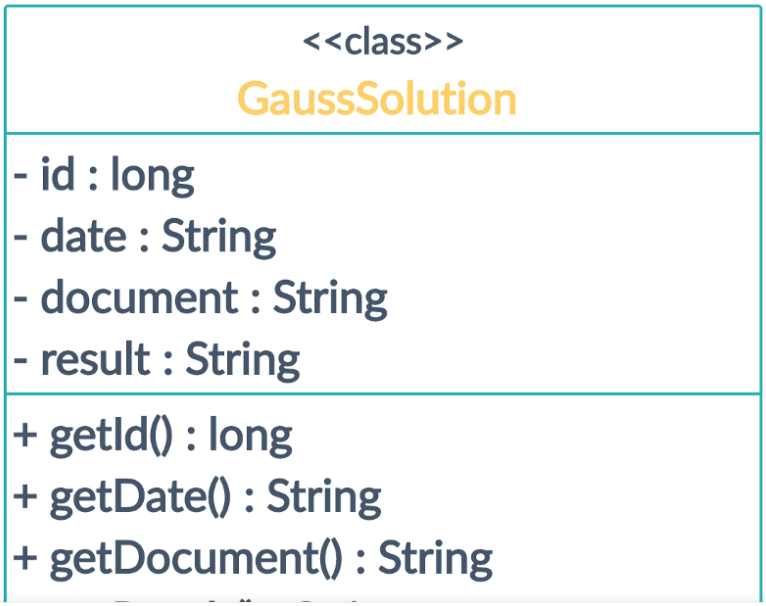
```
public class GetGaussSolutionsController implements GetGaussSolutionsControllerInputBoundary {
    private final GetGaussSolutionsInputBoundary useCase;

    public GetGaussSolutionsController(GetGaussSolutionsInputBoundary useCase) {
        this.useCase = useCase;
    }

    @Override
    public void execute(GetSolutionsControllerRequest getSolutionsControllerRequest) {
        useCase.execute(new GetSolutionsDataRequest());
    }
}

public class GaussSolutionsPresenter extends BaseGaussSolutionPresenter
        implements GaussSolutionsOutputBoundary, GaussSolutionsPresenterOutputBoundary {

    private GaussSolutionsViewModel viewModel;

    @Override
    public GaussSolutionsViewModel getViewModel() {
        return viewModel;
    }
```

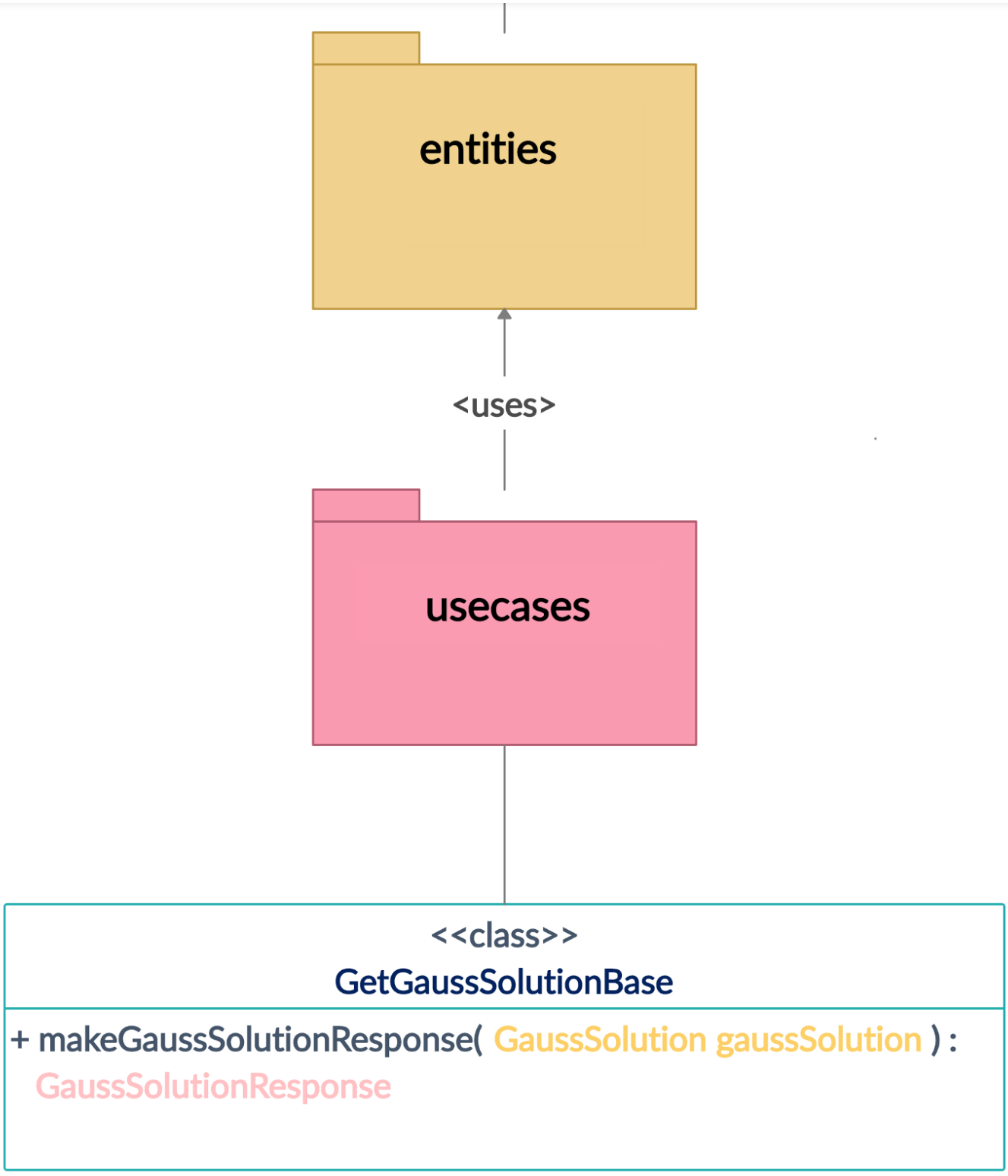
```
                .stream()
                .map(BaseGaussSolutionPresenter::mapToGaussSolutionsViewModel)
                .forEach(gaussSolutionsViewModelBuilder::showSolutionsDataViewModel);
        viewModel = gaussSolutionsViewModelBuilder.build();
    }
}

public interface GetGaussSolutionsInputBoundary {
    void execute(GetSolutionsDataRequest request);
}

public interface GaussSolutionsOutputBoundary {

    void present(GaussSolutionsResponse responseModel);
}

public class GaussSolutionsPresenter extends BaseGaussSolutionPresenter
        implements GaussSolutionsOutputBoundary, GaussSolutionsPresenterOutputBoundary {

    private GaussSolutionsViewModel viewModel;

    @Override
    public GaussSolutionsViewModel getViewModel() {
        return viewModel;
    }

    @Override
    public void present(GaussSolutionsResponse responseModel) {
        GaussSolutionsViewModel.GaussSolutionsViewModelBuilder gaussSolutionsViewModelBuilder =
GaussSolutionsViewModel.builder();
        responseModel.getSolutions()
                .stream()
                .map(BaseGaussSolutionPresenter::mapToGaussSolutionsViewModel)
                .forEach(gaussSolutionsViewModelBuilder::showSolutionsDataViewModel);
        viewModel = gaussSolutionsViewModelBuilder.build();
    }
}
```

In this way the **direction of the dependency** stays from **interfaceadapters** layer to **usecase** layer without breaking any **depency rules** and without creating any dependency from the higher level module to the lower level module.

usecases module

usecases module schema which shows the only **dependency** to **entities** module.





Class diagram of the usecase.

```
public class GetGaussSolutions extends GetGaussSolutionBase implements GetGaussSolutionsInputBoundary {
    private final GaussSolutionsOutputBoundary presenter;
    private final GaussSolutionGateway gaussSolutionGateway;

    public GetGaussSolutions(GaussSolutionsOutputBoundary presenter, GaussSolutionGateway
gaussSolutionGateway) {
        this.presenter = presenter;
        this.gaussSolutionGateway = gaussSolutionGateway;
    }
```

```
            presenter.present(result.build());
        }
    }
```

Api calls

Api call to **solve** linear equations with gausssolver

```
POST
http://localhost:8080/gausssolver-service/api/v1/gausssolver
body
{
    "a": [[2.0,4.0],[5.0, -6.0]],
    "b": [8.0,4.0]
}

Response of the call:

{
    "solution": [
        2.0,
        1.0
    ]
}
```

Api call to **get the solutions**

```
GET
http://localhost:8080/gausssolver-service/api/v1/gausssolver

Response of the call:

{
    "solutionsData": [
        {
            "id": 1,
            "date": "2020-11-11",
            "document": "{\"a\":[[2.0,4.0],[5.0,-6.0]],\"b\"[8.0,4.0]}",
            "result": "[2.0,1.0]"
        }
    ]
}
```

Swagger documentation:

http://localhost:8080/gausssolver-service/swagger-ui.html#/gausssolver

Thank you for reading :)

My repository link:

rildikondi/CleanArchitectureSpring

Clean Architecture with multimodules written in Java and the web layer is build with the help of Spring. Clean…

github.com

References: