I going to tell you about my test project now.

So, it is a WEB application which provides a GUI to user to interact with database. Following features should be implemented:

- Reading data from database and displaying it in browser as a table
- Editing and adding and deleting any record
- Filtering director records by birth date
- Back and front must be two separated applications.

The application has classical client-server architecture. Client and server use RESTful API for interaction. JSON is used to transfer data because it is human readable and lightweight.

Back-end architecture is not really complicated.
The repository layer is responsible for all database queries and mapping results into POJO.
Services interacts with the repository and the model. They are not too smart. Their purpose is to provide a simple interface for the view hiding application structure from it.
The view is responsible for client's HTTP (Hypertext Transfer Protocol) request processing. It just parses the requests, calls corresponding service, and generates a response.
The model is the core of the application. It is completely independent of any other modules. It represents entities from database and contains all possible DTOs.

As for the technologies, Spring stack is used to build the back end. First, it is Spring Beans and Spring Context. I do not think it is possible to create a clear huge project with no IoC container. Spring JDBCis used to interact with database. It is much cleaner and easier than plain JDBC and not too abstract like Spring Data.
Spring MVC helps you to create controllers to process HTTP requests simple.
Spring Test builds test environment to perform integration testing.
Finally, Spring Boot lets you not to worry about configuration providing good default parameters.
To make unit tests Junit5 and Mockito were chosen.
Swagger helps to create documentation based on existing controllers automatically.

Front-end part is completely different.
First, it uses Flux architecture. That means, I have got here one way data flow. So, when view is updated, it calls a dispatcher which changes a state, and this change causes a view's rerender.
The repository layer is a place when front-end communicates to the server.
The state layer is a state management system. It does not have to be Redux, but it both React and Redux are about functional programming and data immutability, so it is easy to use them together.
View is the highest level of whole project. It is final GUI for user.

I did not use anything new or experimental to build front end part:

- React as UI building library.
- Redux as state management system.
- React-Redux to provide a context for react components to avoid props drilling.
- Redux-Thunk as middleware for redux, which provides a possibility to dispatch no object only but functions as well. It is widely used in server requests.
- React Forms to make easier redux form processing.
- Axious to make requests to the server easier.