First day(monday):

* Setup data base connection and configuration.
* Create data base and it is model (User, Task, Project, Client).

Second day(Tuesday):

* Complete data base model (report).
* Start working on user controller but hit the wall that request body is not reaching server side.

Third day (Wednesday):

* Solve the previews issue about request body.
* Implement user data field validation.
* Watching videos about security configuration and user security.

Fourth day (Thursday)

* Apply basic authentication username and password.
* Save password encrypted using bcrypt lib from spring security with 12 round.
* Read article about spring security authentication process.
* Watch 4 videos about spring boot authentication.

Fifth day(Monday):

* Setup security file.
* Implement jwt validation.

6Thday (Tuesday):

* CORS do not prevent form action from another clients that are not registered in CORS.
* Session vs JWT

1. Session stored on server so server need to look it up when needed while jwt has user credential encrypted inside jwt, just decrypt jwt and check it if same as user clams.
2. Jwt can be used with many servers while session cant be share. Every server has it own session id.

* JWT content:

1. Header: algorrithm name & type (two var).
2. Payload: data (name, exp, iat).

* Signature: output of hashing of (Header + payload) by secret key.

secret key should be saved on not accessible file.

* Jwt is validated by signature.

7th day (Wednesday):

the **spring-boot-starter-validation** dependency is correctly in place, the **GlobalExceptionHandler** should be handling the validation errors as originally intended. I have reverted the **adminController.java** to remove the explicit **BindingResult** checks, making the controller cleaner and relying on the global handler.

Here are the key files in their corrected and final state for this validation setup:

1. **adminController.java**: Now relies on **@Valid** to trigger validation and the **GlobalExceptionHandler** to handle errors.
2. **Client.java**: Uses **@NotBlank** (or **@NotEmpty** as per your preference) for field validation.
3. **GlobalExceptionHandler.java**: Catches **MethodArgumentNotValidException** and formats the error response as you requested.

With these in place and the correct dependency in your **pom.xml**, your validation should be robust.