2. Code Review

Description:

Review the provided UserController code and suggest improvements for code quality, security, and maintainability.

Deliverables:

Annotated review comments.

High-level summary of recommended improvements and rationale.

Code to Review:

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable("id") Long id) {

Optional<User> user = userService.findUserById(id);

if (user.isPresent()) {

return ResponseEntity.ok(user.get());

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

@PostMapping("/create")

public ResponseEntity<String> createUser(@RequestBody Map<String, String> payload) {

String username = payload.get("username");

String password = payload.get("password");

User newUser = new User();

newUser.setUsername(username);

newUser.setPassword(password); // Sensitive data issue: plain-text storage

userService.saveUser(newUser);

return ResponseEntity.status(HttpStatus.CREATED).body("User created successfully");

}

@PutMapping("/{id}/update")

public ResponseEntity<String> updateUser(@PathVariable("id") Long id, @RequestBody Map<String, String> payload) {

Optional<User> existingUser = userService.findUserById(id);

if (existingUser.isPresent()) {

User user = existingUser.get();

user.setUsername(payload.getOrDefault("username", user.getUsername()));

user.setPassword(payload.getOrDefault("password", user.getPassword())); // Plain-text storage and improper validation

userService.saveUser(user);

return ResponseEntity.ok("User updated successfully");

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

@DeleteMapping("/{id}/delete")

public ResponseEntity<String> deleteUser(@PathVariable("id") Long id) {

boolean deleted = userService.deleteUserById(id);

if (deleted) {

return ResponseEntity.ok("User deleted successfully");

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

}

1. Here the first this is moving all the logic to service layer.
2. Try to add Global Exception handler
3. Do not save the password as plain text
4. Add swagger and it documentation
5. If it is banal method like CRUD use base method not @PostMapping(“/create”) but just @PostMapping while the class has “.”
6. HttpCodes must be related. E.g: if username is already present return 409 conlict.

3. Database Performance and Optimization

1. First find cause.
2. Put proper indexing for most filtered columns, not all, if you put all writing to db gets slower
3. Return large data with pagination
4. Find places that might cause N+1 problem
5. Try to use ORM
6. Avoid unnecessary columns to select
7. Replace sub-queries with JOIN or ETCH Join
8. If there too many data in db implement Partitioning
9. Code refactoring

@RestController

@RequestMapping("/users")

public class UserController {

private final UserService userService;

public UserController(UserService userService) {

this.userService = userService;

}

@PostMapping

public ResponseEntity<String> createUser(@Valid @RequestBody User user) {

if (userService.findUserByEmail(user.getEmail()) != null) {

return ResponseEntity.status(HttpStatus.CONFLICT)

.body("User with this email already exists");

}

userService.saveUser(user);

return ResponseEntity.status(HttpStatus.CREATED)

.body("User created successfully");

}

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

return userService.findUserById(id)

.map(user -> ResponseEntity.ok(user))

.orElse(ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

@PutMapping("/{id}")

public ResponseEntity<String> updateUser(@PathVariable Long id, @Valid @RequestBody User user) {

return userService.findUserById(id)

.map(existingUser -> {

updateExistingUser(existingUser, user);

userService.saveUser(existingUser);

return ResponseEntity.ok("User updated successfully");

})

.orElse(ResponseEntity.status(HttpStatus.NOT\_FOUND)

.body("User not found"));

}

private void updateExistingUser(User existingUser, User updatedUser) {

if (updatedUser.getName() != null && !updatedUser.getName().isEmpty()) {

existingUser.setName(updatedUser.getName());

}

if (updatedUser.getEmail() != null && updatedUser.getEmail().contains("@")) {

existingUser.setEmail(updatedUser.getEmail());

}

}

}

To make better implement construcor injection

As before said move business logic to the database