

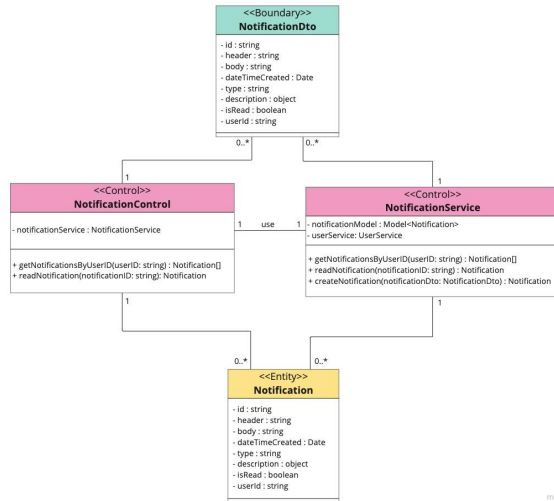


Mapping Design to Code

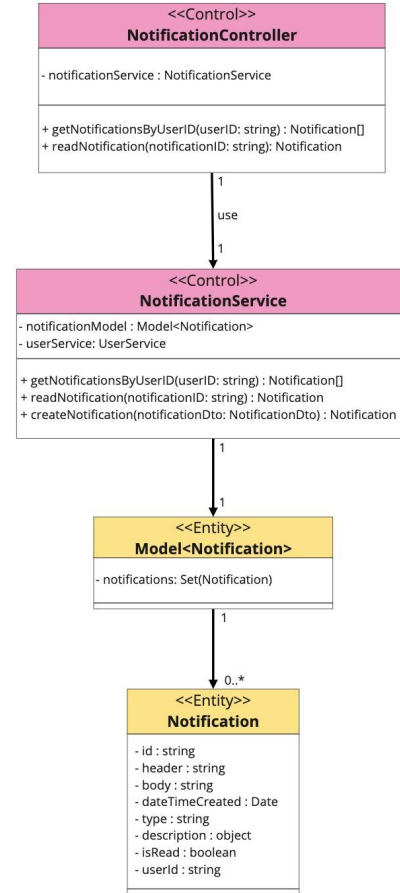
Notification System



Class Diagram

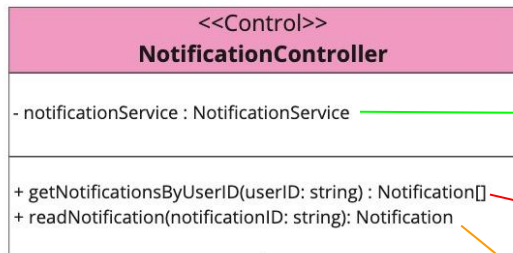


miro



miro

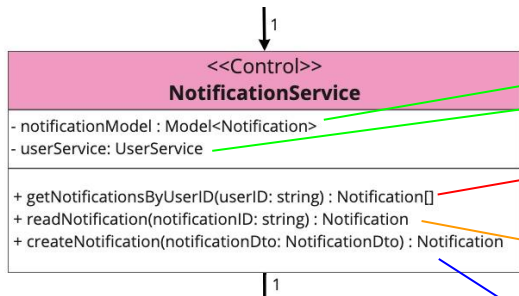
NotificationControl



1

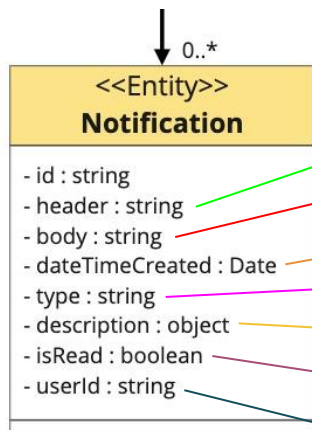
```
5  @Controller('notification')
6  export class NotificationController {
7  → constructor(private notificationService: NotificationService) {}
8
9
10 @Get("user/:userId")
11 @ApiOperation({summary: "Get Notification By User Id", tags: ["Notification"]})
12 @ApiResponse({status: 400, description: "No User Id Exist"})
13 → getNotificationsByUserID(@Param("userId") userId: string){
14     return this.notificationService.getNotificationsByUserId(userId);
15 }
16
17 @Patch("/:notificationId")
18 @ApiOperation({summary: "Update Notification Status (Read Status)", tags: ["Notification"]})
19 @ApiResponse({description: "Return updated Notification"})
20 → readNotification(@Param("notificationId") notificationId: string){
21     return this.notificationService.readNotification(notificationId);
22 }
23
24 }
```

NotificationService



```
7  @Injectable()
8  export class NotificationService {
9      constructor(
10         @InjectModel('notifications') private readonly notificationModel: Model<any>,
11         private userService: UserService
12     ) {}
13
14     async getNotificationsByUserId(userId: string) {
15         if (await this.userService.findById(userId)) {
16             const notifications = await this.notificationModel.find({userId: userId}).sort({dateTimeCreated: -1})
17             return notifications
18         }
19     }
20
21     async readNotification(notificationID: string) {
22         const notification = await this.notificationModel.findById(notificationID)
23         notification.isRead = true
24         return notification.save()
25     }
26
27     async createNotification(notificationDto: object){
28         const notification = new this.notificationModel(notificationDto)
29         return notification.save()
30     }
31 }
```

Notification



```
8  @Schema()
9  export class Notification {
10
11      @Prop({required: true})
12      header: string;
13
14      @Prop({required: true})
15      body: string;
16
17      @Prop({required: true, default: Date.now})
18      dateTimeCreated: Date;
19
20      @Prop({required: true})
21      type: string;
22
23      @Prop({default: {}})
24      description: NotificationDescription;
25
26      @Prop({required: true, default: false})
27      isRead: boolean;
28
29      @Prop({required: true})
30      userId: ObjectId;
31  }
```



White Box Testing

Test AuthService Class

Method

- Student / Tutor Log in
- Admin Log in

```
1 import { Injectable, BadRequestException, NotFoundException } from "@nestjs/common";
2 import { UserService } from '../user.service';
3
4 @Injectable()
5 export class AuthService {
6   constructor(private userService: UserService) {}
7
8   > async signin(email: string, password: string) { ...
26   }
27
28   > async signinAdmin(email: string, password: string) { ...
47   }
48 }
```

Virtual Database

Student

```
id: "622f7b31262b9d8b120dc61a",
type: "student",
firstname: "TestCredit",
lastname: "Credit",
username: "moneymoneymoney",
email: "money1@gmail.com",
password: "aaaaa",
phoneNumber: "0655555555",
displayName: "GG",
birthdate: "2000-10-03",
gender: "male",
educationalLevel: 7,
picture: "",
isAdmin: false,
coursesLearned: [],
citizenId: "6113319262678",
citizenImage: "",
dateTimeCreated: new Date(),
dateTimeUpdated: new Date()
```

Admin

```
id: "622f7b9e262b9d8b120dc622",
type: "admin",
firstname: "TestAdmin",
lastname: "Admin",
username: "admin",
email: "admin@gmail.com",
password: "aaaaa",
phoneNumber: "0655555555",
displayName: "GG",
birthdate: "2000-10-03",
gender: "male",
educationalLevel: 7,
picture: "",
isAdmin: true,
coursesLearned: [],
citizenId: "2680284325881",
citizenImage: "",
dateTimeCreated: new Date(),
dateTimeUpdated: new Date()
```


Test case for Student / Tutor Login

Test Case

- User Login Success
- User Login Failed - No User
- User Login Failed - Wrong Password
- User Login Failed - Admin Login at User Login

```
it('user login success', async () => {
  try {
    const user = await authService.signin('money1@gmail.com', 'aaaaa')
    expect(user).toBeDefined();
  } catch(err) {}
})

it('user login failed - no user', async () => {
  await expect(authService.signin('money1@gmail', 'aaaaa')).rejects.toEqual(
    new BadRequestException("Email or Password is incorrect.")
  )
})

it('user login failed - wrong password', async () => {
  await expect(authService.signin('money1@gmail.com', 'bbbbbb')).rejects.toEqual(
    new BadRequestException("Email or Password is incorrect.")
  )
})

it('user login failed - admin cannot login at user login', async () => {
  await expect(authService.signin('admin@gmail.com', 'aaaaa')).rejects.toEqual(
    new BadRequestException("Admin cannot login here")
  )
})
```

Test case for Admin Login

Test Case

- Admin Login Success
- Admin Login Failed - No Admin
- Admin Login Failed - Wrong Password
- Admin Login Failed - User Login at Admin Login

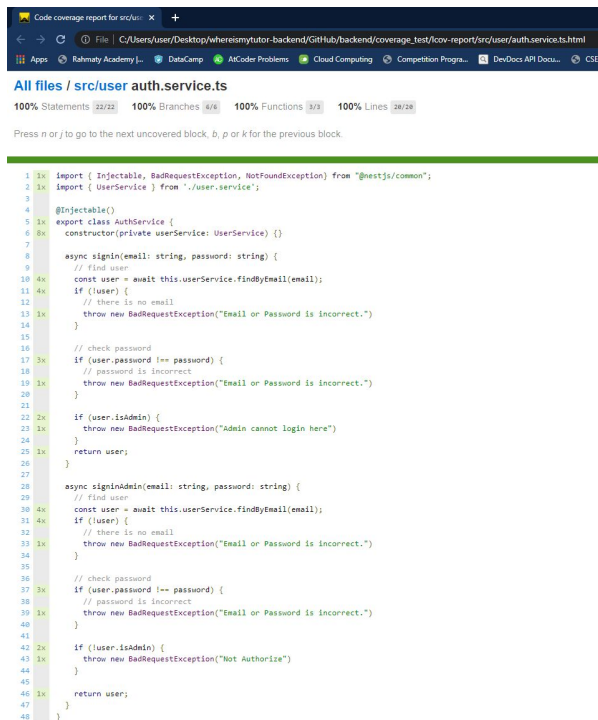
```
it('admin login success', async () => {
  try {
    const user = await authService.signinAdmin('admin@gmail.com', 'aaaaa')
    expect(user).toBeDefined();
  } catch(err) {}
})

it('admin login failed - no admin', async () => {
  await expect(authService.signinAdmin('money1@gmail', 'aaaaa')).rejects.toEqual(
    new BadRequestException("Email or Password is incorrect.")
  )
})

it('admin login failed - wrong password', async () => {
  await expect(authService.signinAdmin('money1@gmail.com', 'bbbbbb')).rejects.toEqual(
    new BadRequestException("Email or Password is incorrect.")
  )
})

it('admin login failed - user cannot login at admin login', async () => {
  await expect(authService.signinAdmin('money1@gmail.com', 'aaaaa')).rejects.toEqual(
    new BadRequestException("Not Authorize")
  )
})
```

Result : Statement Coverage



The screenshot shows a web browser displaying a code coverage report for the file `src/user/auth.service.ts`. The report indicates that 100% of the statements, branches, functions, and lines are covered. The code is displayed with line numbers and coverage status (1x for covered, 0x for not covered).

```
1 1x import { Injectable, BadRequestException, NotFoundException } from "@nestjs/common";
2 1x import { UserService } from "../user.service";
3
4 @Injectable()
5 1x export class AuthService {
6 0x   constructor(private userService: UserService) {}
7
8   async signIn(email: string, password: string) {
9     // find user
10 4x   const user = await this.userService.findByEmail(email);
11 4x   if (!user) {
12     // there is no email
13 1x   throw new BadRequestException("Email or Password is incorrect.");
14   }
15
16   // check password
17 3x   if (user.password !== password) {
18     // password is incorrect
19 1x   throw new BadRequestException("Email or Password is incorrect.");
20   }
21
22 2x   if (user.isAdmin) {
23 1x   throw new BadRequestException("Admin cannot login here");
24   }
25 1x   return user;
26 }
27
28 async signInAdmin(email: string, password: string) {
29   // find user
30 4x   const user = await this.userService.findByEmail(email);
31 4x   if (!user) {
32     // there is no email
33 1x   throw new BadRequestException("Email or Password is incorrect.");
34   }
35
36   // check password
37 3x   if (user.password !== password) {
38     // password is incorrect
39 1x   throw new BadRequestException("Email or Password is incorrect.");
40   }
41
42 2x   if (user.isAdmin) {
43 1x   throw new BadRequestException("Not Authorize");
44   }
45
46 1x   return user;
47 }
48 }
```