Bilkent University

Department of Computer Engineering

CS 319 Term Project

Spring 2025

Section 1

Group 1

Deliverable 4

Submission

Group Members:

Emre Algür - 22202673

Irmak İmdat - 22201570

Alper Yıldırım - 22102033

Ömer Yaslıtaş - 21902874

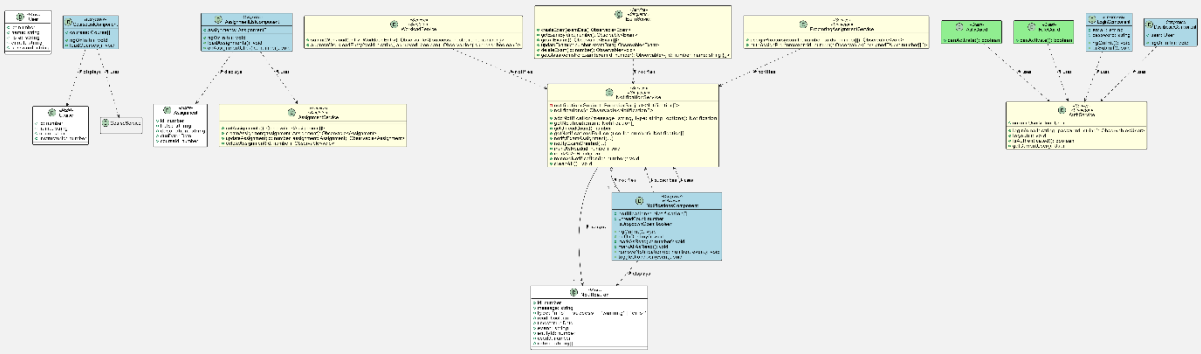
Melih Rıza Yıldız - 21902958

Aykhan Ahmadzada - 21903609

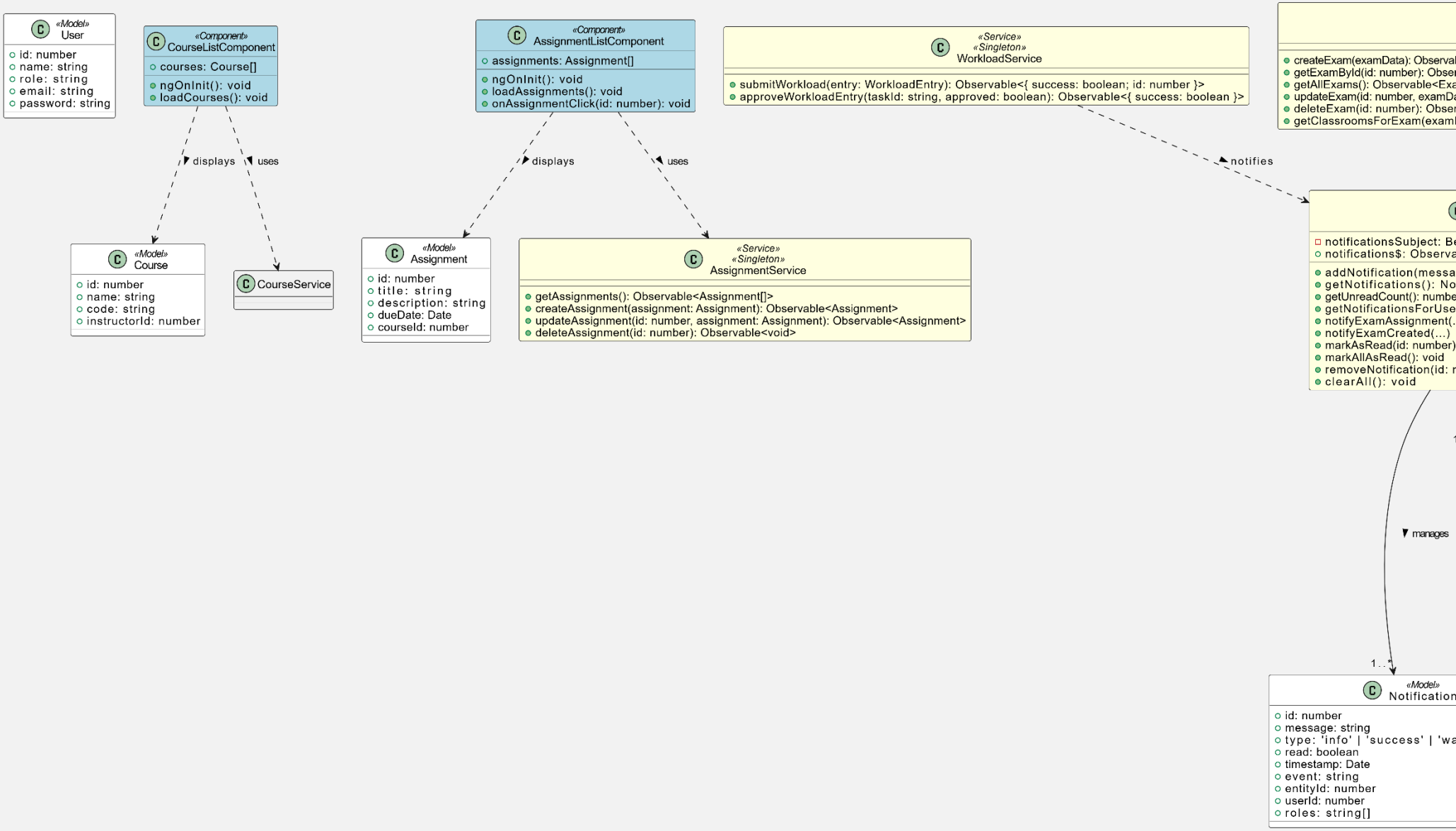
Instructor: Eray Tüzün

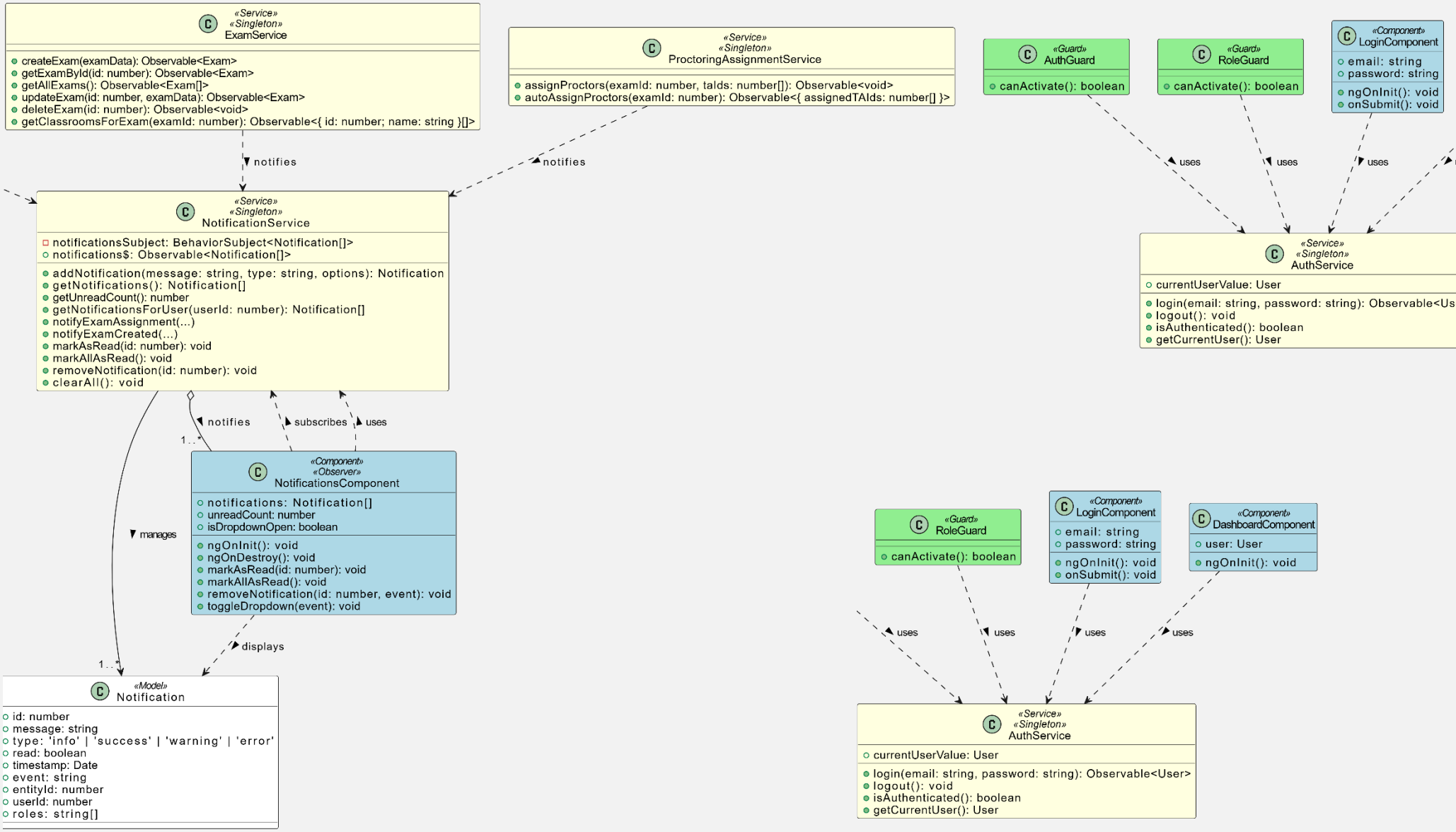
Teaching Assistant: Yahya Elnouby

1. Class Diagram



Zoomed:


«Model»
User
id: number
name: string
role: string
email: string
password: string
«Component»
CourseListComponent
courses: Course[]
ngOnInit(): void
loadCourses(): void
displays
uses
«Model»
Course
id: number
name: string
code: string
instructorId: number
CourseService
«Component»
AssignmentListComponent
assignments: Assignment[]
ngOnInit(): void
loadAssignments(): void
onAssignmentClick(id: number): void
displays
uses
«Model»
Assignment
id: number
title: string
description: string
dueDate: Date
courseId: number
«Service»
«Singleton»
AssignmentService
getAssignments(): Observable<Assignment[]>
createAssignment(assignment: Assignment): Observable<Assignment>
updateAssignment(id: number, assignment: Assignment): Observable<Assignment>
deleteAssignment(id: number): Observable<void>
«Service»
«Singleton»
WorkloadService
submitWorkload(entry: WorkloadEntry): Observable<{ success: boolean; id: number }>
approveWorkloadEntry(taskId: string, approved: boolean): Observable<{ success: boolean }>
notifies
createExam(examData): Observal
getExamById(id: number): Obser
getAllExams(): Observable<Exa
updateExam(id: number, examDa
deleteExam(id: number): Obser
getClassroomsForExam(exam
notificationsSubject: Be
notifications$: Observa
addNotification(messa
getNotifications(): No
getUnreadCount(): numbe
getNotificationsForUse
notifyExamAssignment(.
notifyExamCreated(...)
markAsRead(id: number)
markAllAsRead(): void
removeNotification(id: n
clearAll(): void
manages
1
1..*
«Model»
Notification
id: number
message: string
type: 'info' | 'success' | 'wa
read: boolean
timestamp: Date
event: string
entityId: number
userId: number
roles: string[]


«Service»
«Singleton»
ExamService
createExam(examData): Observable<Exam>
getExamById(id: number): Observable<Exam>
getAllExams(): Observable<Exam[]>
updateExam(id: number, examData): Observable<Exam>
deleteExam(id: number): Observable<void>
getClassroomsForExam(examId: number): Observable<{ id: number; name: string }[]>
«Service»
«Singleton»
ProctoringAssignmentService
assignProctors(examId: number, talds: number[]): Observable<void>
autoAssignProctors(examId: number): Observable<{ assignedTAIds: number[] }>
«Guard»
AuthGuard
canActivate(): boolean
«Guard»
RoleGuard
canActivate(): boolean
«Component»
LoginComponent
email: string
password: string
ngOnInit(): void
onSubmit(): void
notifies
notifies
uses
uses
uses
«Service»
«Singleton»
NotificationService
notificationsSubject: BehaviorSubject<Notification[]>
notifications$: Observable<Notification[]>
addNotification(message: string, type: string, options): Notification
getNotifications(): Notification[]
getUnreadCount(): number
getNotificationsForUser(userId: number): Notification[]
notifyExamAssignment(...)
notifyExamCreated(...)
markAsRead(id: number): void
markAllAsRead(): void
removeNotification(id: number): void
clearAll(): void
«Service»
«Singleton»
AuthService
currentUserValue: User
login(email: string, password: string): Observable<Us
logout(): void
isAuthenticated(): boolean
getCurrentUser(): User
notifies
subscribes
uses
1..*
«Component»
«Observer»
NotificationsComponent
notifications: Notification[]
unreadCount: number
isDropdownOpen: boolean
ngOnInit(): void
ngOnDestroy(): void
markAsRead(id: number): void
markAllAsRead(): void
removeNotification(id: number, event): void
toggleDropdown(event): void
manages
displays
1..*
«Model»
Notification
id: number
message: string
type: 'info' | 'success' | 'warning' | 'error'
read: boolean
timestamp: Date
event: string
entityId: number
userId: number
roles: string[]
«Guard»
RoleGuard
canActivate(): boolean
«Component»
LoginComponent
email: string
password: string
ngOnInit(): void
onSubmit(): void
«Component»
DashboardComponent
user: User
ngOnInit(): void
uses
uses
uses
uses
«Service»
«Singleton»
AuthService
currentUserValue: User
login(email: string, password: string): Observable<User>
logout(): void
isAuthenticated(): boolean
getCurrentUser(): User

1.1. Singleton Pattern

Services provided with { providedIn: 'root' } are singletons by default. This means only one instance of the service exists throughout the application, which is a classic implementation of the Singleton pattern. Singleton is used to ensure a single source of truth for shared state or logic, such as authentication, notifications, or data access.

Example:

NotificationService (src/app/services/notification.service.ts)

```
@Injectable({
  providedIn: 'root'
})
export class NotificationService {
  private notificationsSubject = new BehaviorSubject<Notification[]>([]);
  notifications$ = this.notificationsSubject.asObservable();
  // ...
}
```

AuthService (src/app/services/auth.service.ts)

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private currentUserSubject: BehaviorSubject<User | null>;
  public currentUser: Observable<User | null>;
  // ...
}
```

These services are injected wherever needed, but only one instance exists. For example, AuthService manages the current user session and authentication state globally, and NotificationService manages notifications for the whole app.

1.2. Observer Pattern

Components subscribe to these observables to react to changes in state, such as new notifications or authentication status. This pattern decouples the source of data from the consumers, allowing for reactive programming.

Example:

NotificationService (src/app/services/notification.service.ts)

```
private notificationsSubject = new BehaviorSubject<Notification[]>([]);
notifications$ = this.notificationsSubject.asObservable();
```

Usage in a component (src/app/components/notifications/notifications.component.ts)

```
constructor(private notificationService: NotificationService) {
  this.notificationService.notifications$.subscribe(notifications => {
```

```
    this.notifications = notifications;
    this.unreadCount = this.notificationService.getUnreadCount();
  });
}
```

The NotificationService exposes an observable (notifications$). Components subscribe to this observable to automatically update their UI when notifications change, without needing to poll or manually check for updates.