



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

Ay Khan Ahmadzada - 21903609

Instructor: Eray Tüzün

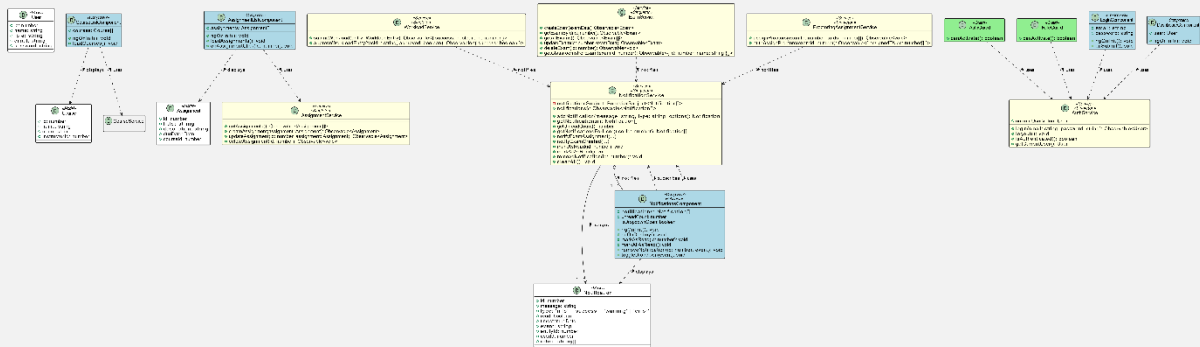
Teaching Assistant: Yahya Elnouby

1. Class Diagram

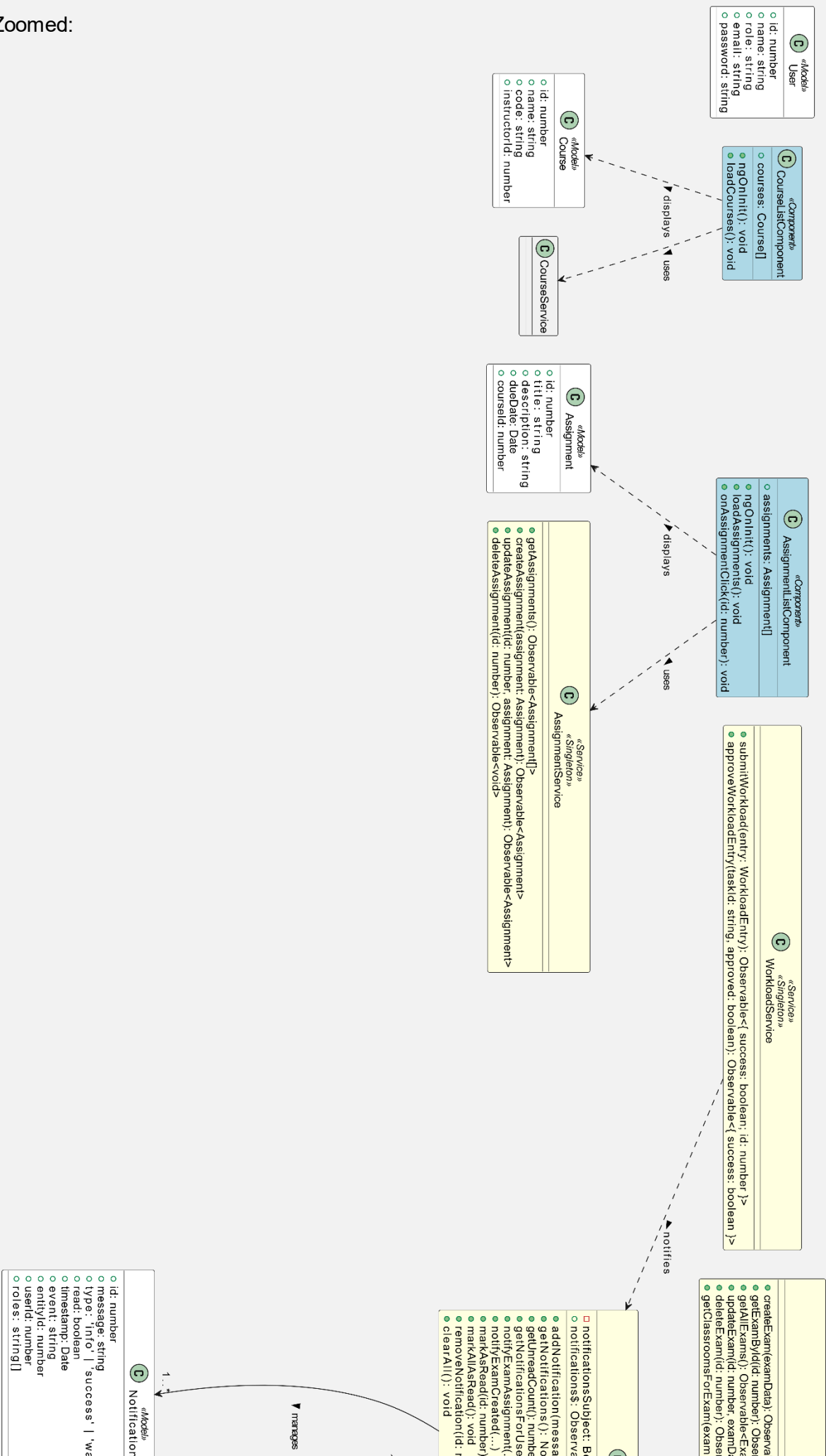
1.1. Singleton Pattern

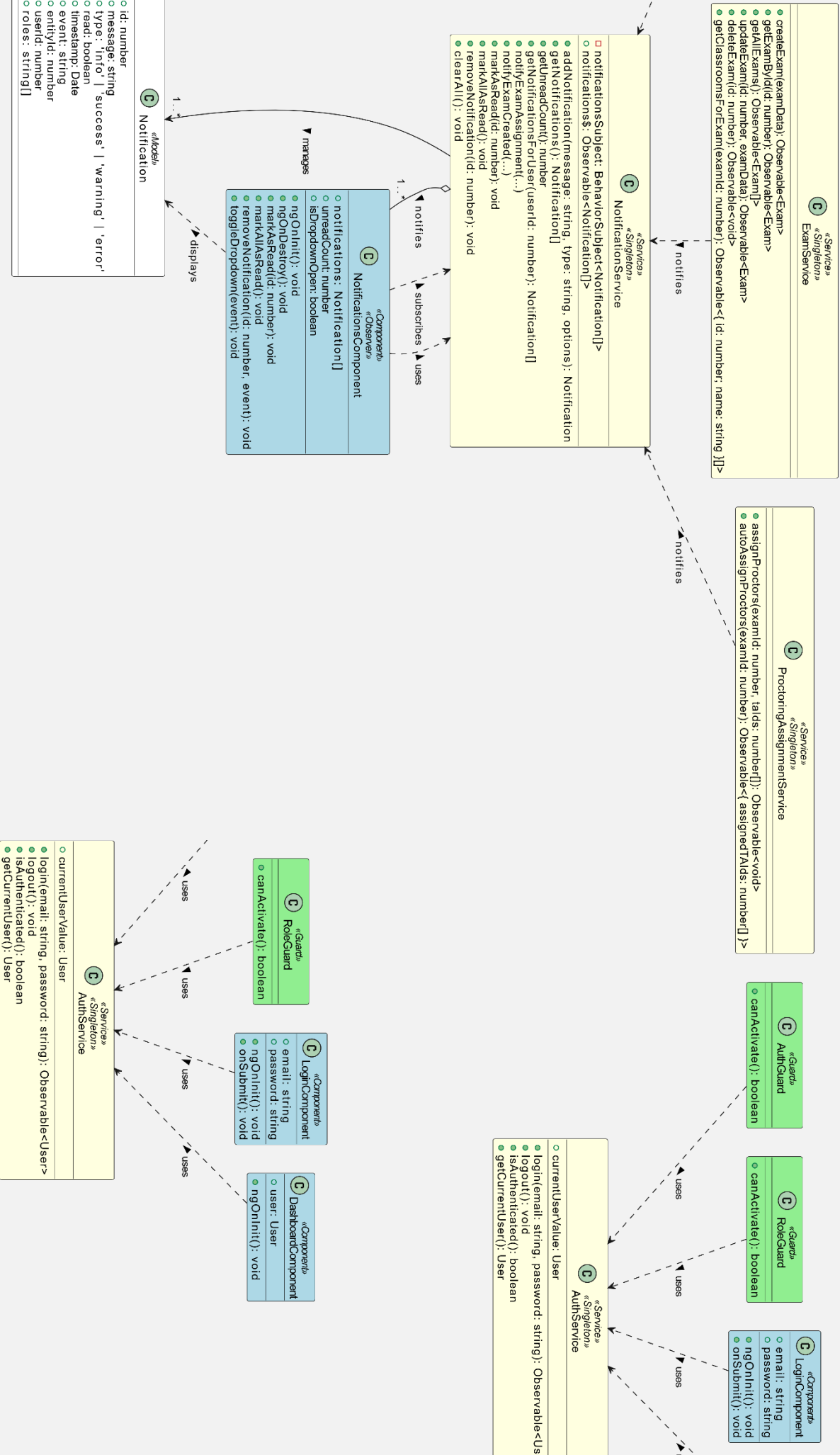
1.2. Observer Pattern

1. Class Diagram



Zoomed:





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.