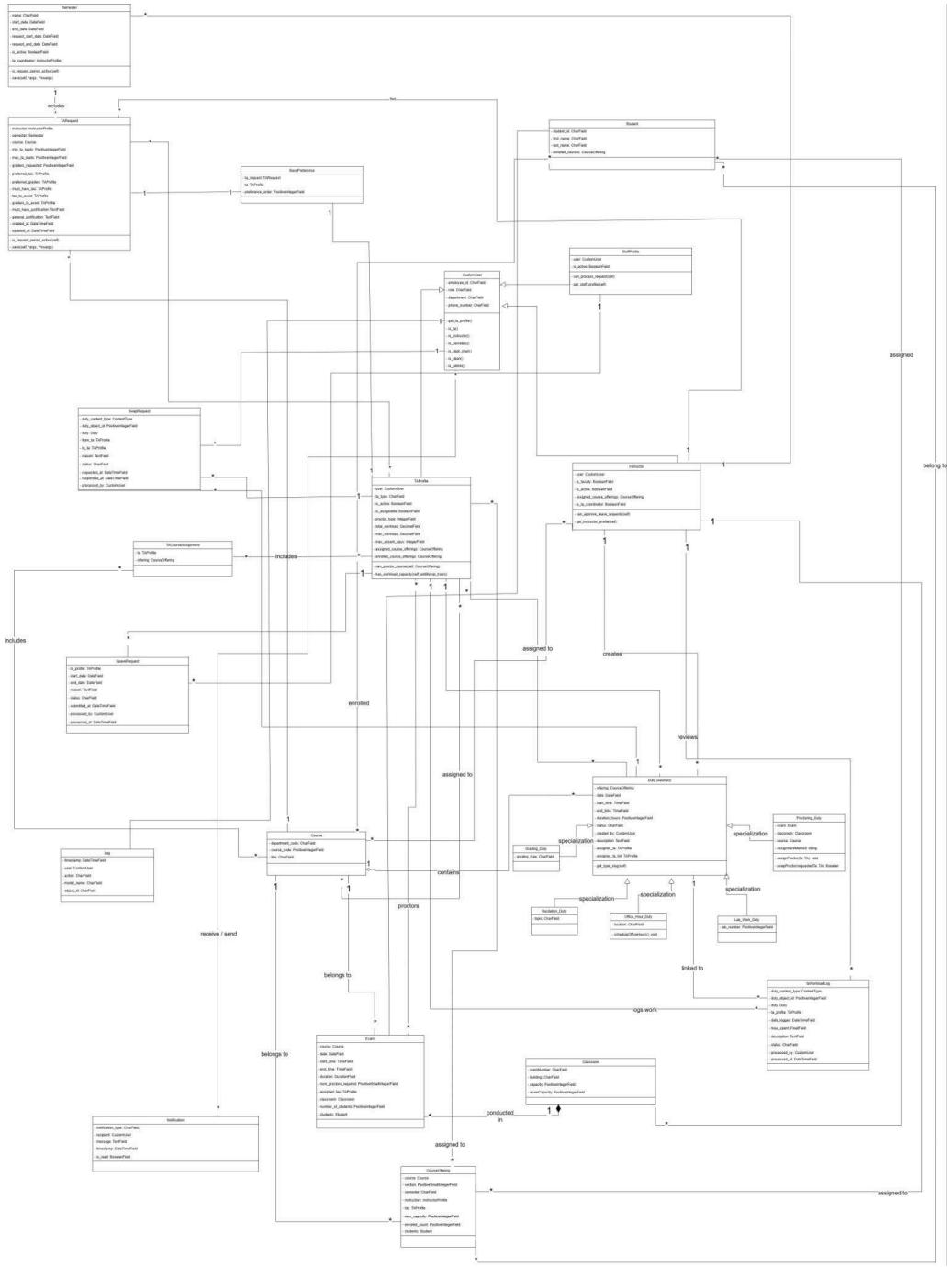
Department of Computer Engineering, Bilkent University
CS319 Object Oriented Software Engineering
Group 11
Detailed Class Diagram and Design Patterns - D4
Section 1 - Eray Tüzün
11/05/2025

Furkan Mert Aksakal 22003191
Furkan Komaç 22102165
Alper Biçer 22003097
Erkan Can Arslan 22103948
Deniz Yazıcı 21902557

Contents

1. Detailed Class Diagram

Link for the Detailed Class Diagram:
https://drive.google.com/file/d/1f2aDfOhgbxdaSYdC2rKcmkP0KFv6DgXL/view?usp=sharing
Alternative Link:
https://viewer.diagrams.net/?tags=%7B%7D&lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Detailed_Class_Diagram.drawio&dark=auto#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1f2aDfOhgbxdaSYdC2rKcmkP0KFv6DgXL%26export%3Ddownload



2. Design Patterns Used

a. Observer Pattern

Within our TA Management System, create_or_update_user_profiles is an Observer that is subscribed to the Django post_save signal for the CustomUser model. The moment the user record is saved, whether as a new save or update, the handler automatically creates or synchronizes the corresponding TAProfile, InstructorProfile, or StaffProfile based on the user's role. By handling these save events rather than adding profile-management logic to registration or update processes, the Observer Pattern decouples user persistence from profile maintenance, establishing a single centralized means of synchronizing profiles.

Real-Life Example from the Project:

```
@receiver(post_save, sender=CustomUser)
def create_or_update_user_profiles(sender, instance, created,
**kwargs):
    if instance.role == CustomUser.Roles.TA:
        TAProfile.objects.get_or_create(user=instance)
    elif instance.role == CustomUser.Roles.INSTRUCTOR:
        InstructorProfile.objects.get_or_create(user=instance)
    elif instance.role in [
        CustomUser.Roles.SECRETARY,
        CustomUser.Roles.DEAN,
        CustomUser.Roles.DEPT_CHAIR
    ]:
        StaffProfile.objects.get_or_create(user=instance)
```

b. Decorator Pattern

Our class-based views employ the Decorator Pattern via Django's @method_decorator wrapper around the dispatch method in order to inject authentication and permission without affecting the business logic of the view. For example, decorating UserUpdateView with @method_decorator([login_required, user_passes_test(is_admin_user)], name="dispatch") effortlessly enforces that only logged-in admins can invoke any CRUD operation. This approach elegantly inserts cross-cutting concerns—such as access control—between basic view behavior, promoting concerns separation and permitting simple addition or modification of policy without modifying the view implementation

Example from the Project:

```
@method_decorator([login_required,
user_passes_test(is_admin_user)], name="dispatch")
class UserUpdateView(UpdateView):
    model = CustomUser
    form_class = CustomUserEditForm
    template_name = "accounts/user_edit.html"
    success_url = reverse_lazy("accounts:user_list")

    def form_valid(self, form):
        # custom TA-profile updating logic...
        return super().form_valid(form)
```

c. Role Based Access (RBAC)

We implement the RBAC pattern in our TA Management System through the inclusion of a role field in the CustomUser model (e.g. TA, Instructor, Secretary, Dean) and implementing access rules via a shared mixin or decorators that check request.user.role before doing any sensitive action. For instance, DutyApproveView uses a RoleRequiredMixin with required_roles = [CustomUser.Roles.INSTRUCTOR, CustomUser.Roles.DEAN] such that only instructors or deans can approve duty logs; anyone else will receive a 403 Forbidden. Likewise, views to log a duty or submit a report are all wrapped up in decorators like @user_passes_test(is_ta_user) so that these are restricted to TA roles. By keeping role checks in one place, the RBAC pattern elegantly separates authorization from business logic so that it is simple to add new roles or change permissions without disrupting each view's fundamental code.

Example from the Project:

```
@login_required
def log_completed_duty(request):
    if not hasattr(request.user, 'ta_profile'):
        return redirect('home')
```