Notes:

Student:

+Controller level:

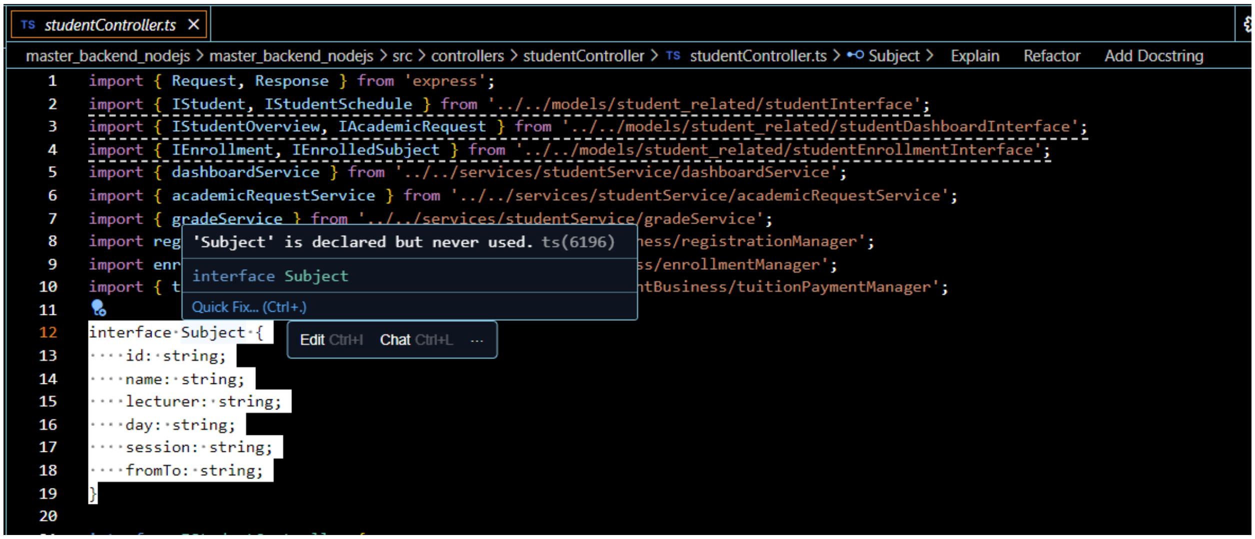
- some routes expect studentID in reg.params others in req.body

(Controller: Uses (req as any).user.id (type unknown)

Service: Expects studentId as string in IStudent interface)

-controllers expect IStudent but have raw data?

- declare new interface no use? the current?


studentController.ts
master_backend_nodejs > master_backend_nodejs > src > controllers > studentController > studentController.ts > Subject >
Explain
Refactor
Add Docstring
Quick Fix... (Ctrl+.)
Edit Ctrl+I
Chat Ctrl+L
···

```
import { Request, Response } from 'express';
import { IStudent, IStudentSchedule } from '../../models/student_related/studentInterface';
import { IStudentOverview, IAcademicRequest } from '../../models/student_related/studentDashboardInterface';
import { IEnrollment, IEnrolledSubject } from '../../models/student_related/studentEnrollmentInterface';
import { dashboardService } from '../../services/studentService/dashboardService';
import { academicRequestService } from '../../services/studentService/academicRequestService';
import { gradeService } from '../../services/studentService/gradeService';
import reg                                              ness/registrationManager';
import enr                                              ss/enrollmentManager';
import { t                                              ntBusiness/tuitionPaymentManager';

interface Subject {
    id: string;
    name: string;
    lecturer: string;
    day: string;
    session: string;
    fromTo: string;
}

```

'Subject' is declared but never used. ts(6196)

interface Subject

- controller doesn't validate the shape of req.user before accessing id

- no TypeScript interface for the <u>authenticated user object (need check)</u>

- Interface for Request/Response?

+Business:

-dashboardmanager: logic? (recheck others)

-

+Service:

-dashboardService: in-memory, the same to enrollmentManager, registrationManager, tuitionPaymentManager

-gradeService: in-memory, the same to studentService, subjectRegistrationService

Financial:

+Controller level:

- check import for duplicate

+Business:

- Payment Processing: no audit trail for payment updates, no validation of payment amounts against tuition

- Tuition Calculation: Calculation logic not clearly defined (need check), No validation of tuition settings

+Service:

- in-memory

- interface for payment data structure? (need check)

- Service layer returns loosely typed data

Academic:

+Controller level:

-

+Business:

- No validation of credit values

- No validation of time formats

+Service level:

- Database layer returns any[] from queries

- JSON fields (schedule, prerequisite_subjects) need proper typing

Admin:

+Controller level:

- adminController: getUserManagement has pagination parameters but no implement

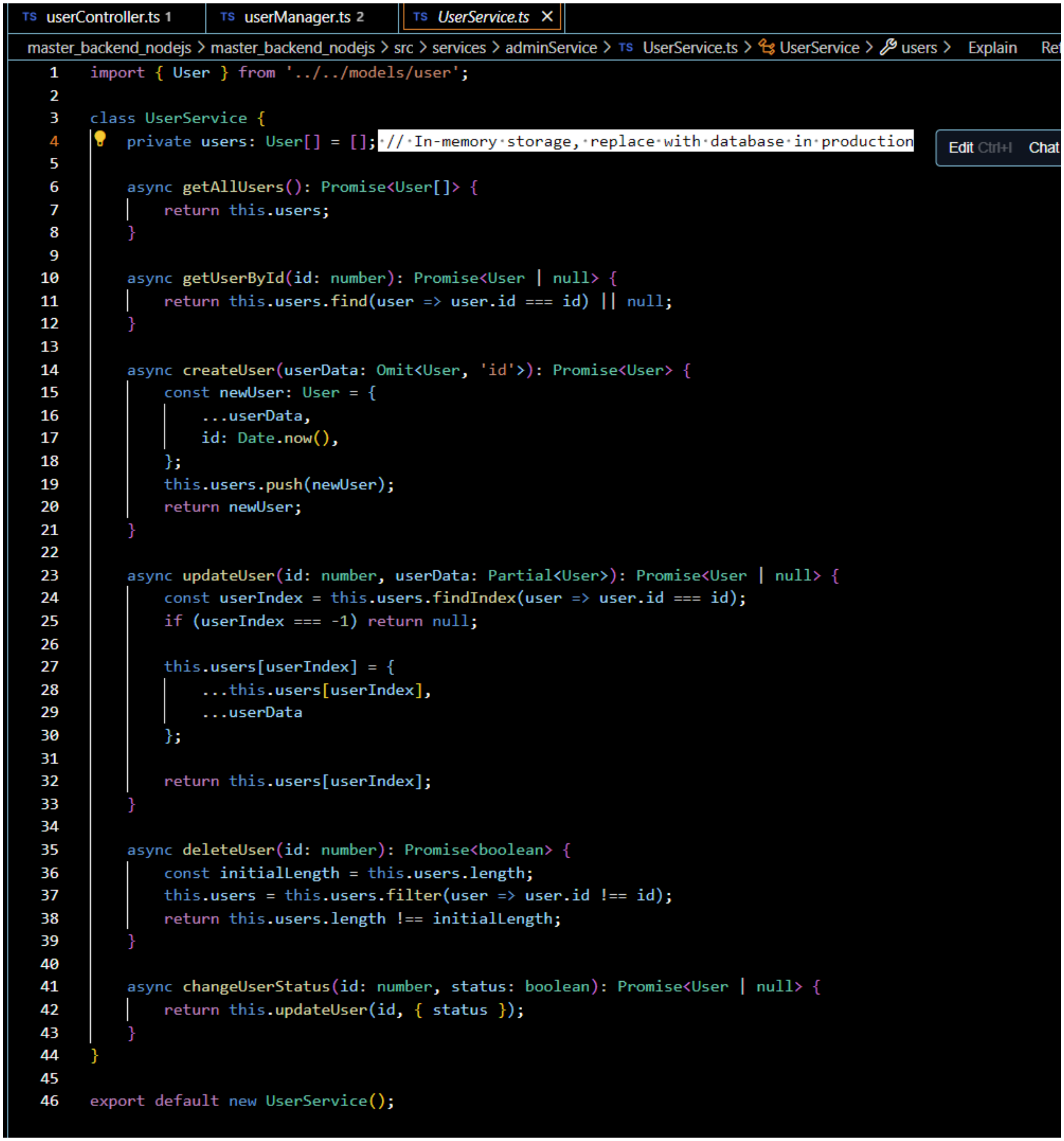
- user: in-memory?!

+Business:

+Service:

- Service layer returns loosely typed data

- No cascading deletes


TS userController.ts 1
TS userManager.ts 2
TS UserService.ts
master_backend_nodejs > master_backend_nodejs > src > services > adminService > TS UserService.ts > UserService > users > Explain Ref
Edit Ctrl+I
Chat

```typescript
import { User } from '../../models/user';

class UserService {
    private users: User[] = []; // In-memory storage, replace with database in production

    async getAllUsers(): Promise<User[]> {
        return this.users;
    }

    async getUserById(id: number): Promise<User | null> {
        return this.users.find(user => user.id === id) || null;
    }

    async createUser(userData: Omit<User, 'id'>): Promise<User> {
        const newUser: User = {
            ...userData,
            id: Date.now(),
        };
        this.users.push(newUser);
        return newUser;
    }

    async updateUser(id: number, userData: Partial<User>): Promise<User | null> {
        const userIndex = this.users.findIndex(user => user.id === id);
        if (userIndex === -1) return null;

        this.users[userIndex] = {
            ...this.users[userIndex],
            ...userData
        };

        return this.users[userIndex];
    }

    async deleteUser(id: number): Promise<boolean> {
        const initialLength = this.users.length;
        this.users = this.users.filter(user => user.id !== id);
        return this.users.length !== initialLength;
    }

    async changeUserStatus(id: number, status: boolean): Promise<User | null> {
        return this.updateUser(id, { status });
    }
}

export default new UserService();
```