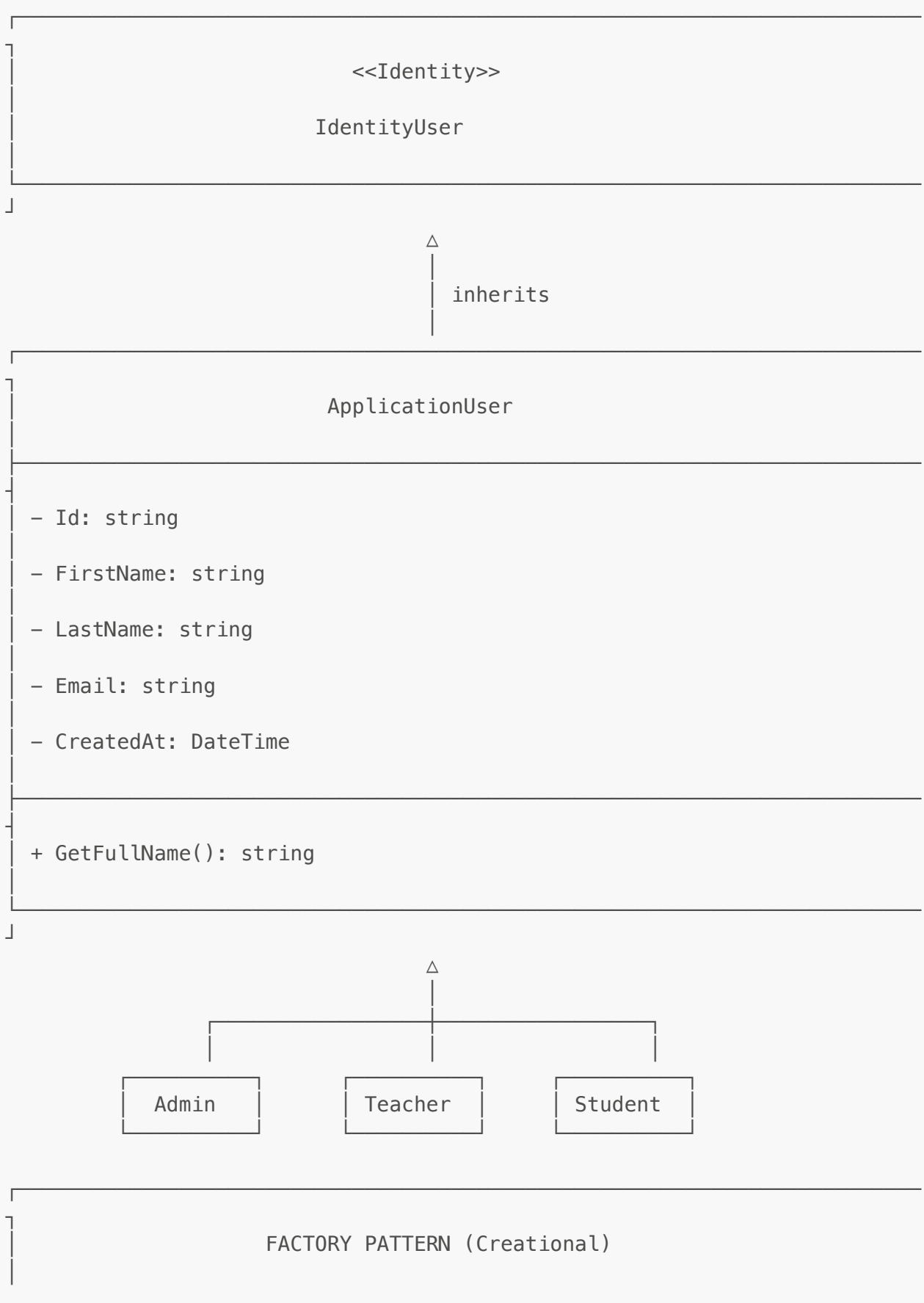
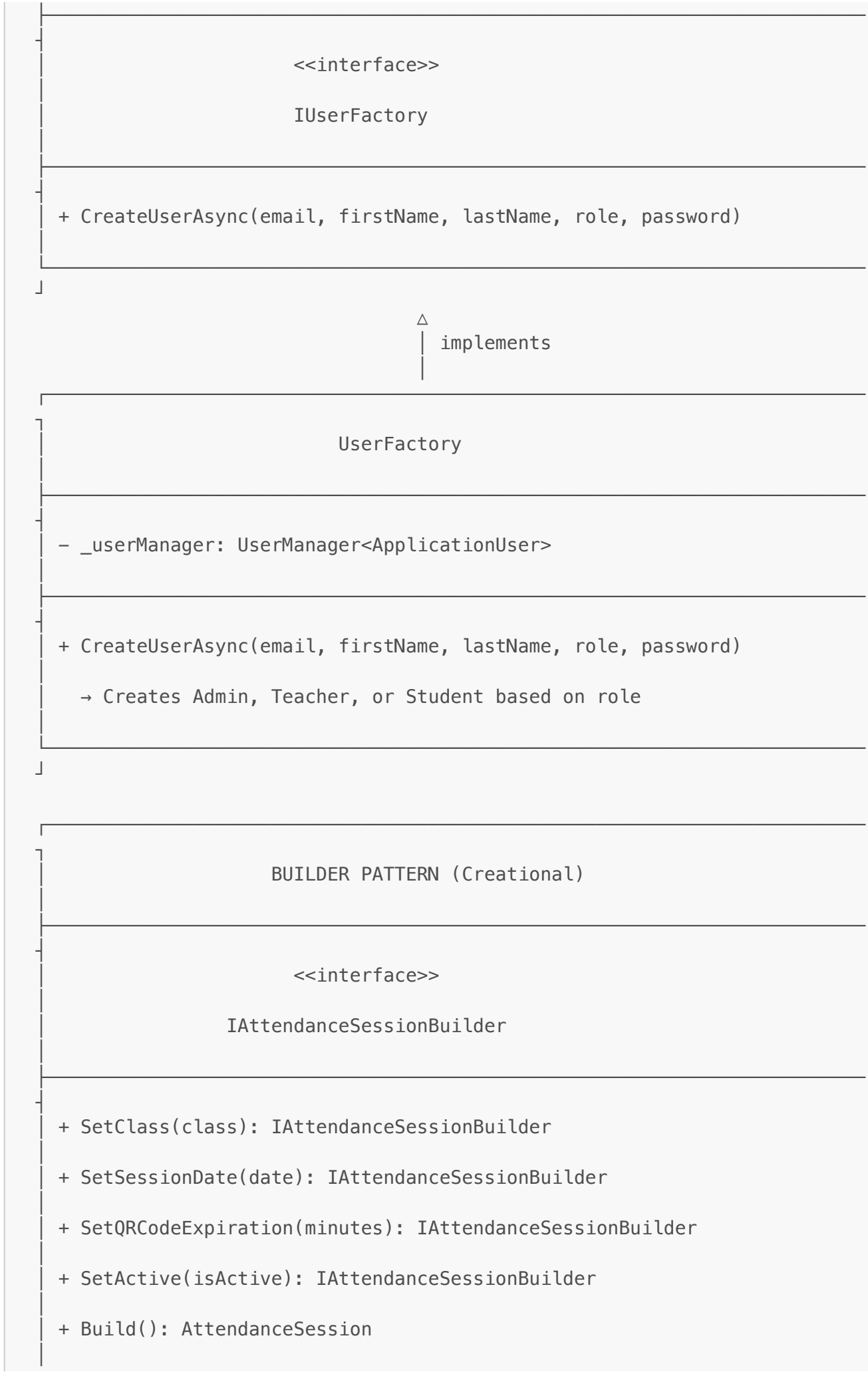
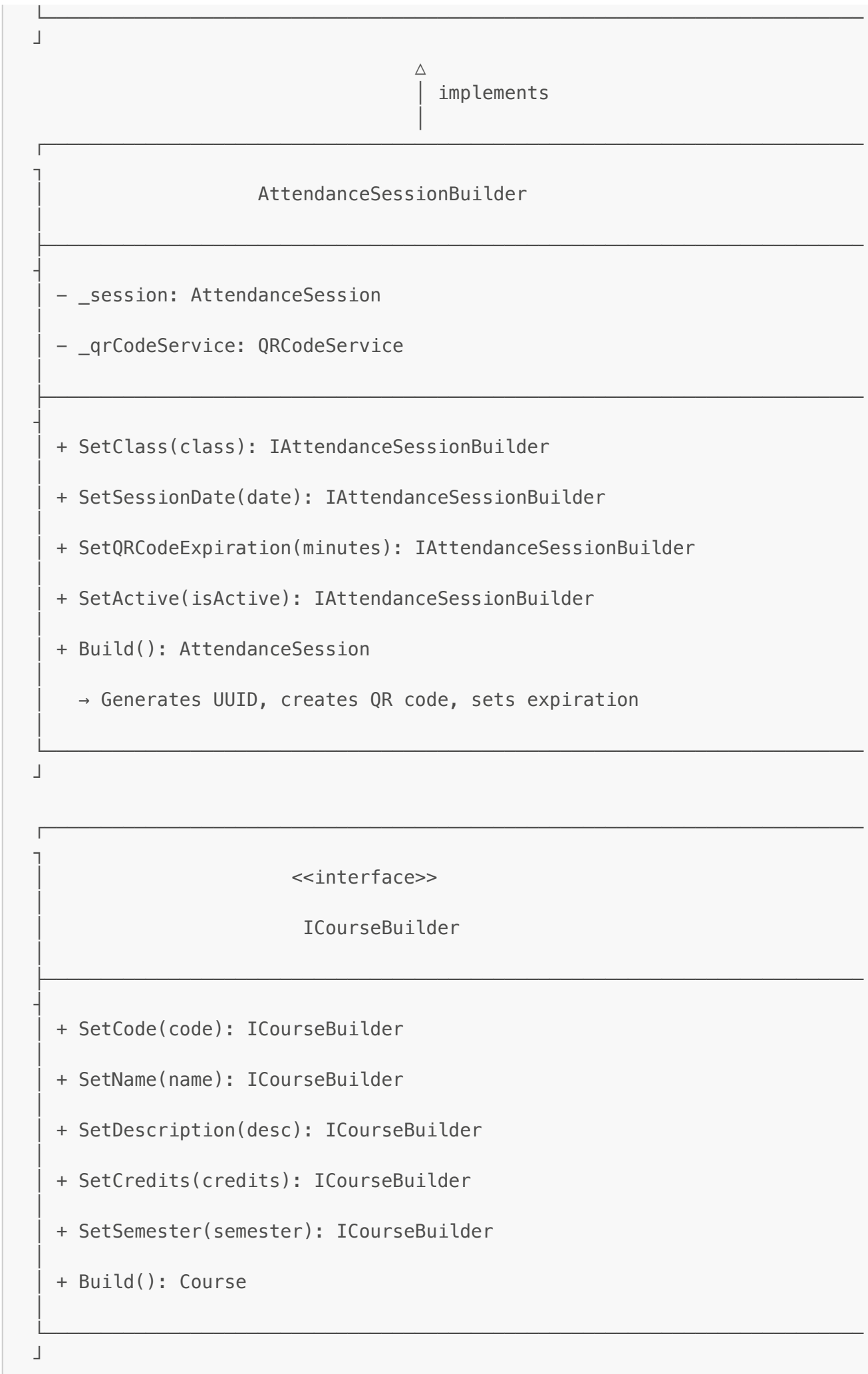
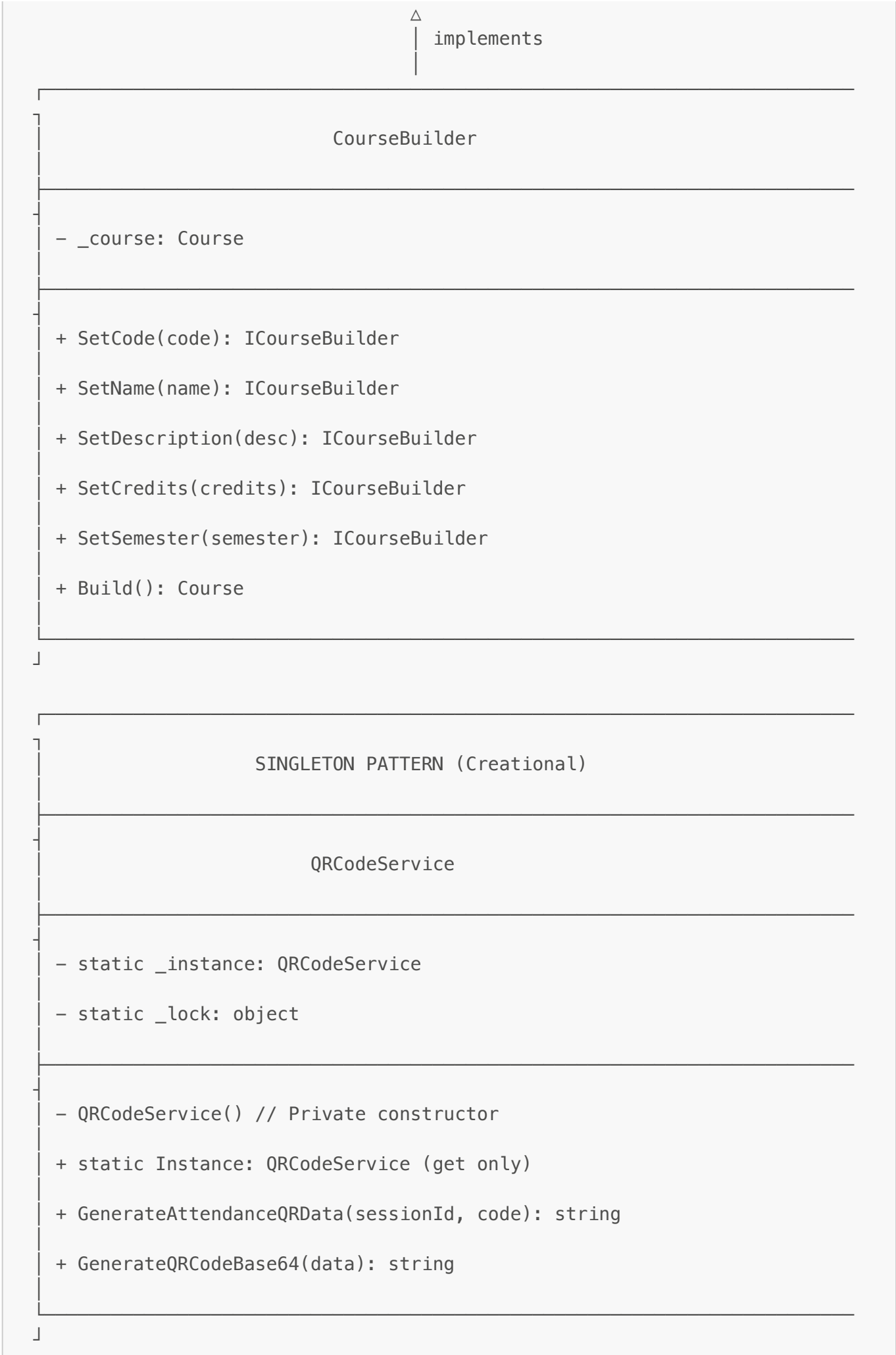
UML Diagrams for Attendance System

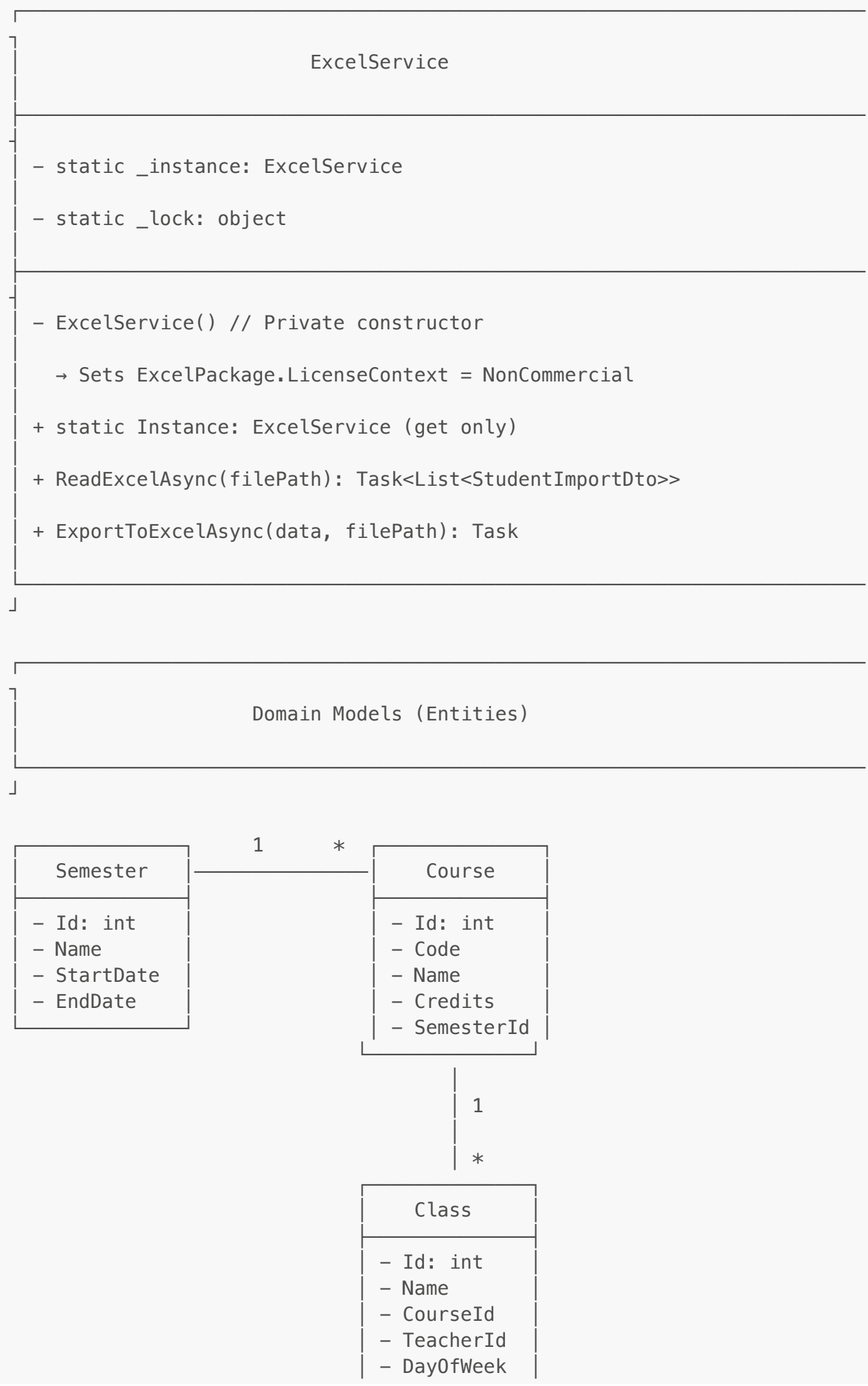
Class Diagram (Main Components)

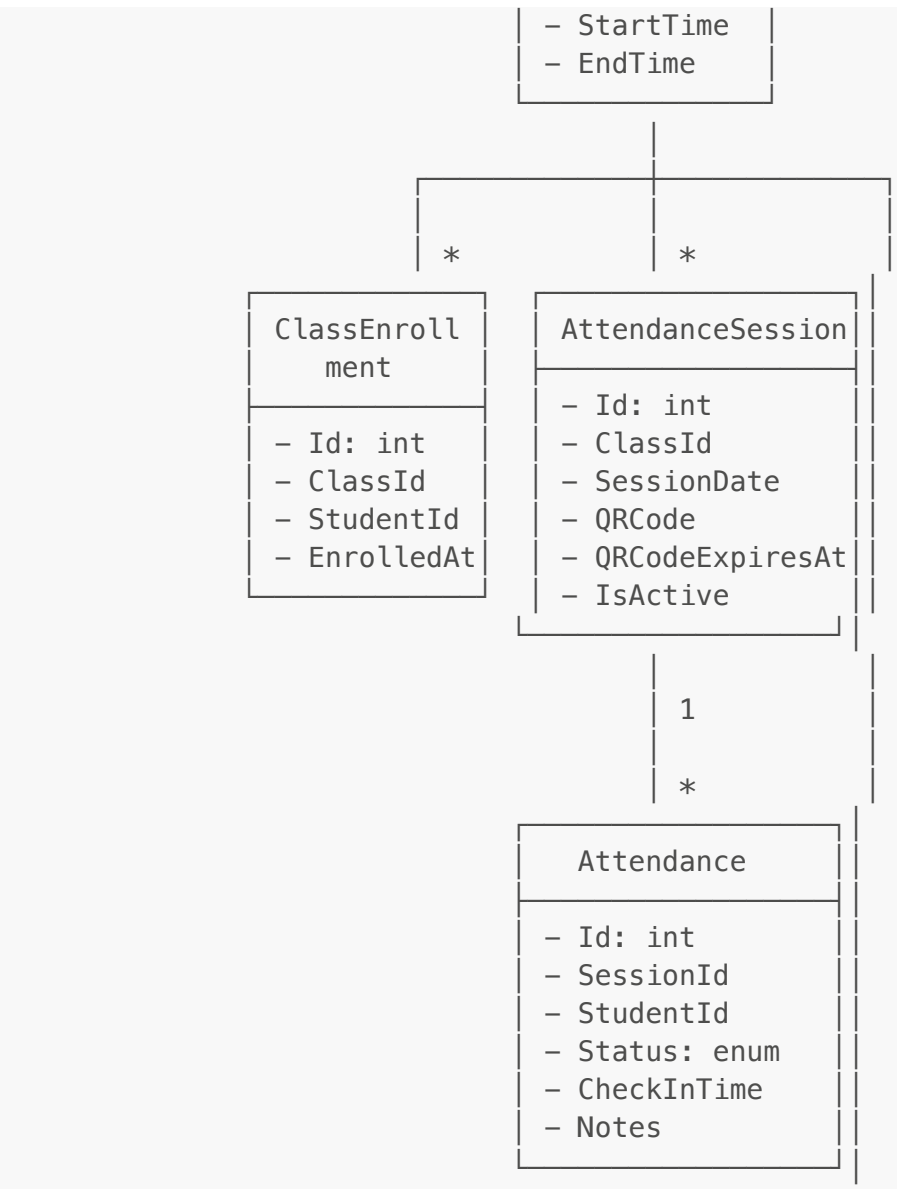



<<interface>>
IUserFactory
+ CreateUserAsync(email, firstName, lastName, role, password)
implements
UserFactory
- _userManager: UserManager<ApplicationUser>
+ CreateUserAsync(email, firstName, lastName, role, password)
→ Creates Admin, Teacher, or Student based on role
BUILDER PATTERN (Creational)
<<interface>>
IAttendanceSessionBuilder
+ SetClass(class): IAttendanceSessionBuilder
+ SetSessionDate(date): IAttendanceSessionBuilder
+ SetQRCodeExpiration(minutes): IAttendanceSessionBuilder
+ SetActive(isActive): IAttendanceSessionBuilder
+ Build(): AttendanceSession


△
implements
AttendanceSessionBuilder
- _session: AttendanceSession
- _qrCodeService: QRCodeService
+ SetClass(class): IAttendanceSessionBuilder
+ SetSessionDate(date): IAttendanceSessionBuilder
+ SetQRCodeExpiration(minutes): IAttendanceSessionBuilder
+ SetActive(isActive): IAttendanceSessionBuilder
+ Build(): AttendanceSession
→ Generates UUID, creates QR code, sets expiration
<<interface>>
ICourseBuilder
+ SetCode(code): ICourseBuilder
+ SetName(name): ICourseBuilder
+ SetDescription(desc): ICourseBuilder
+ SetCredits(credits): ICourseBuilder
+ SetSemester(semester): ICourseBuilder
+ Build(): Course


△
implements
CourseBuilder
- _course: Course
+ SetCode(code): ICourseBuilder
+ SetName(name): ICourseBuilder
+ SetDescription(desc): ICourseBuilder
+ SetCredits(credits): ICourseBuilder
+ SetSemester(semester): ICourseBuilder
+ Build(): Course
SINGLETON PATTERN (Creational)
QRCodeService
- static _instance: QRCodeService
- static _lock: object
- QRCodeService() // Private constructor
+ static Instance: QRCodeService (get only)
+ GenerateAttendanceQRData(sessionId, code): string
+ GenerateQRCodeBase64(data): string

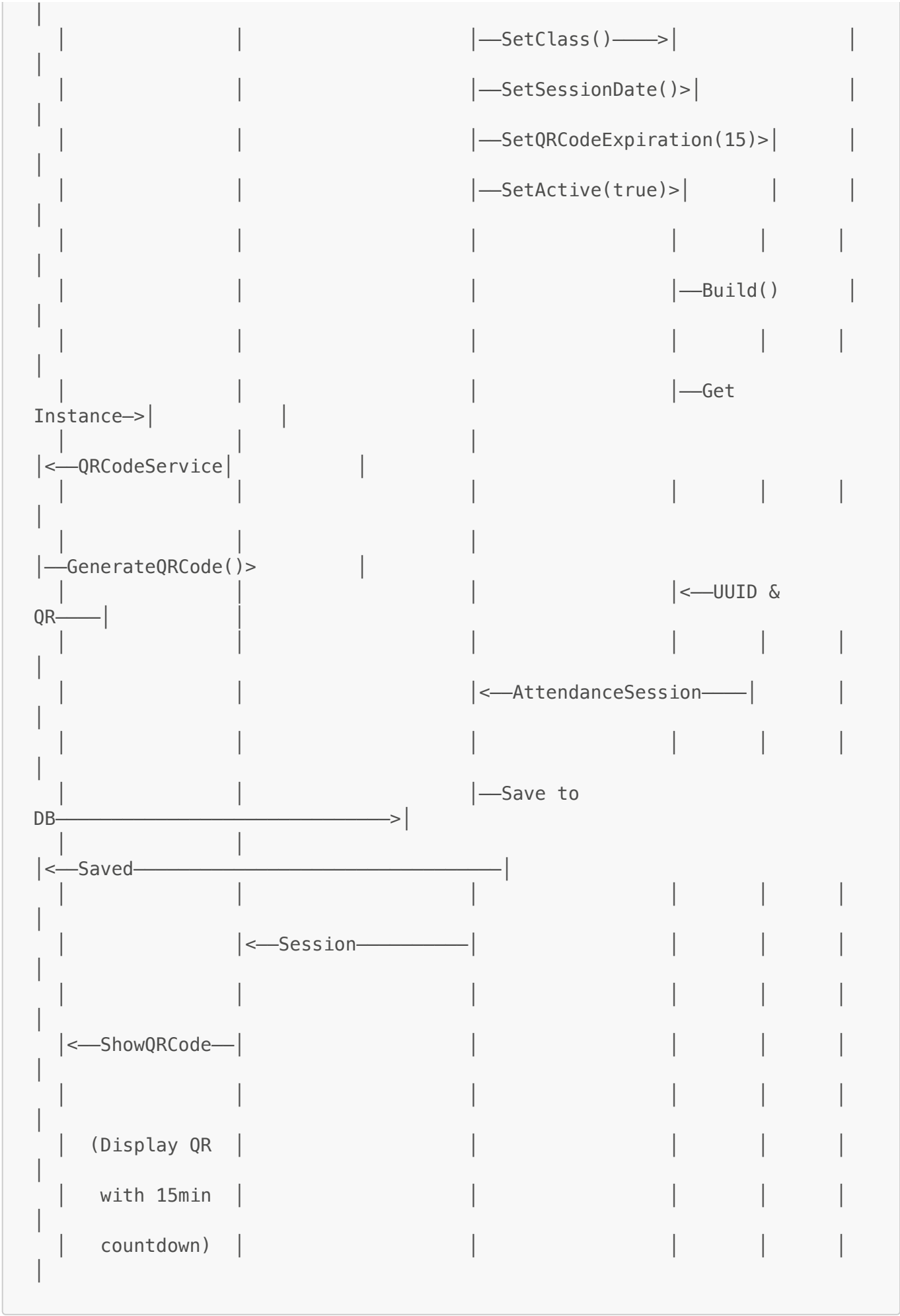

ExcelService
- static _instance: ExcelService
- static _lock: object
- ExcelService() // Private constructor
→ Sets ExcelPackage.LicenseContext = NonCommercial
+ static Instance: ExcelService (get only)
+ ReadExcelAsync(filePath): Task<List<StudentImportDto>>
+ ExportToExcelAsync(data, filePath): Task
Domain Models (Entities)
Semester
- Id: int
- Name
- StartDate
- EndDate
1
*
Course
- Id: int
- Code
- Name
- Credits
- SemesterId
1
*
Class
- Id: int
- Name
- CourseId
- TeacherId
- DayOfWeek



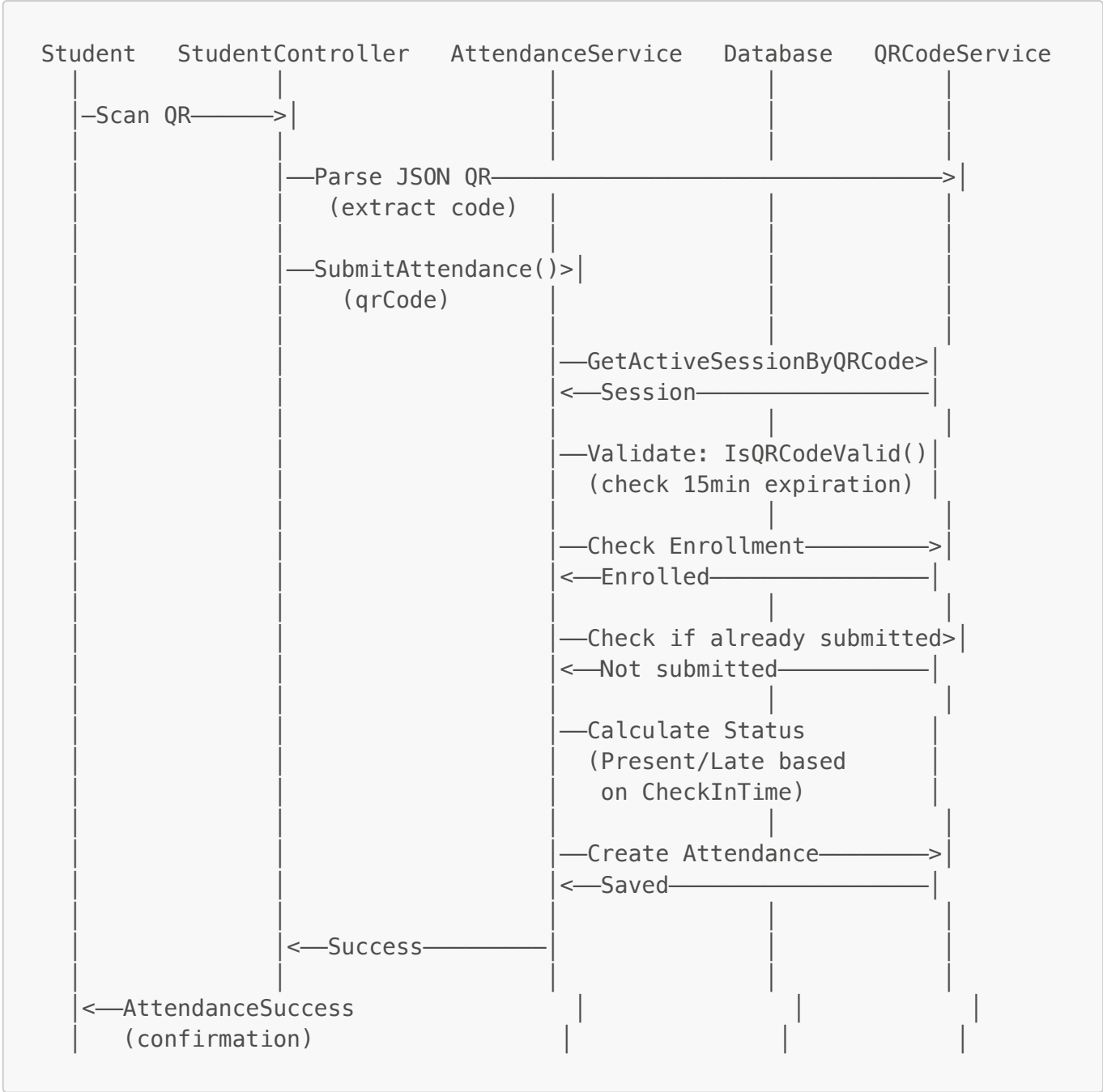

```
AttendanceStatus (enum):
- Present
- Late
- Absent
- Excused
```

Sequence Diagram 1: Start Attendance Session (Teacher)

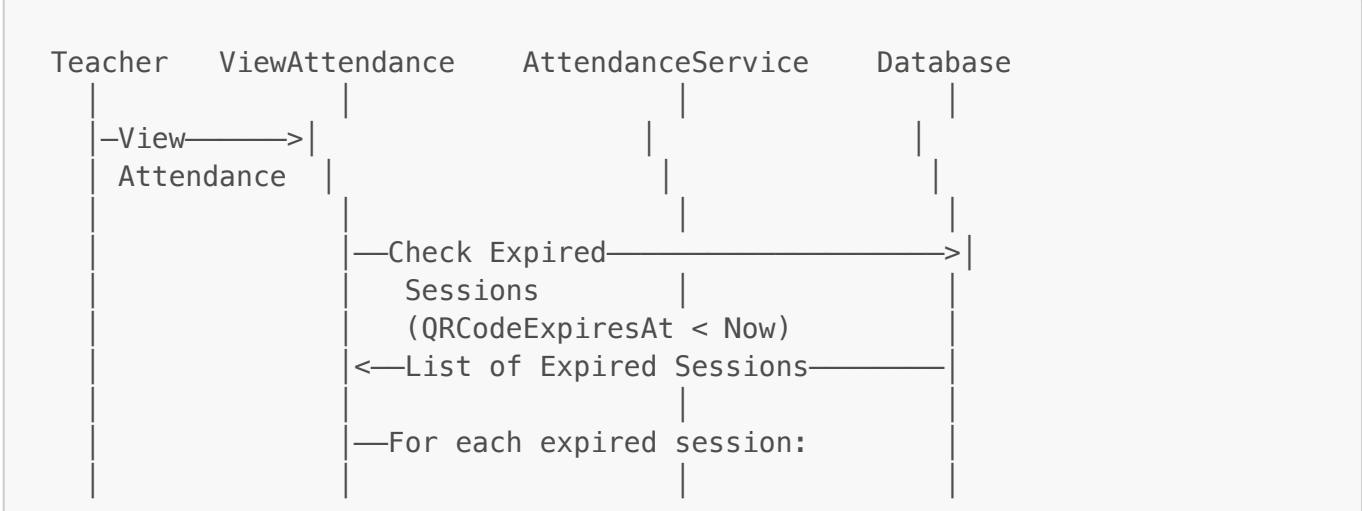
```
Teacher    TeacherController   AttendanceService   SessionBuilder
QRCodeService   Database
  |               |                   |                |              |
|
  |──StartSession─>|                   |                |              |
|
  |               |                   |                |              |
|
  |               |──CreateSessionAsync─>|              |              |
|
  |               |                   |                |              |
```

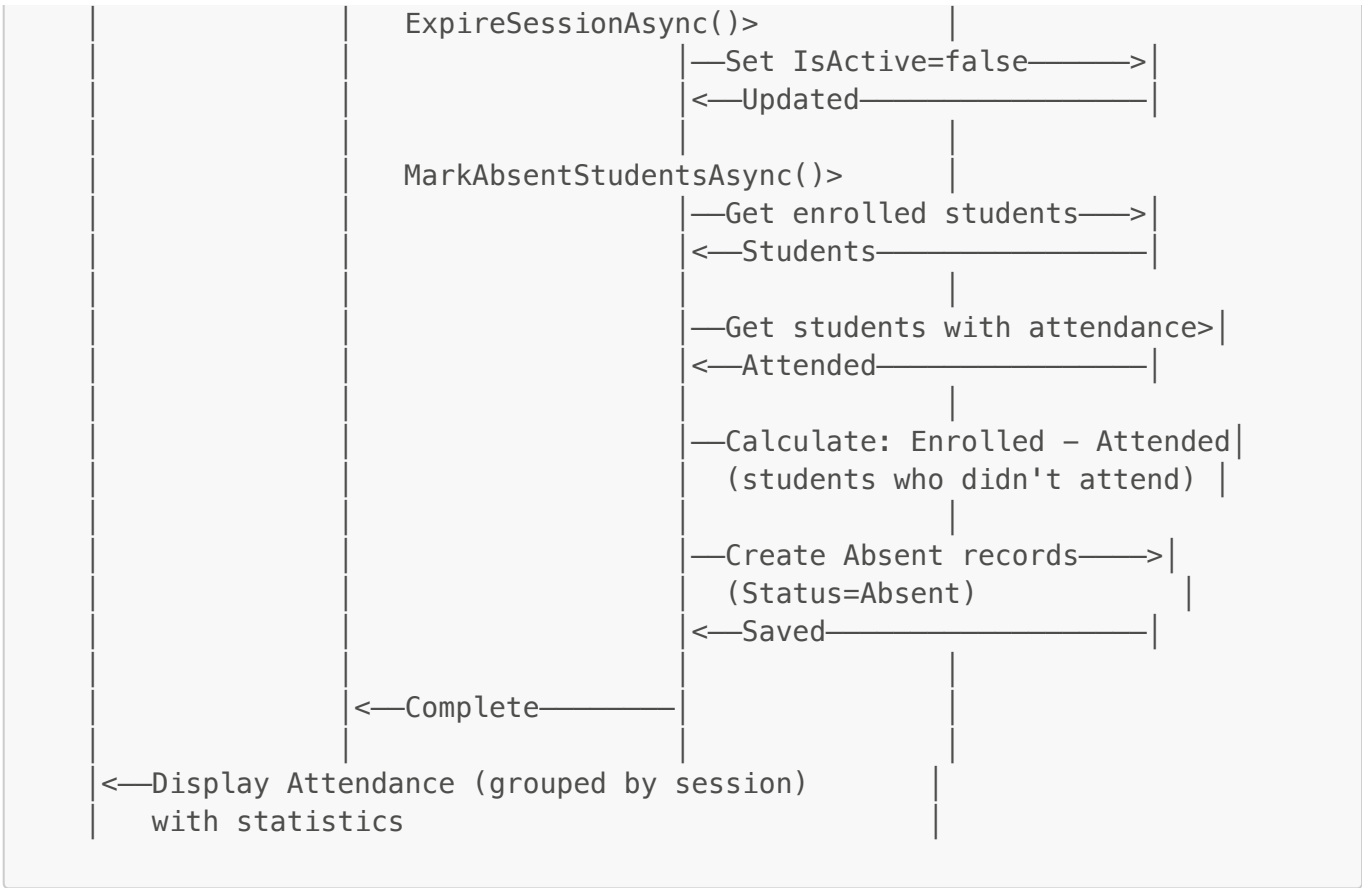

|
| |-SetClass()---->| |
|
| |-SetSessionDate()>| |
|
| |-SetQRCodeExpiration(15)>| |
|
| |-SetActive(true)>| | |
|
| | | | | |
|
| |-Build() |
|
| | | | | |
|
| |-Get
Instance->| |
<-QRCodeService	
-GenerateQRCode()>
			<-UUID &
QR-----	
	|<-AttendanceSession-----|
	|-Save to
DB------------------------------>|
<-Saved--------------------------------------
	<-Session----------			
<-ShowQRCode-				
(Display QR				
with 15min				
countdown)				

Sequence Diagram 2: Submit Attendance (Student with QR Scan)



Sequence Diagram 3: Auto-Expire Session & Mark Absent Students





Design Pattern Usage Summary

1. **Factory Pattern** - User Creation

- **Problem**: Need to create different types of users (Admin, Teacher, Student) with proper role assignment
- **Solution**: UserFactory creates users based on role parameter
- **Benefits**:
  - Centralized user creation logic
  - Ensures proper role assignment
  - Reduces code duplication across controllers

2. **Builder Pattern** - Complex Object Construction

- **Problem**: AttendanceSession and Course require many parameters and complex initialization
- **Solution**: SessionBuilder and CourseBuilder allow step-by-step construction
- **Benefits**:
  - Readable, fluent API
  - Flexible construction process
  - Automatic QR code generation for sessions
  - Validation before object creation

3. **Singleton Pattern** - Shared Services

- **Problem**: QRCodeService and ExcelService should have only one instance
- **Solution**: Thread-safe singleton with lazy initialization
- **Benefits**:

- Single shared instance reduces memory
- Thread-safe implementation prevents race conditions
- EPPlus license configuration centralized in ExcelService constructor

Why These Patterns?

Your Attendance System uses these patterns because:

1. **Factory** - Simplifies user creation across Admin, Teacher, Student roles
2. **Builder** - Makes complex session/course creation readable and maintainable
3. **Singleton** - Ensures QR code generation and Excel operations share state efficiently

These are **real-world applications** of design patterns, not just academic examples!