Java Programming: Classes and Exceptions

Desenvolvimento de Software e Sistemas Móveis (DSSMV)
Licenciatura em Engenharia de Telecomunicações e Informática
LETI/ISEP

2025/26

Paulo Baltarejo Sousa and Carlos Filipe Freitas
{pbs,caf}@isep.ipp.pt

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

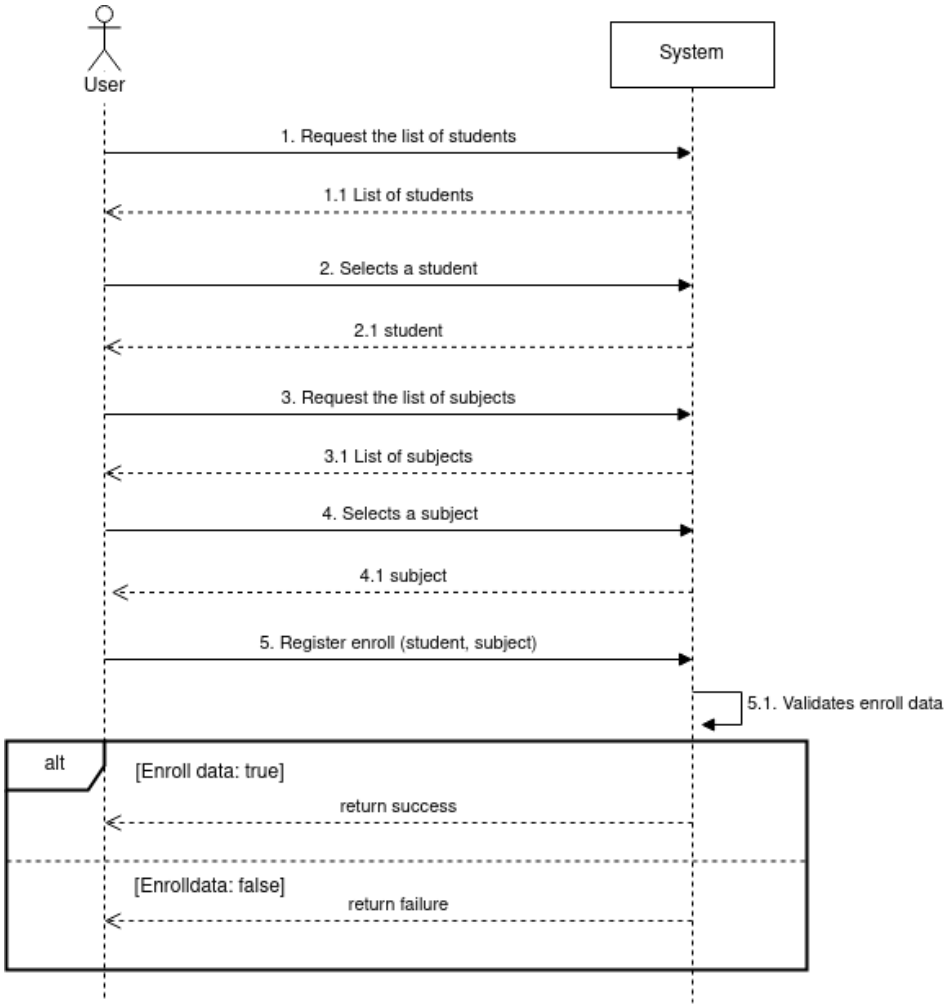
- Presentations found on the internet;
- Books;
- Web sites;
- ...

Outline

Introducing classes

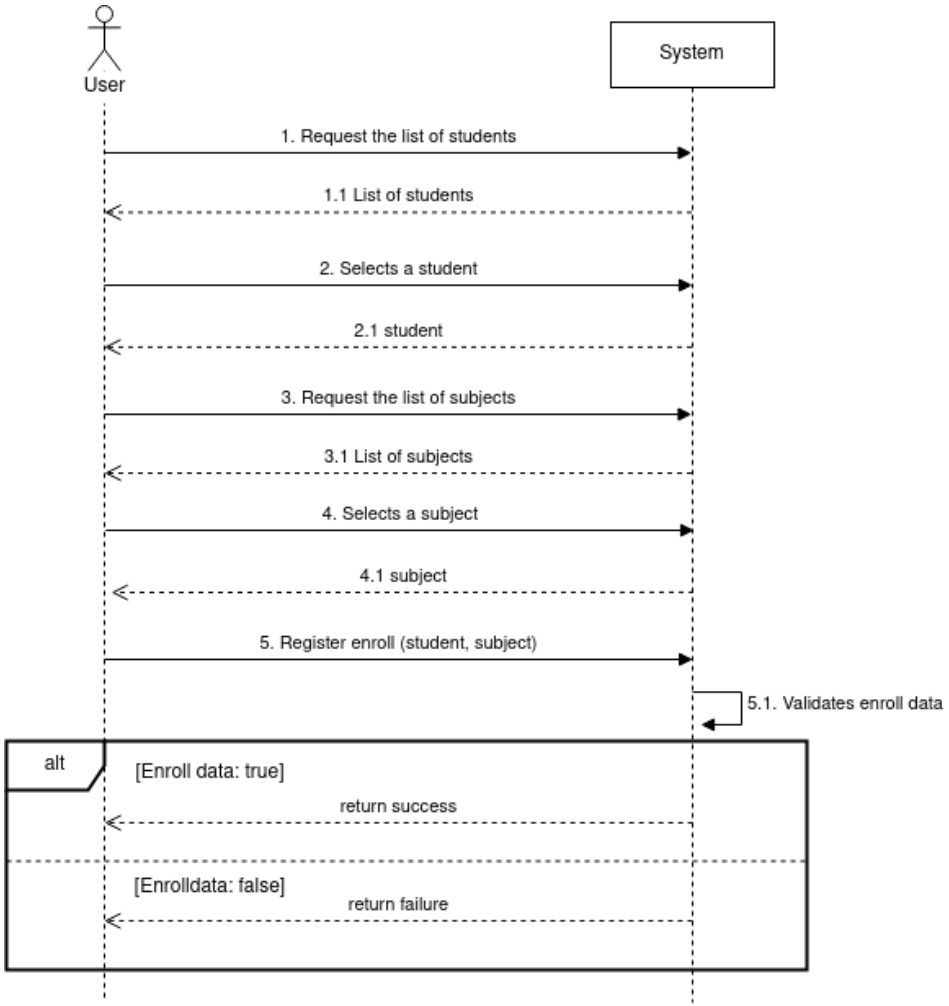
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



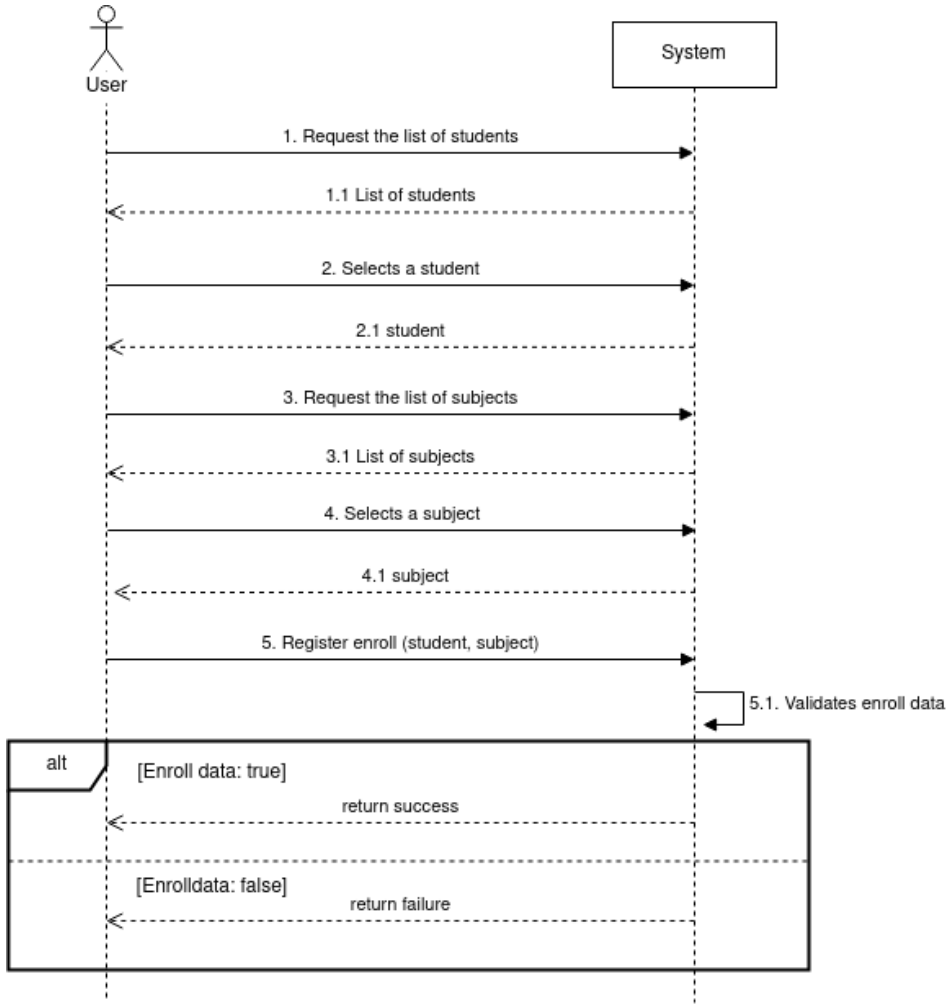
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



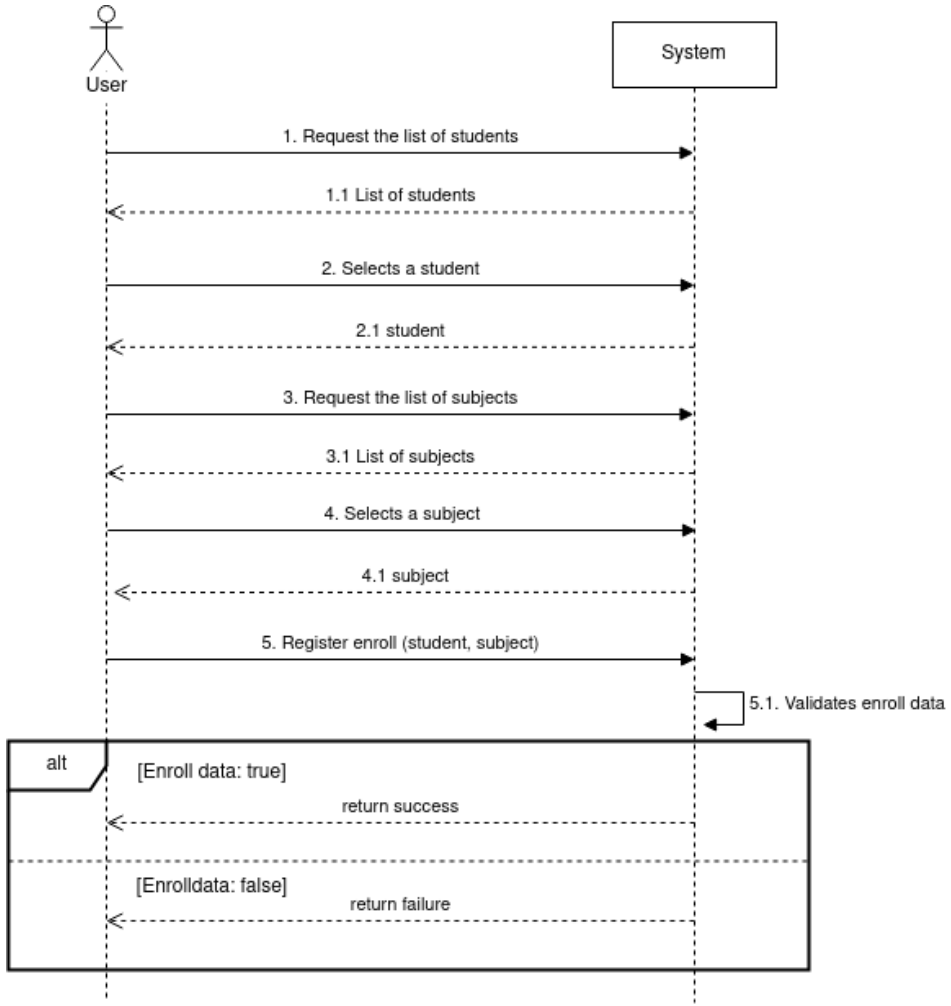
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



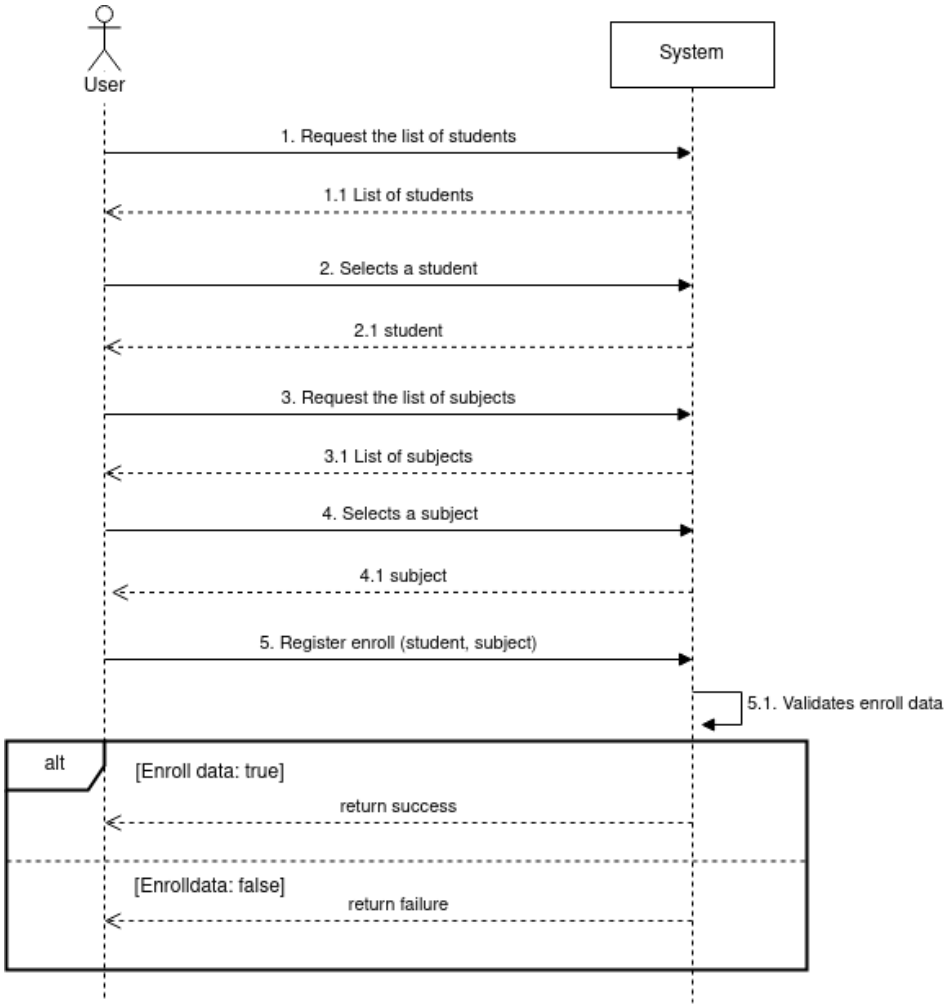
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



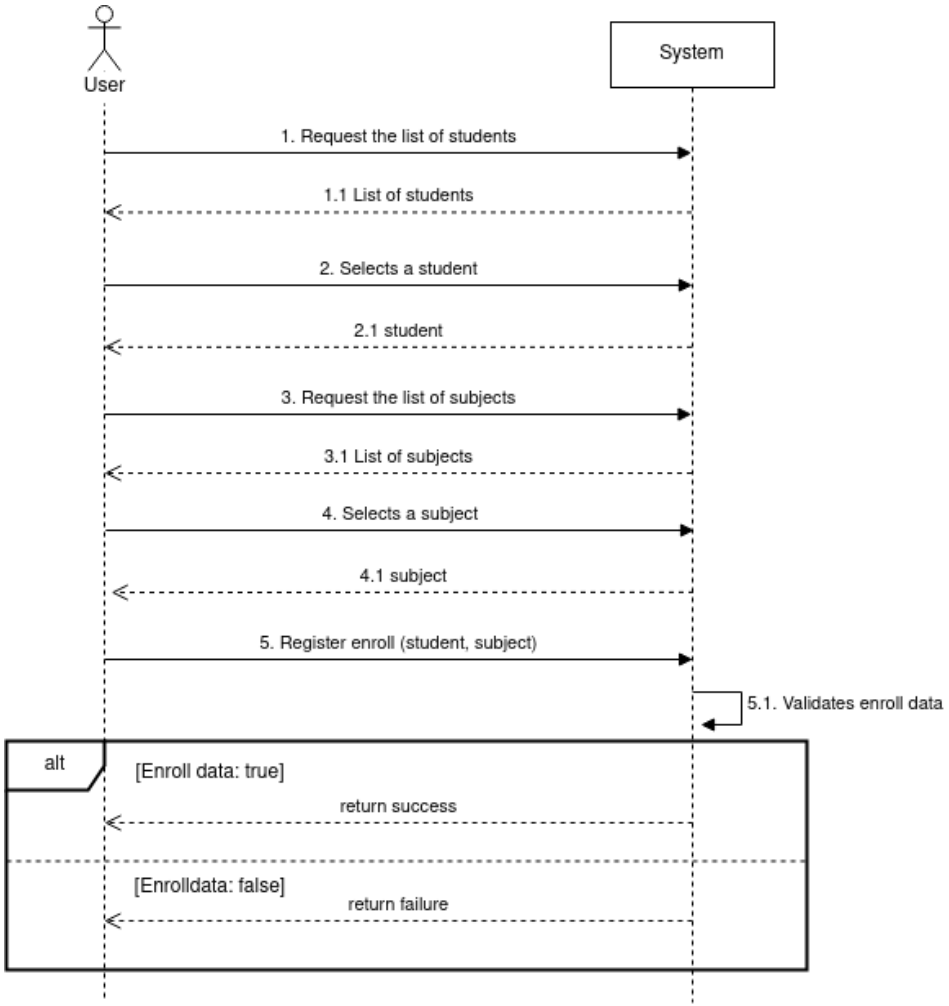
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



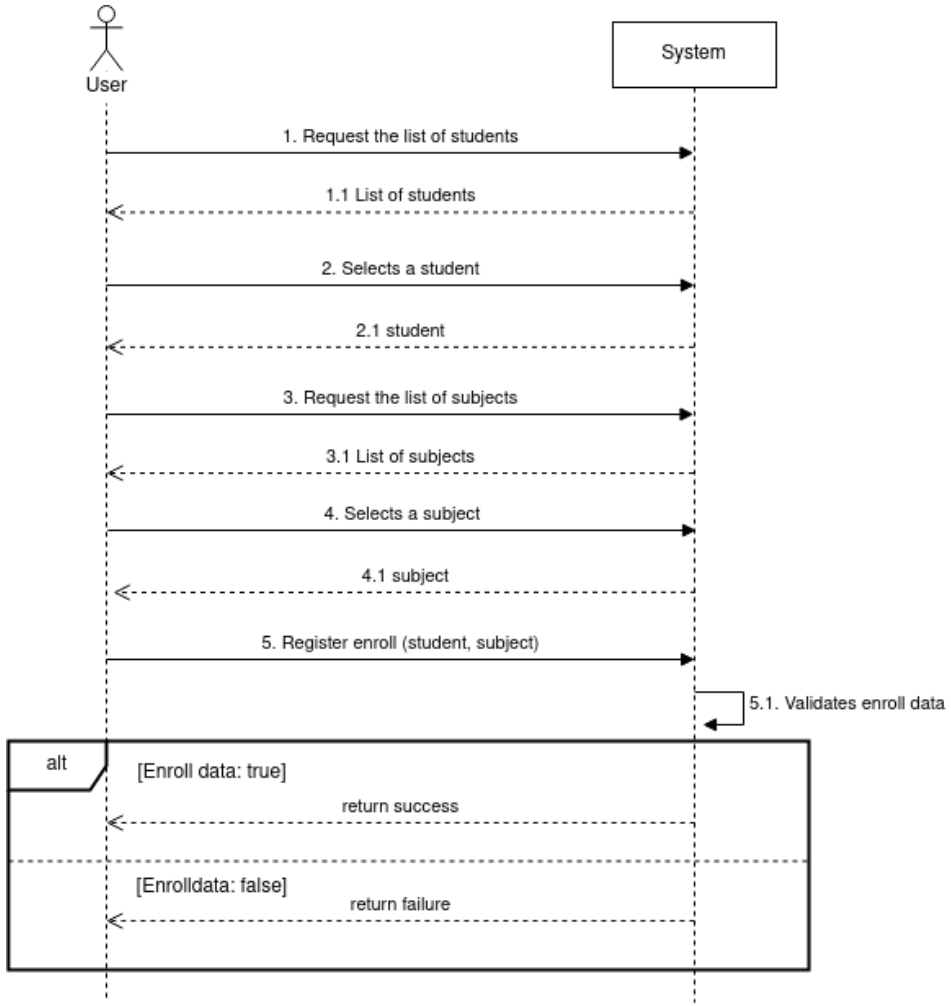
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



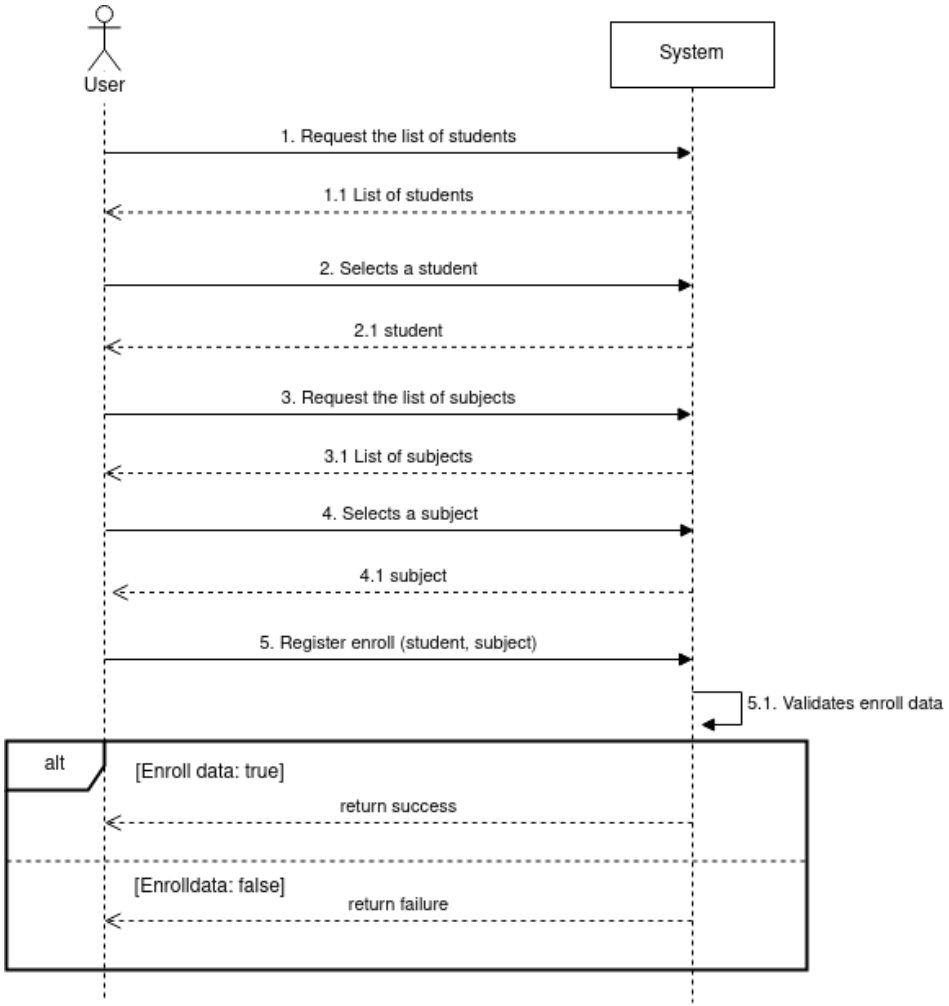
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



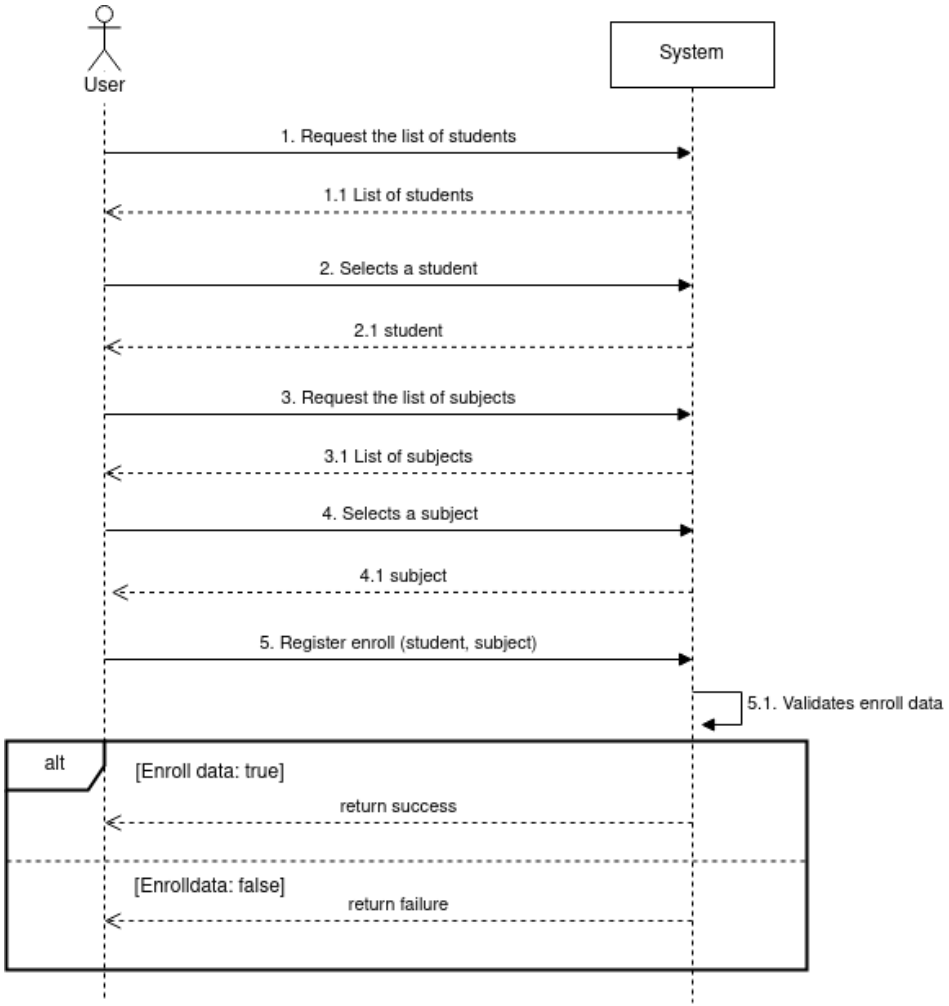
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



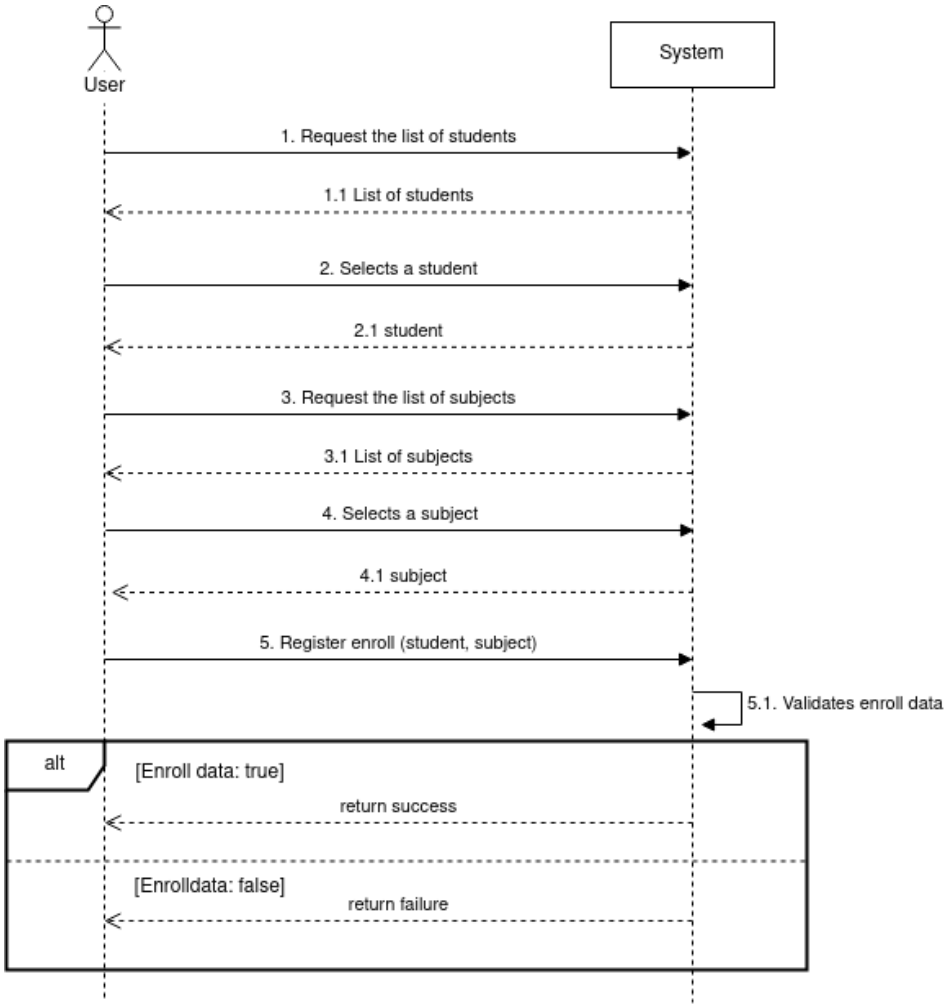
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



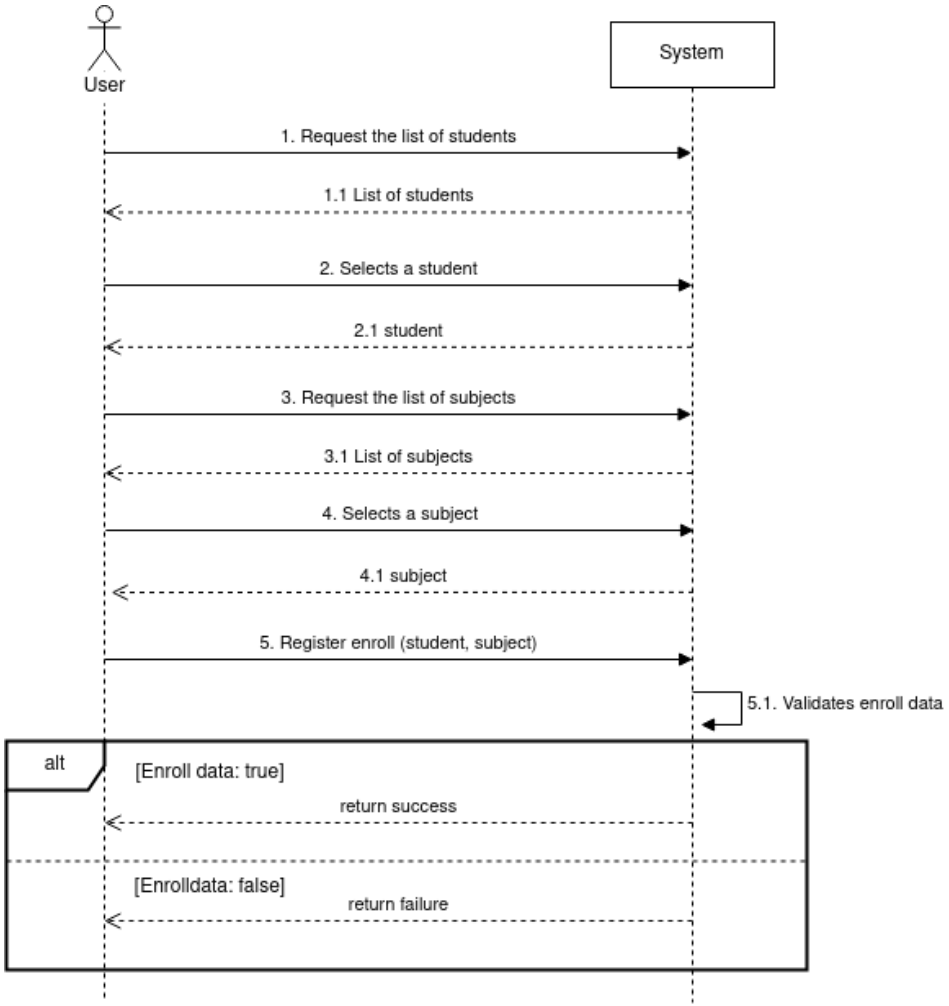
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



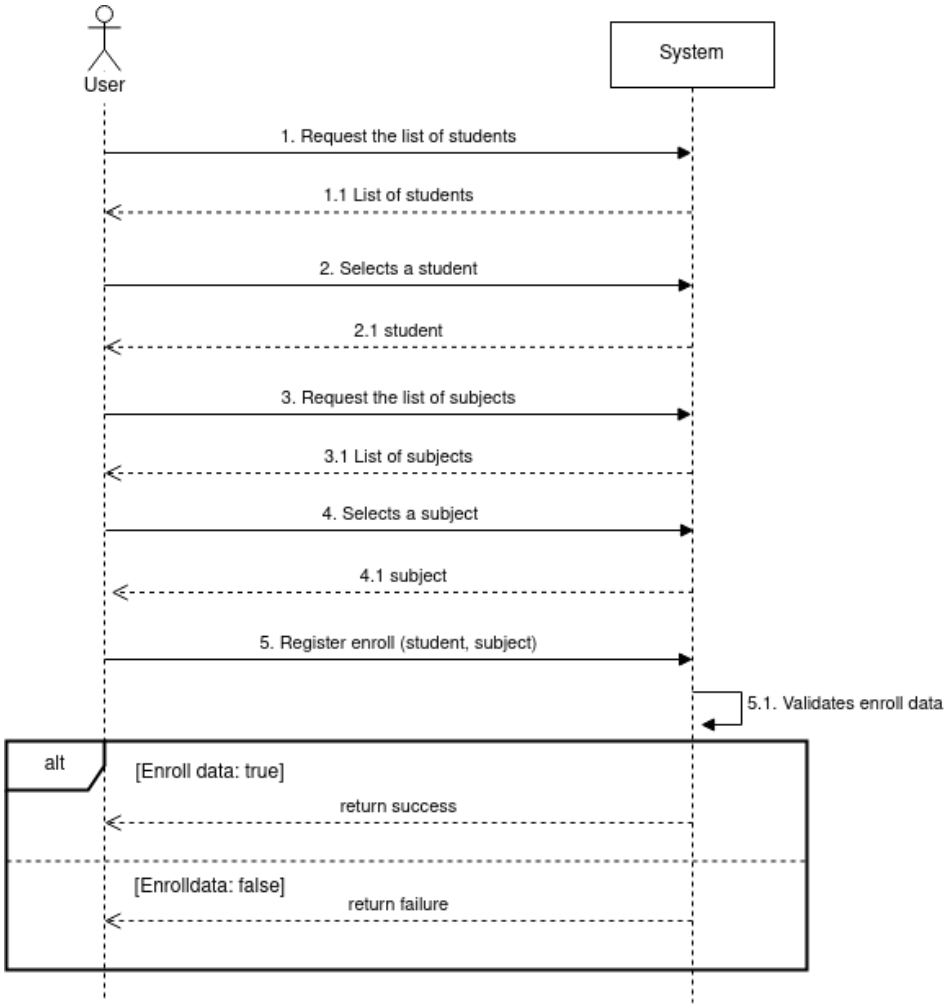
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



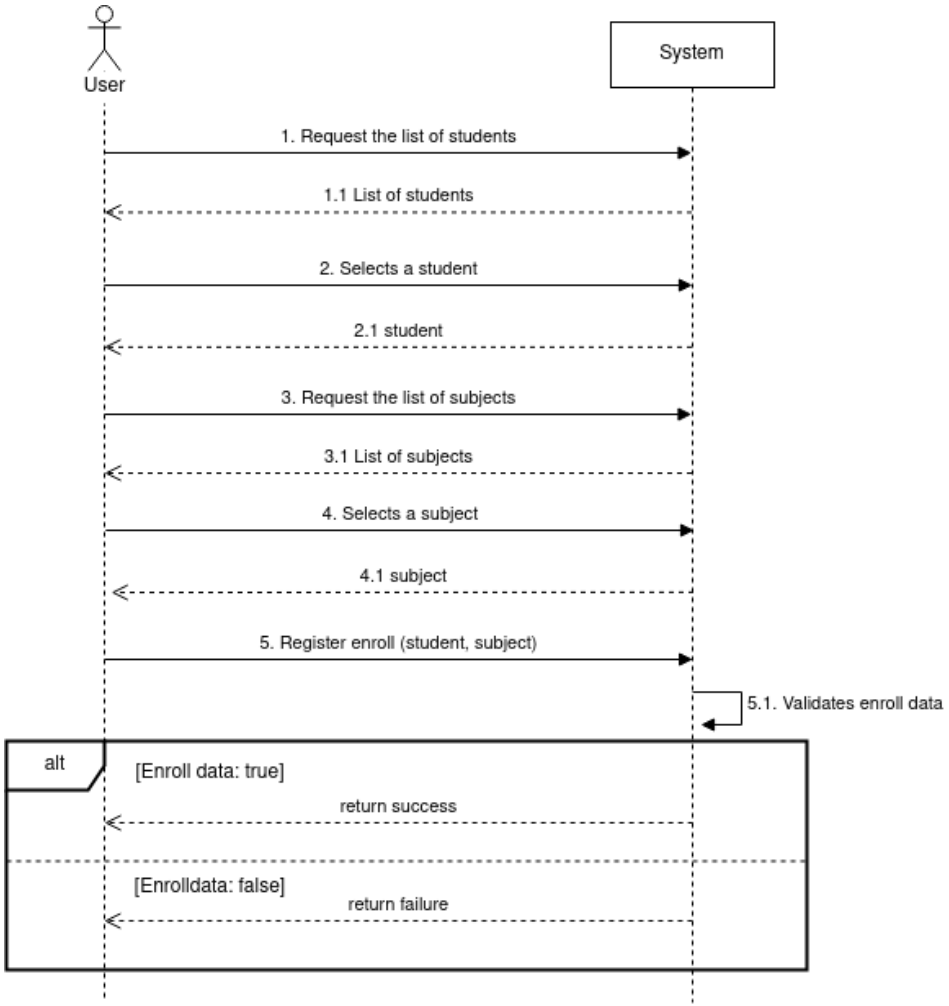
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



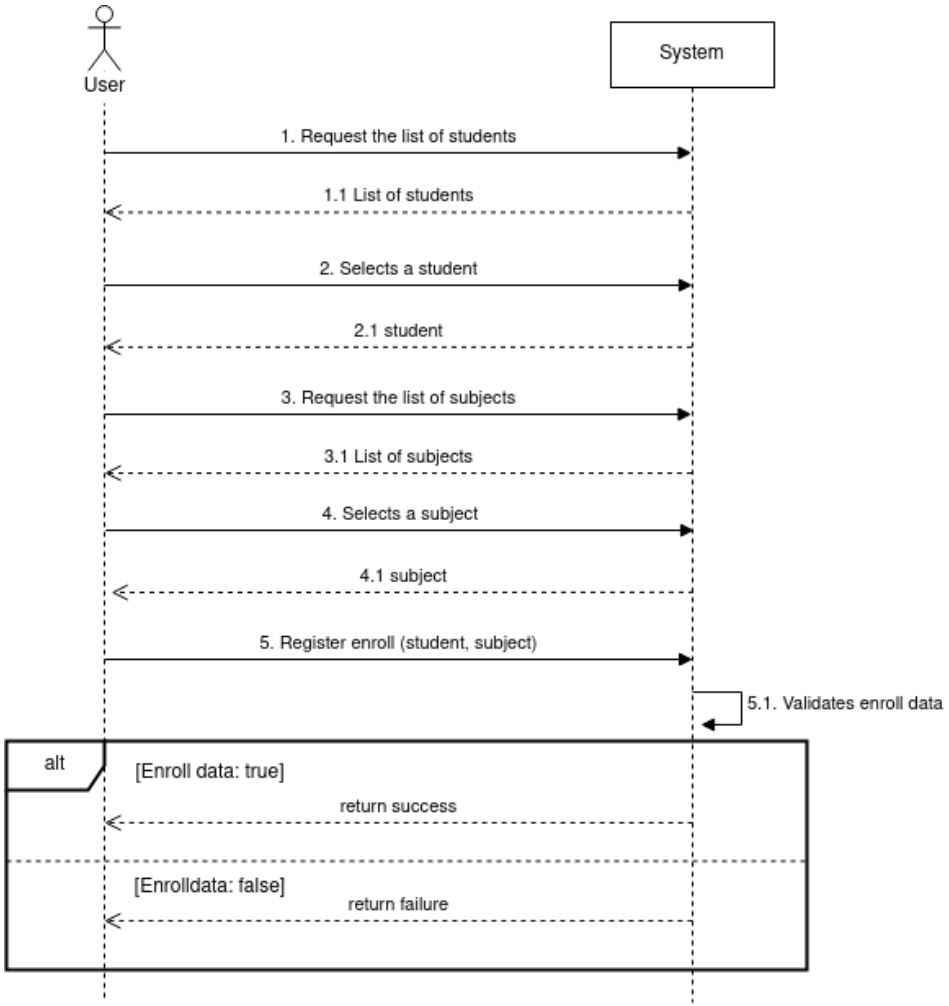
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



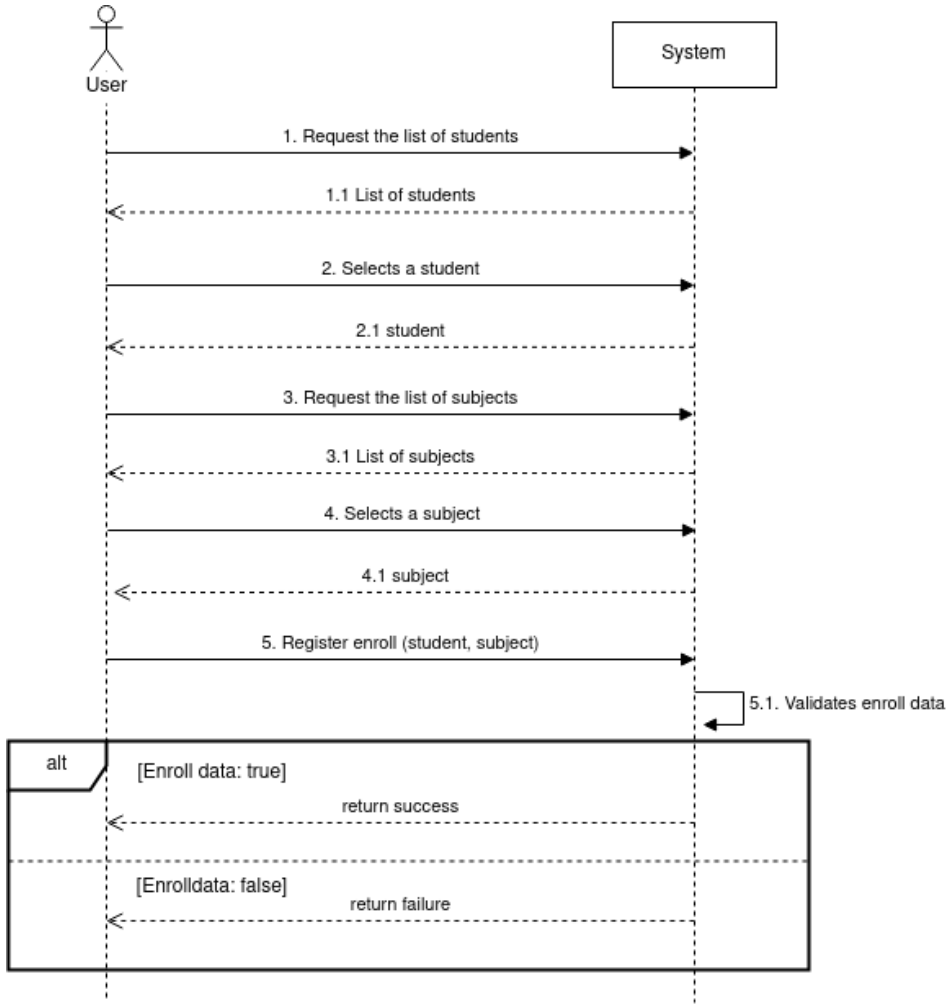
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



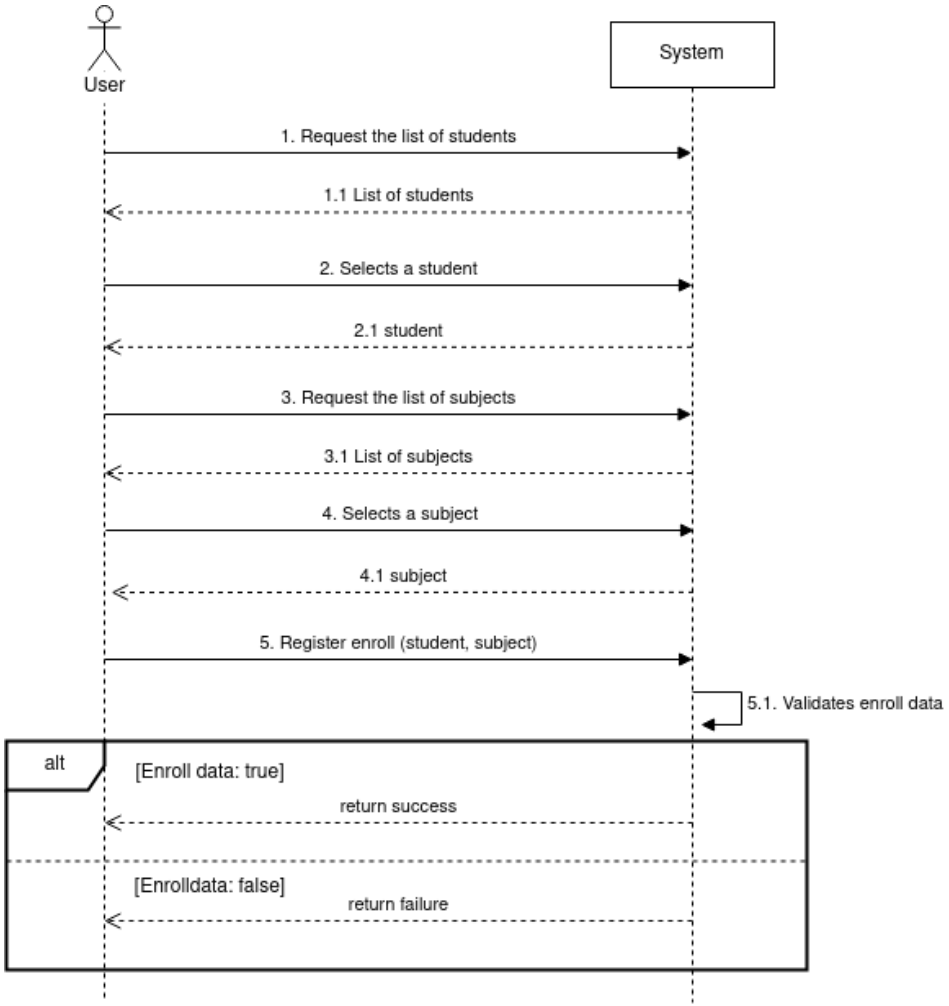
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



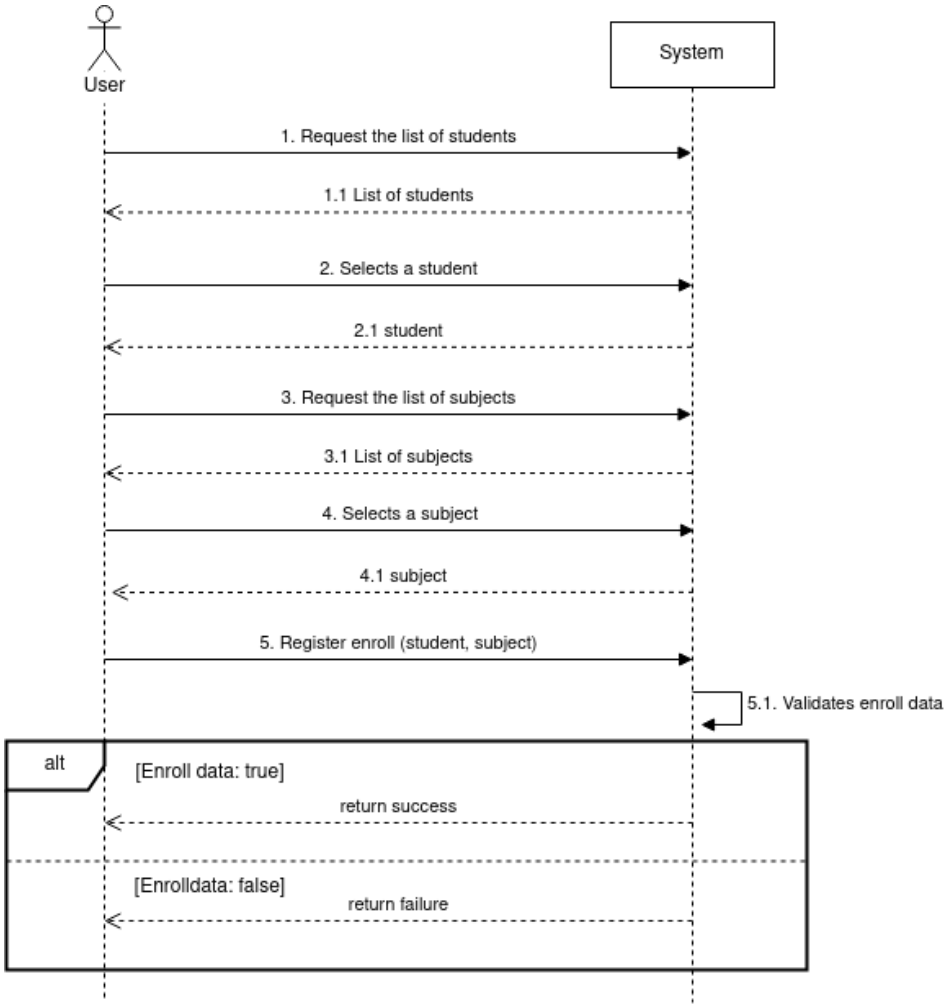
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



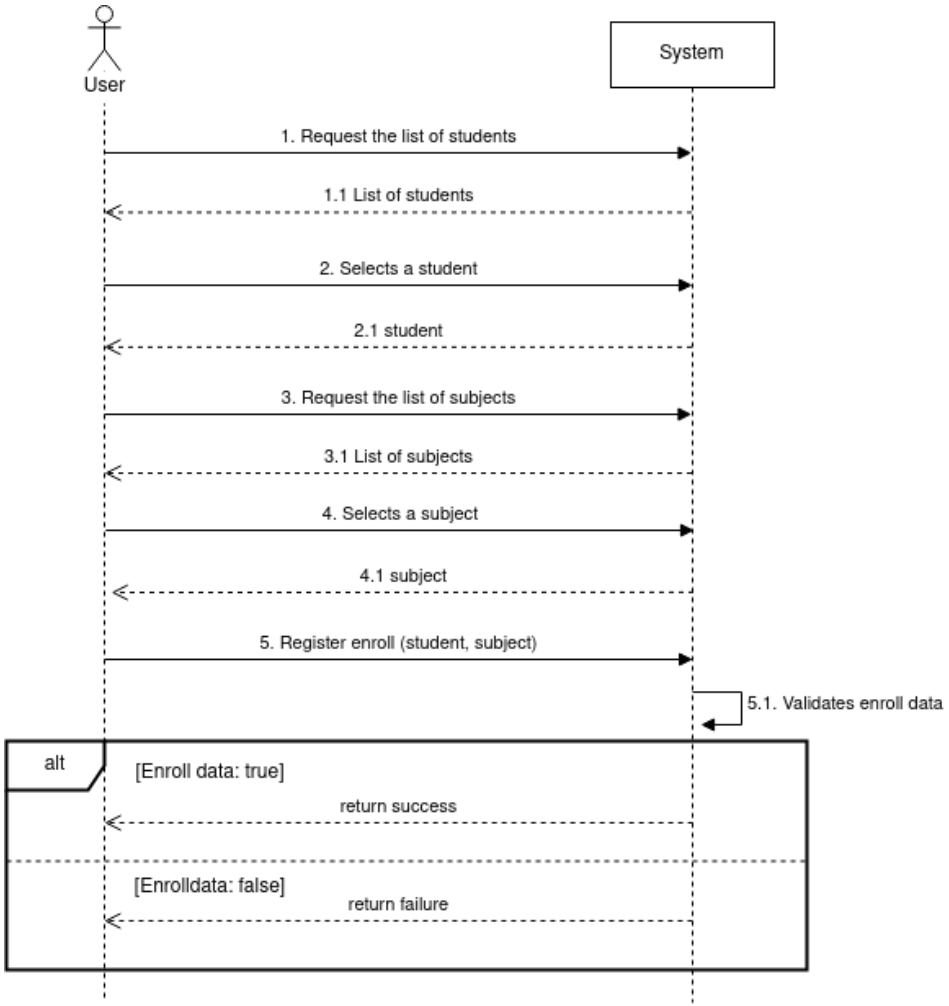
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



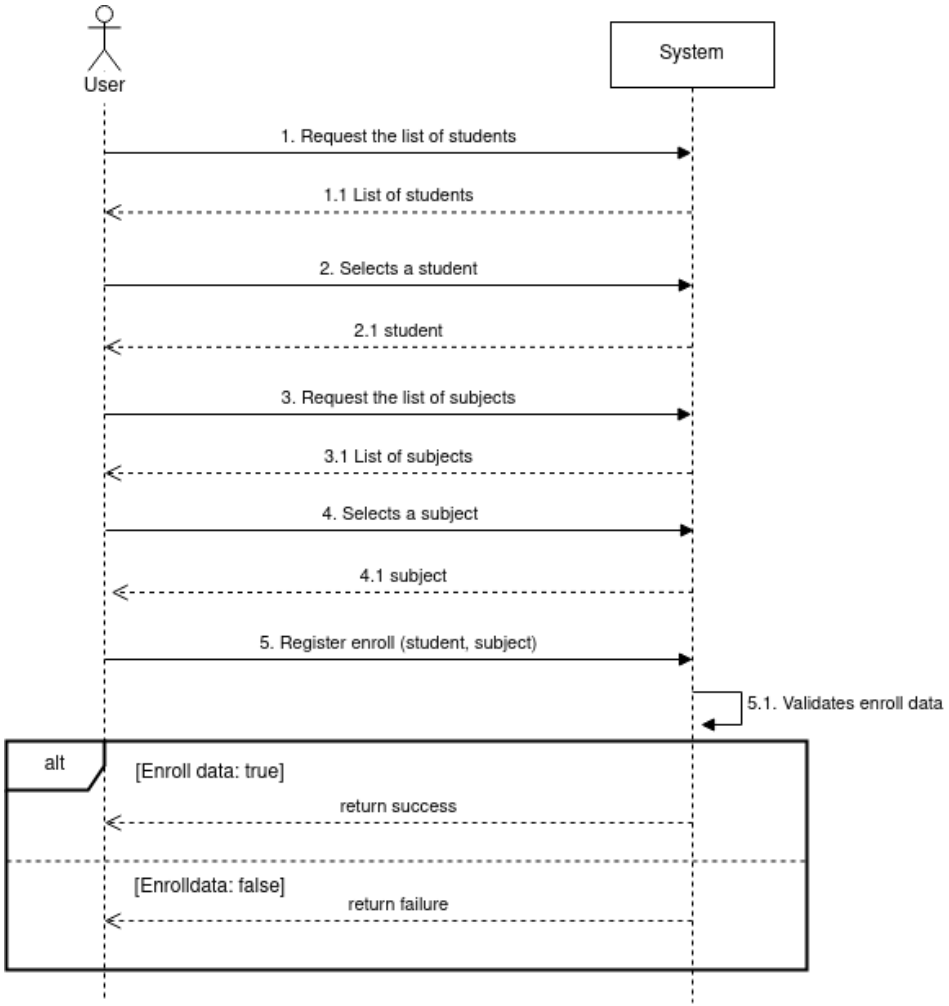
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



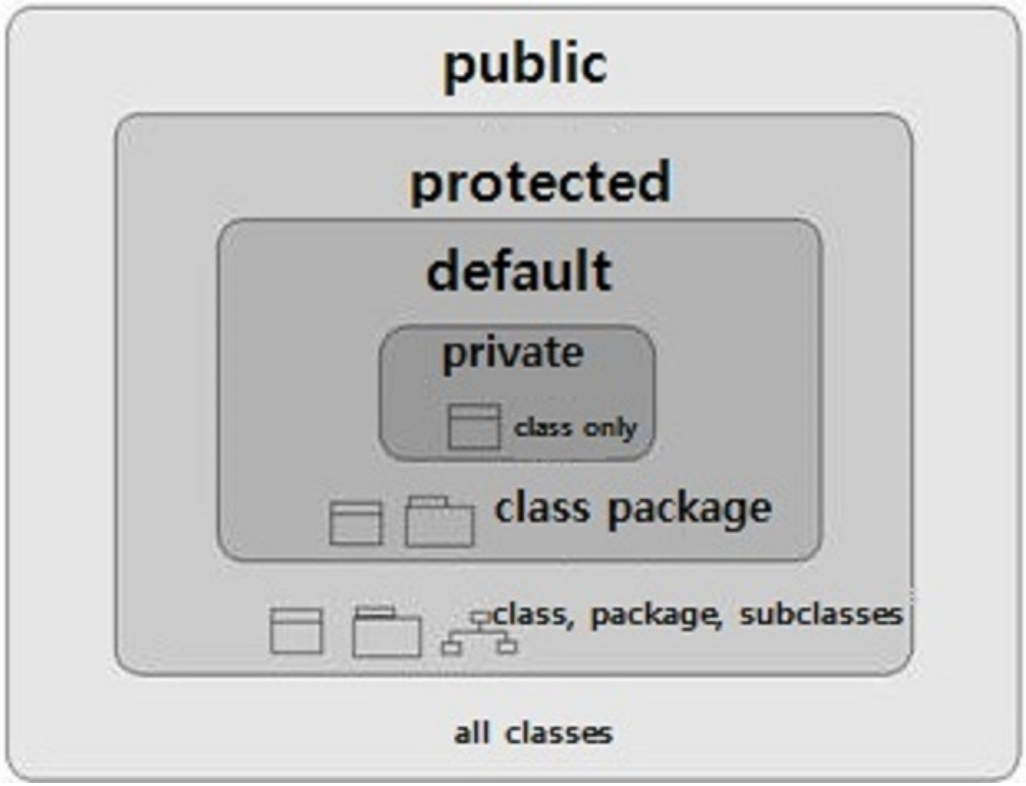
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



Access modifier (I)

- Access modifier is a general term which **may refer to a class, member variable, method or constructor**:
- Access modifiers **are applied to features to specify which other classes can access them**.



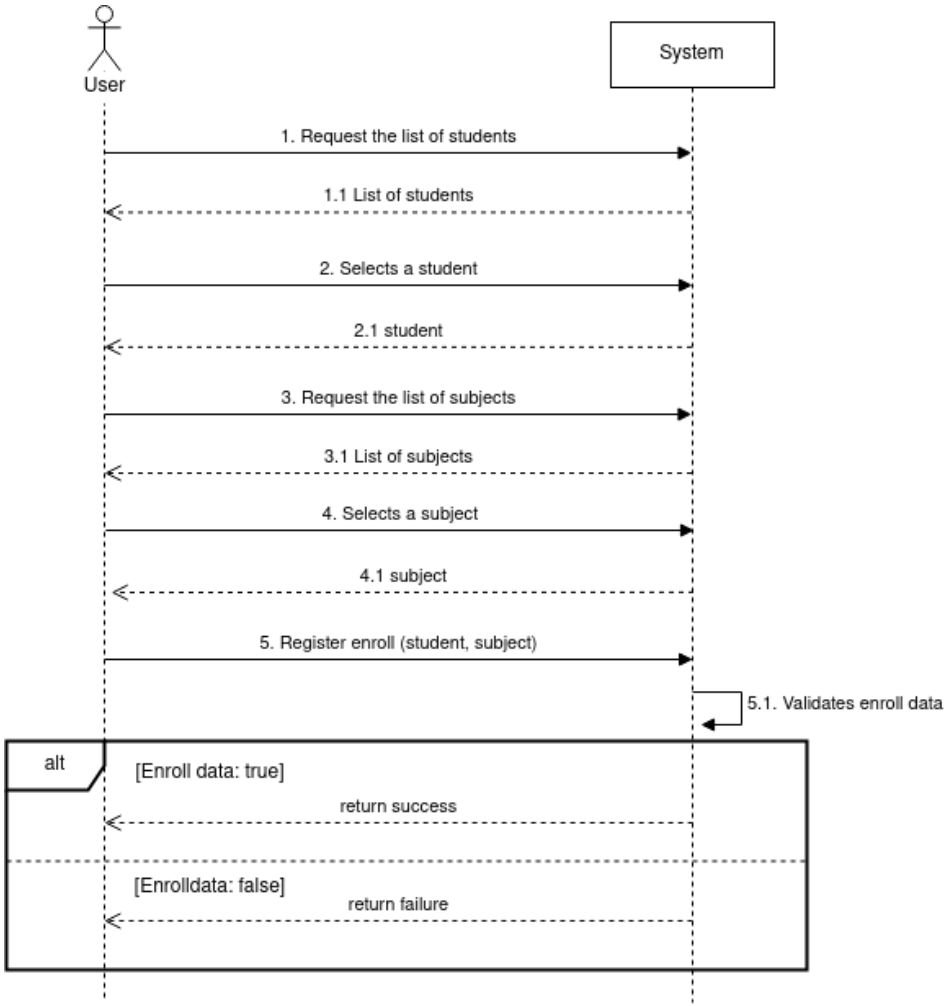
Access modifier (II)

- `public`
  - The least restrictive access modifier is `public`.
    - A feature (i.e. class, variable or method) that has been marked as `public` will be accessible by any other class.
- `private`
  - The most restrictive access modifier is `private`.
    - A feature that has been marked as `private` will only be accessible by an instance of the class it is has been defined in.
- `protected`
  - The `protected` access modifier is more restrictive than `public` but less restrictive than `private`.
    - A feature that has been marked as `protected` will only be accessible by subclasses of the class or a class in the same package.
- `Default`
  - A feature that has not been explicitly specified with any access modifier, then it will be accessible from classes in the same package as itself.

Naming Conventions & Comments

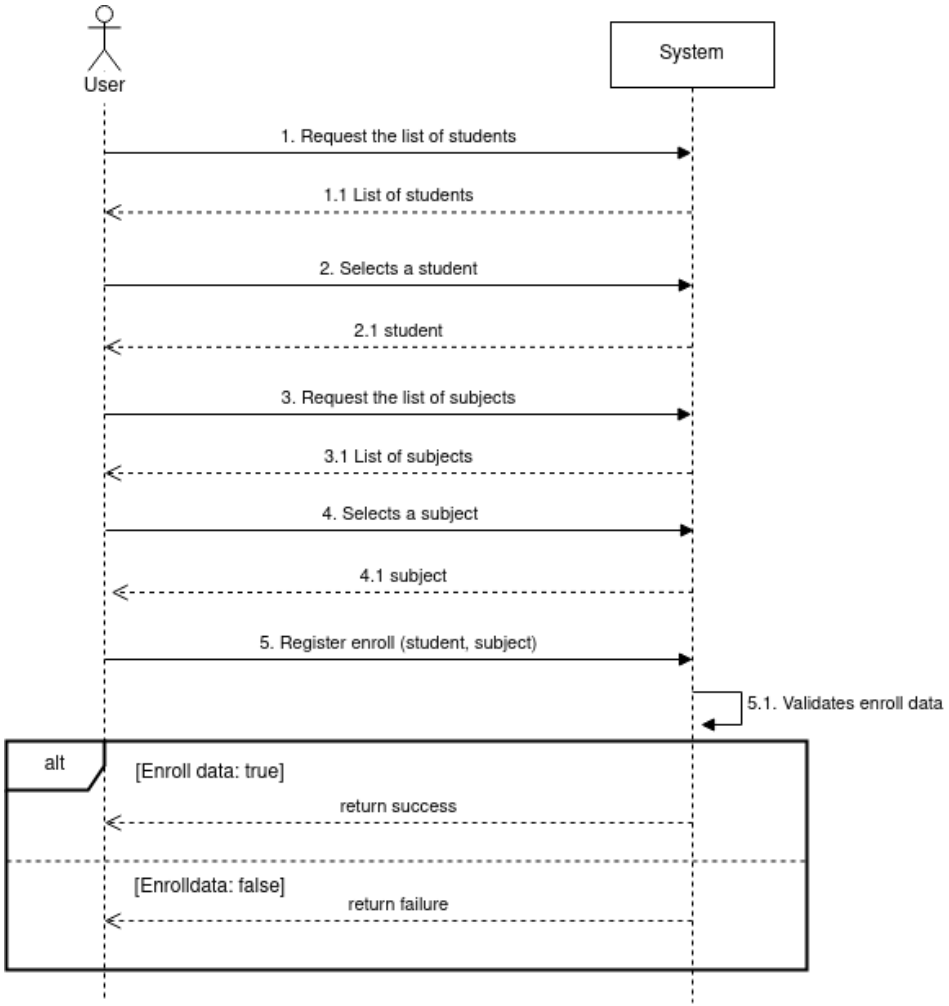
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



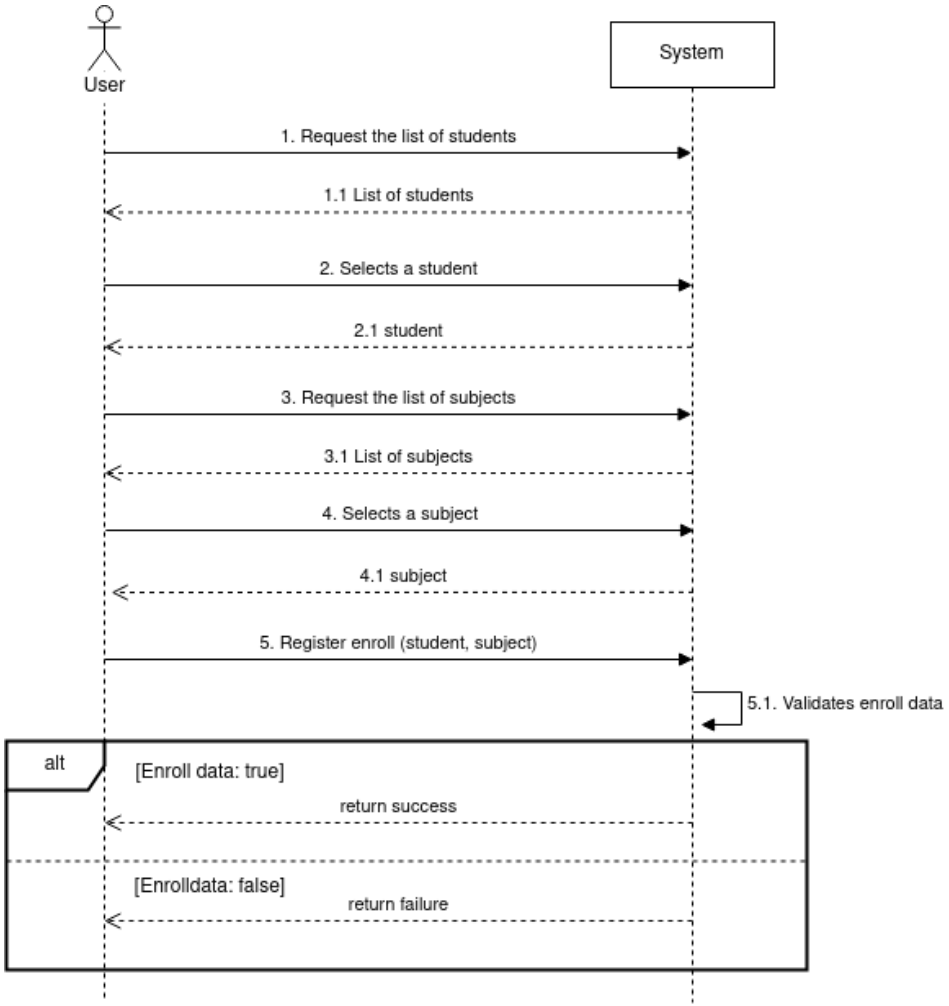
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



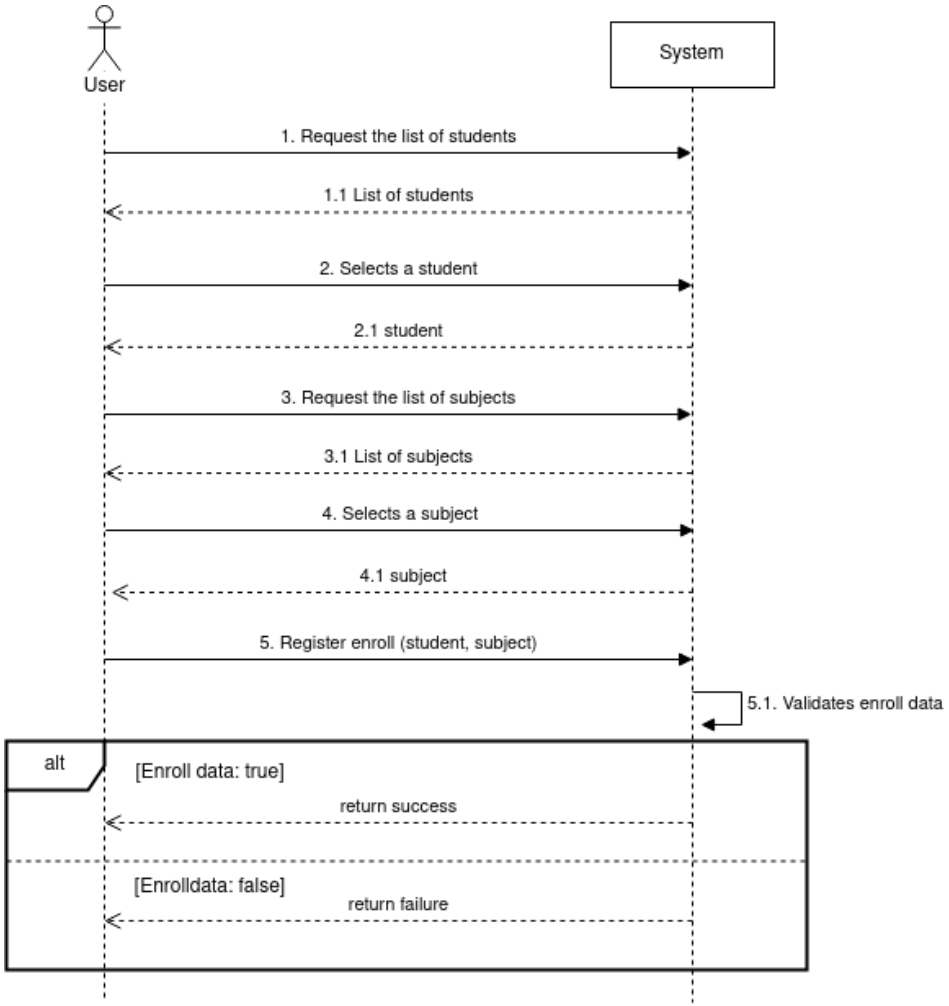
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



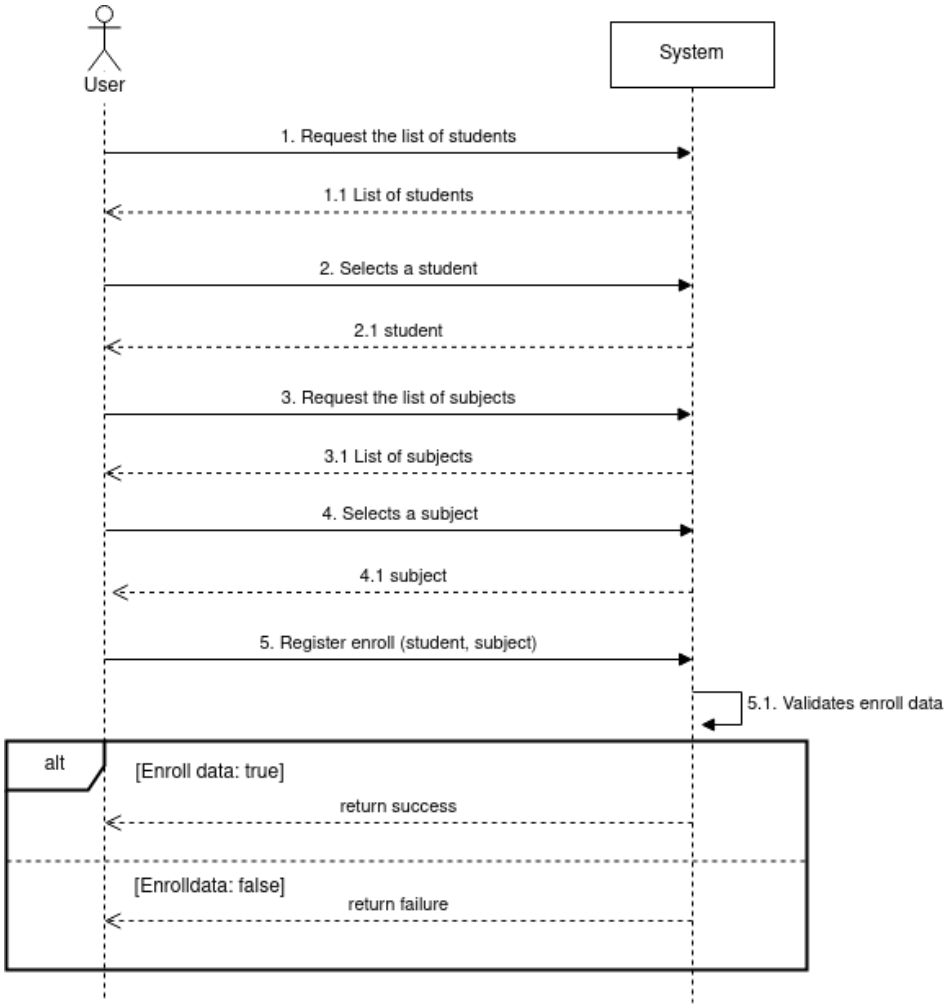
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



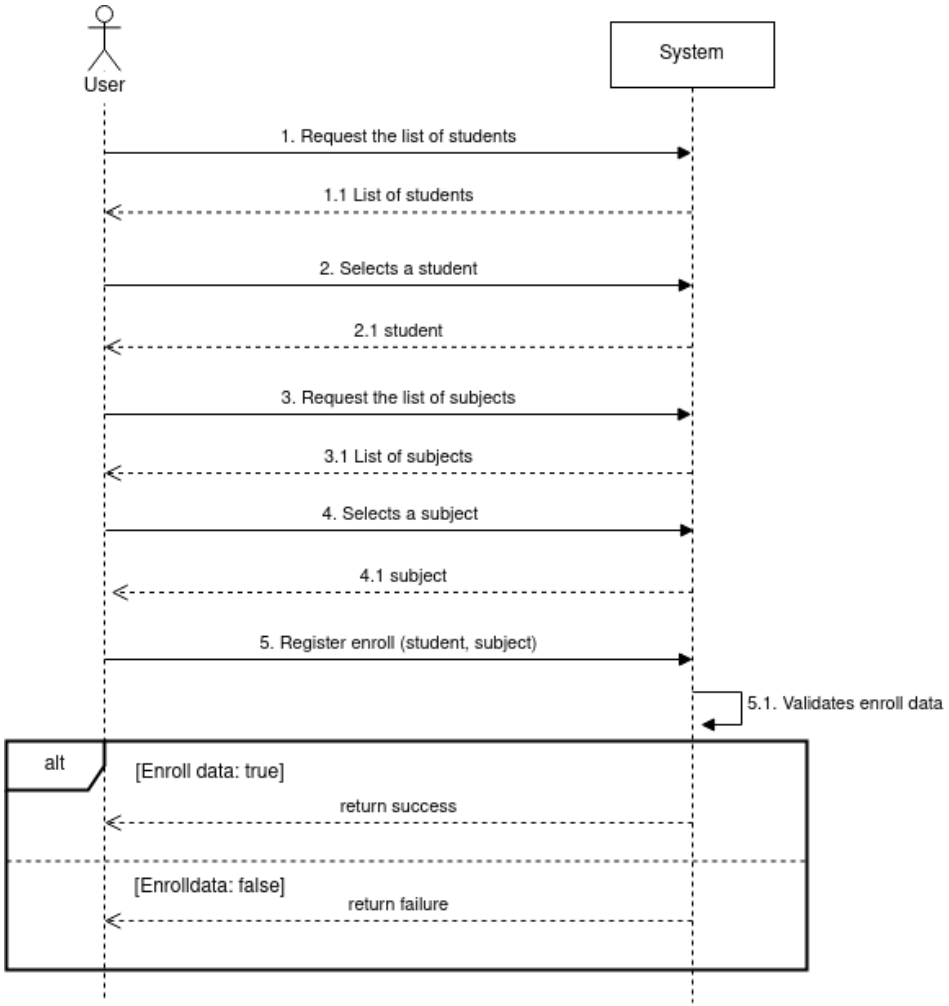
A Closer Look at Methods

Overloading Methods (I)

- **Overloading**
  - When **two or more methods within the same class that share the same name**, as long as their **parameter declarations are different**.
  - Overloaded methods may have different **return types**
    - The return type alone **is insufficient to distinguish two versions of a method**.
- When an **overloaded method is invoked**:
  - It is used the **type** and/or **number of arguments** as its guide to determine which version of the overloaded method to actually call.
    - The **return type is not considered**.

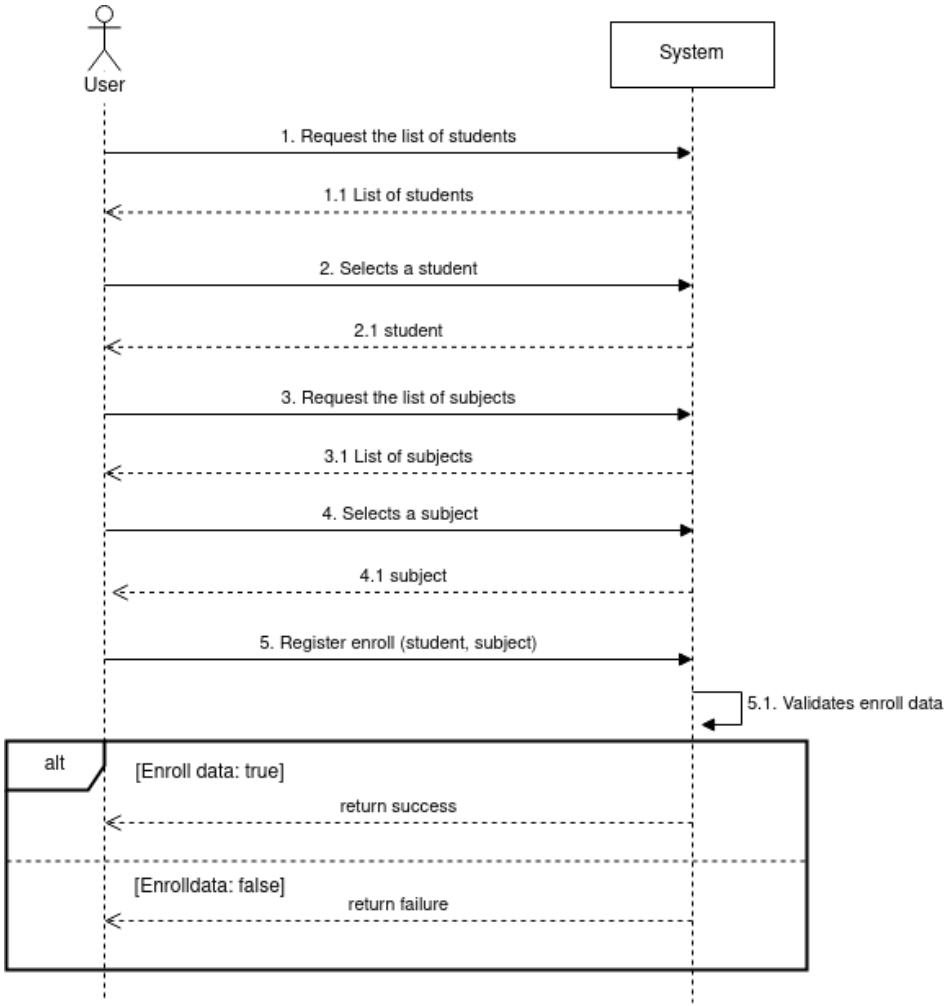
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



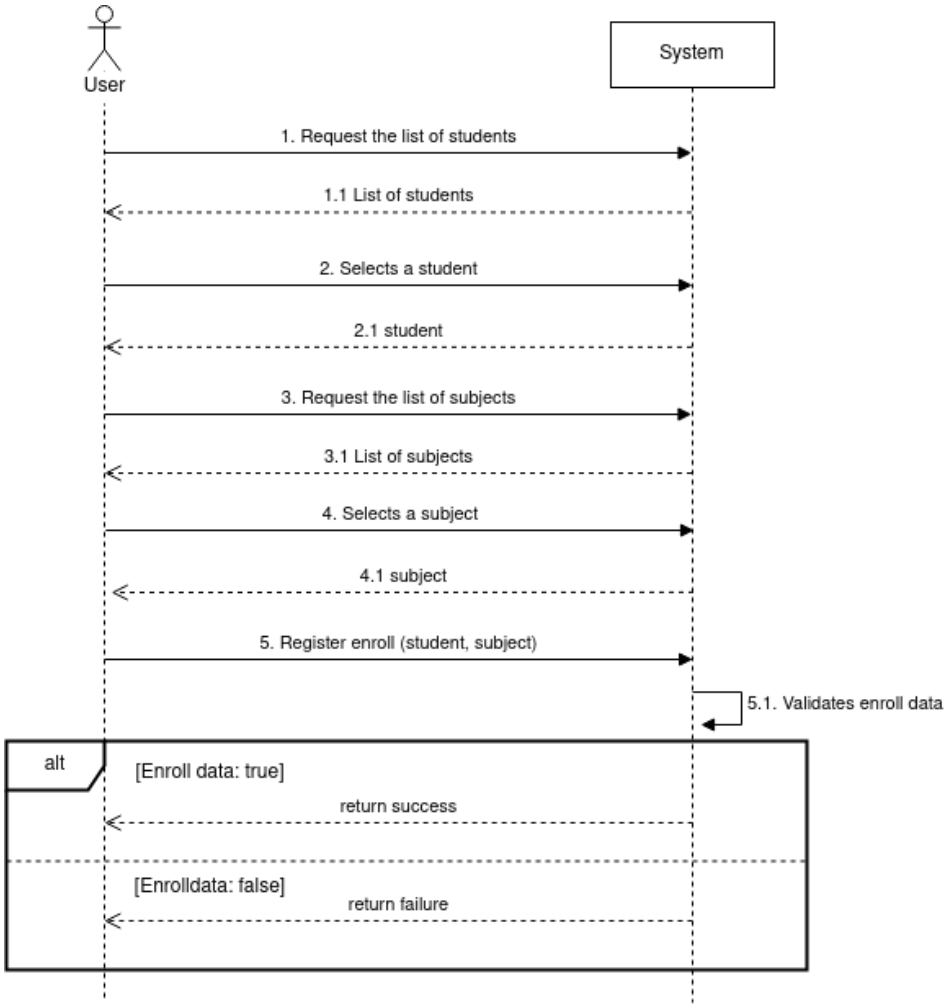
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



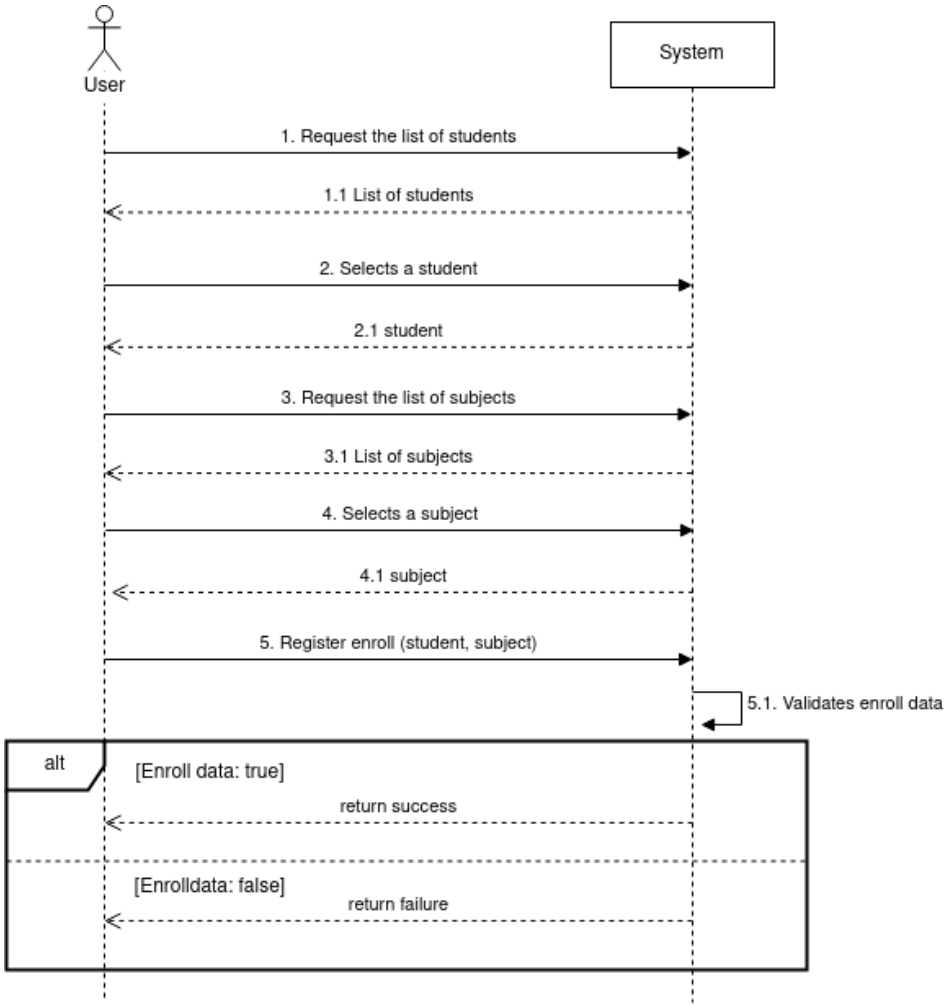
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



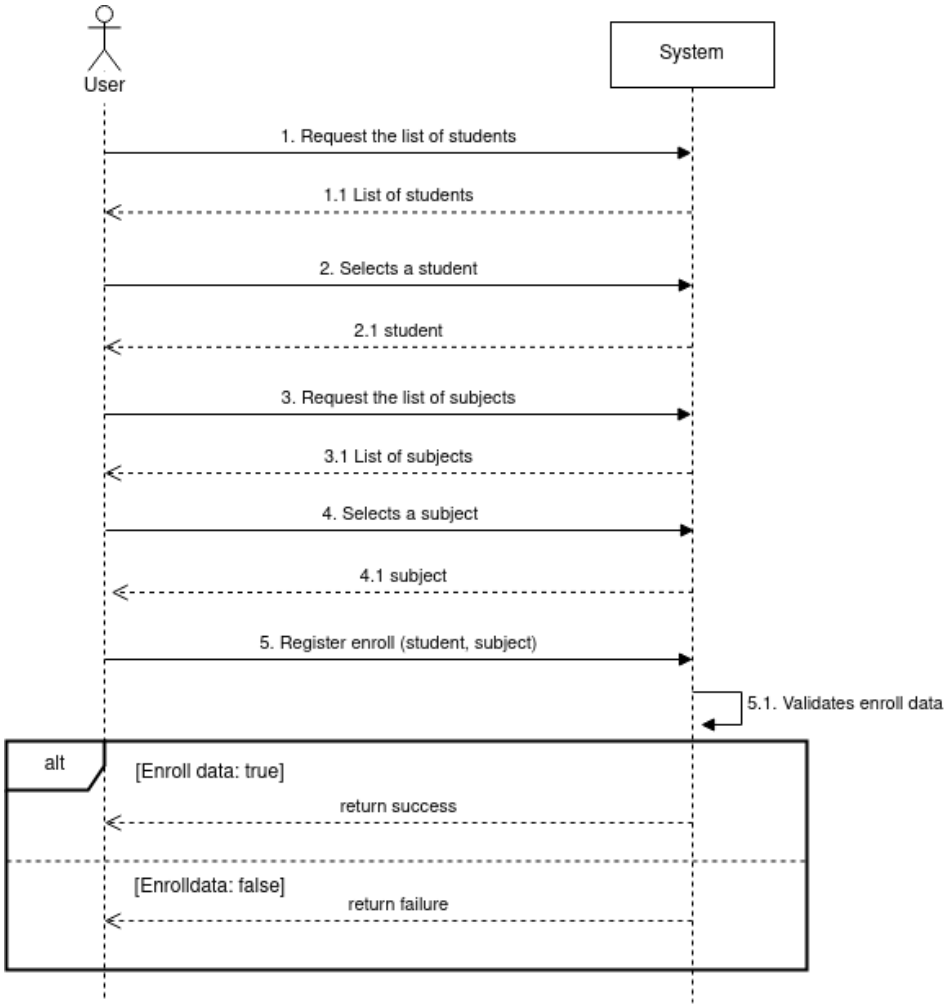
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



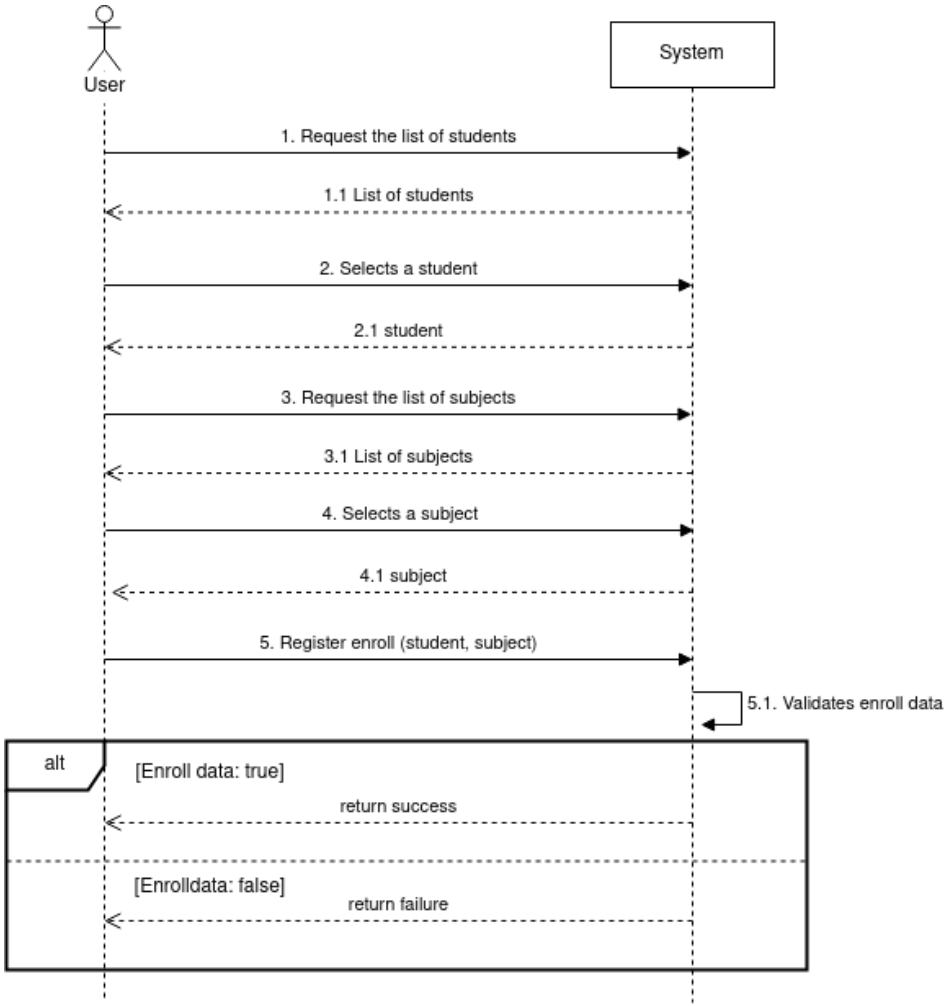
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



Analysis: Functional Requirements (V)

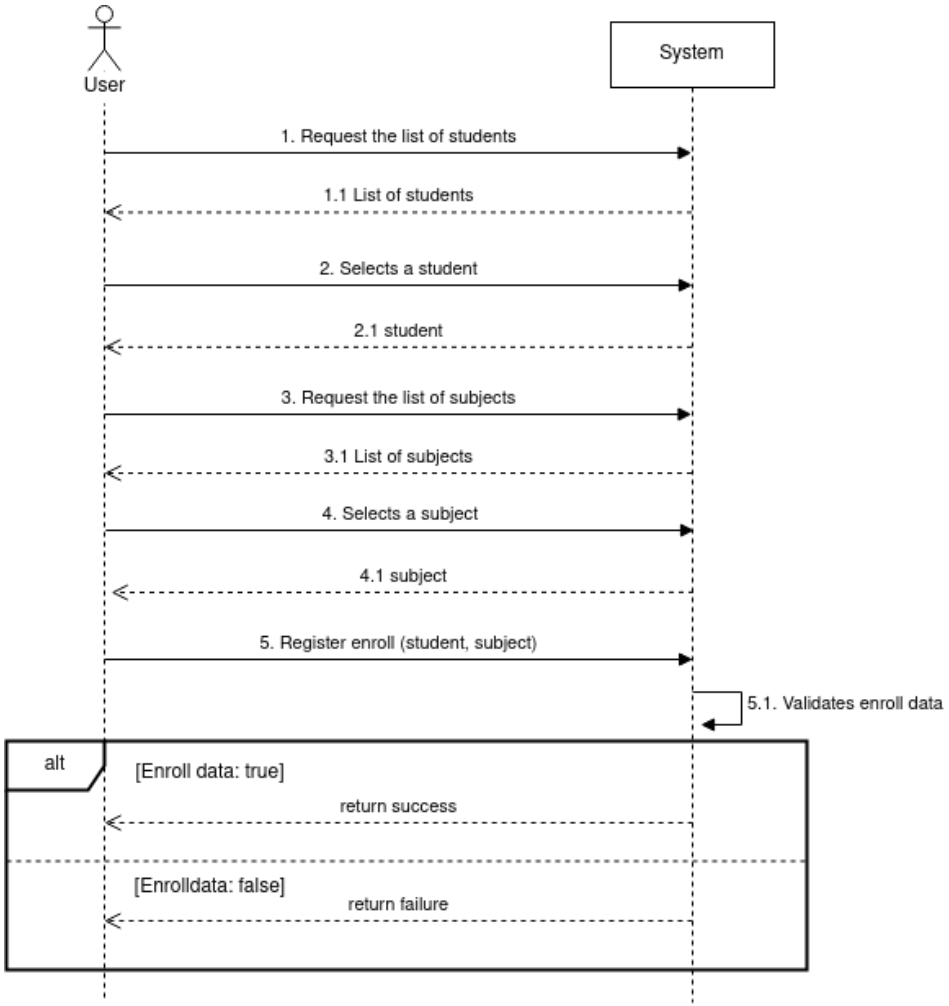
- SSD: UC 10 - Create Enroll



Inheritance & Polymorphism

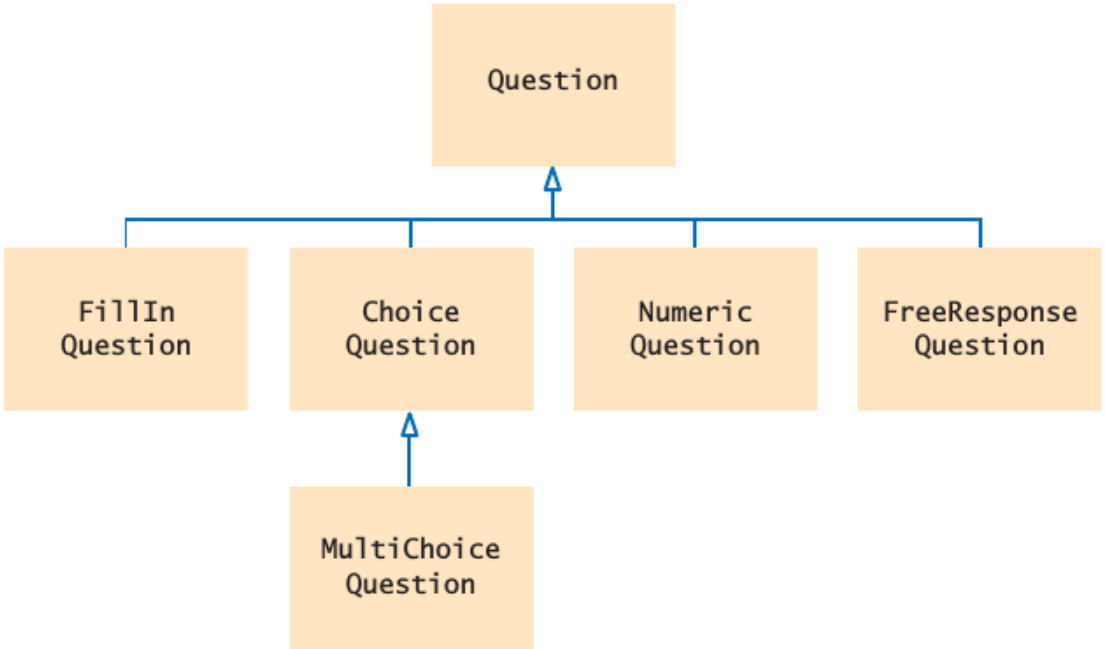
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



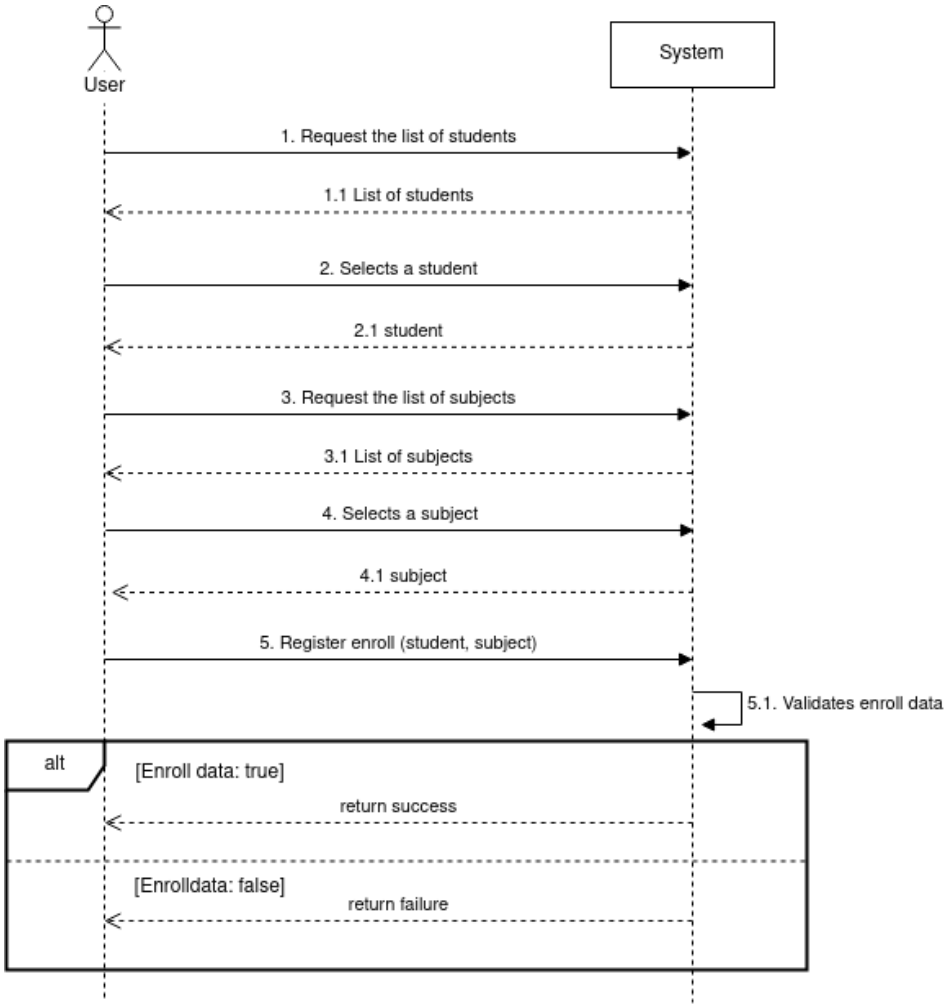
Deriving a class

- In object-oriented design, **inheritance** is a relationship between a more general class (called the **superclass**) and a more specialized class (called the **subclass**).
- The subclass inherits data and behavior from the superclass.



Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



Calling the Superclass Constructors and Methods

- To call a superclass constructor or method, use the `super` reserved word.

Syntax

```
public ClassName(parameterType parameterName, . . .)
{
   super(arguments);
   . . .
}
```

The superclass constructor is called first.

The constructor body can contain additional statements.

```
public ChoiceQuestion(String questionText)
{
   super(questionText);
   choices = new ArrayList<String>;
}
```

If you omit the superclass constructor call, the superclass constructor with no arguments is invoked.

Syntax

```
super.methodName(parameters);
```

Calls the method of the superclass instead of the method of the current class.

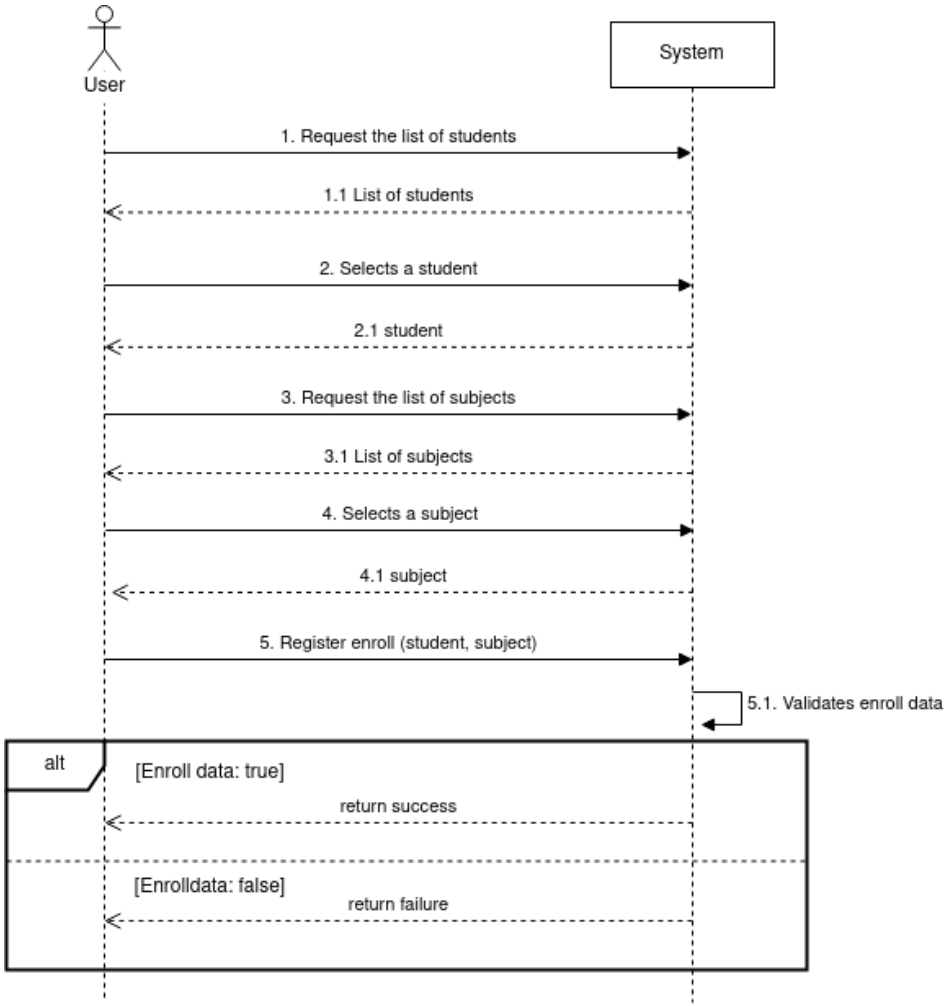
```
public void deposit(double amount)
{
   transactionCount++;
   super.deposit(amount);
}
```

If you omit super, this method calls itself.

See page 437.

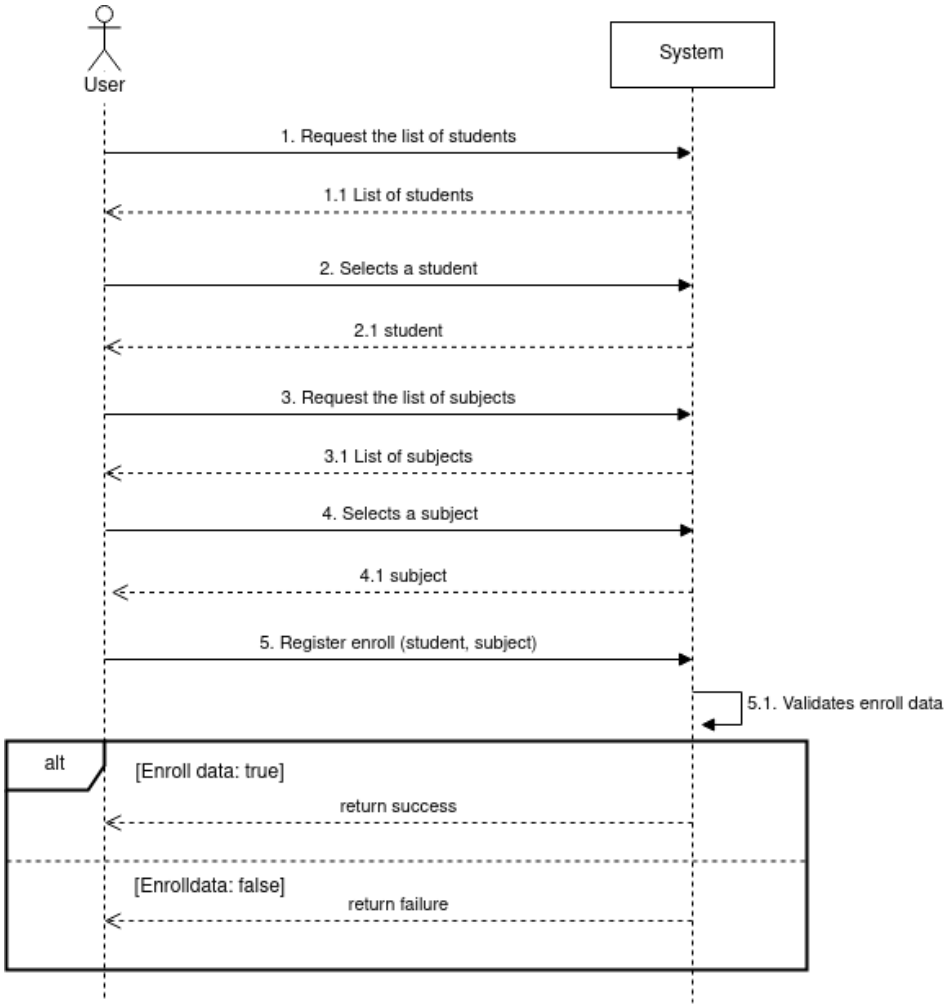
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



Analysis: Functional Requirements (V)

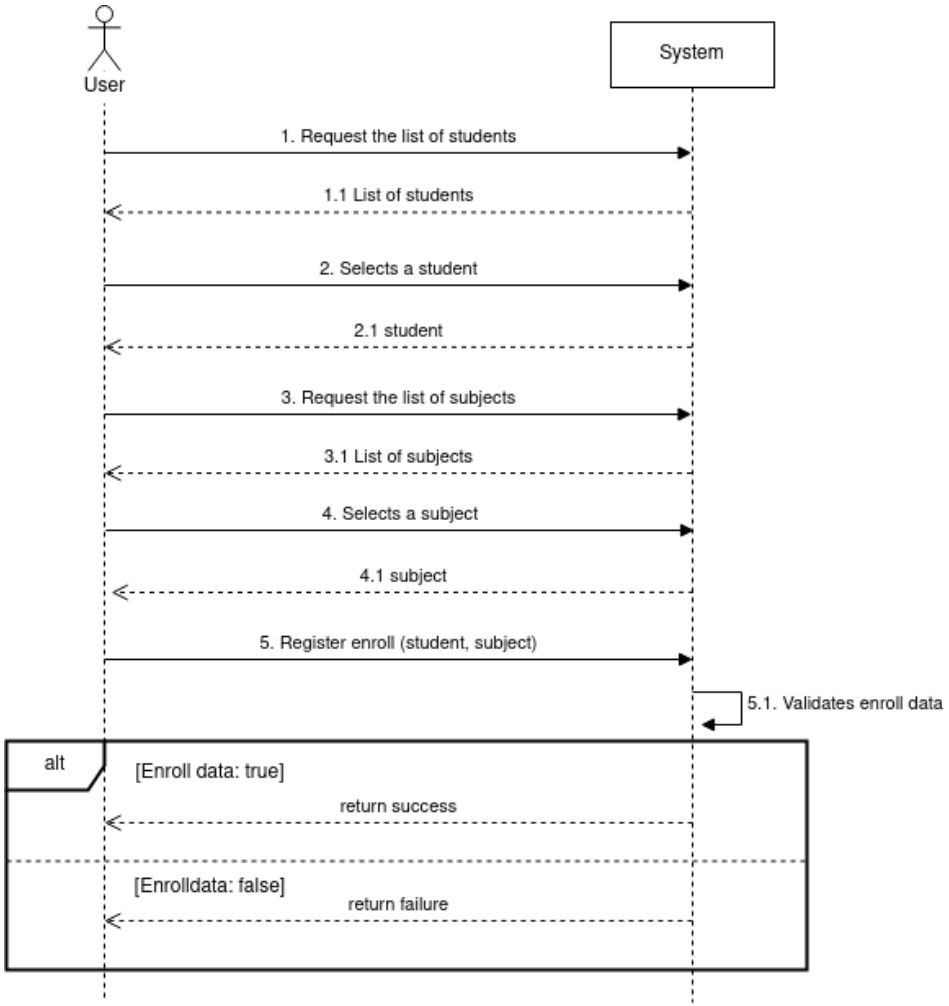
- SSD: UC 10 - Create Enroll



Handling Exceptions

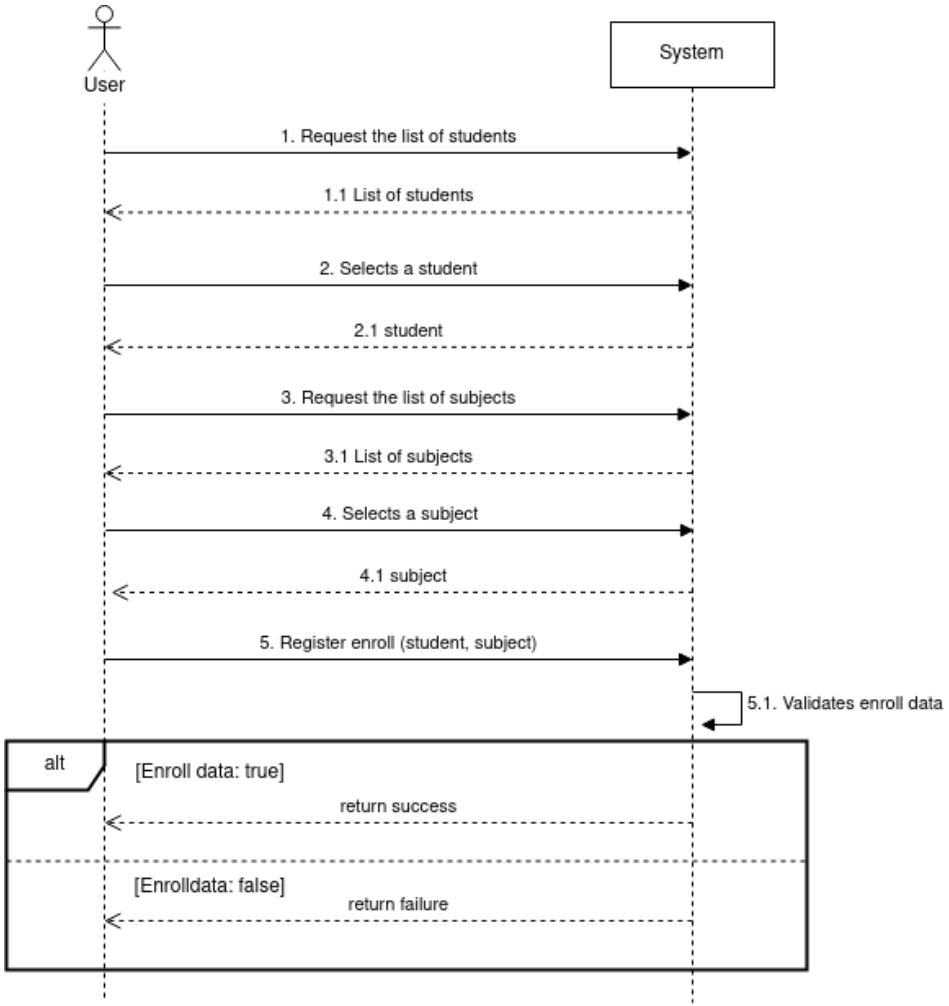
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



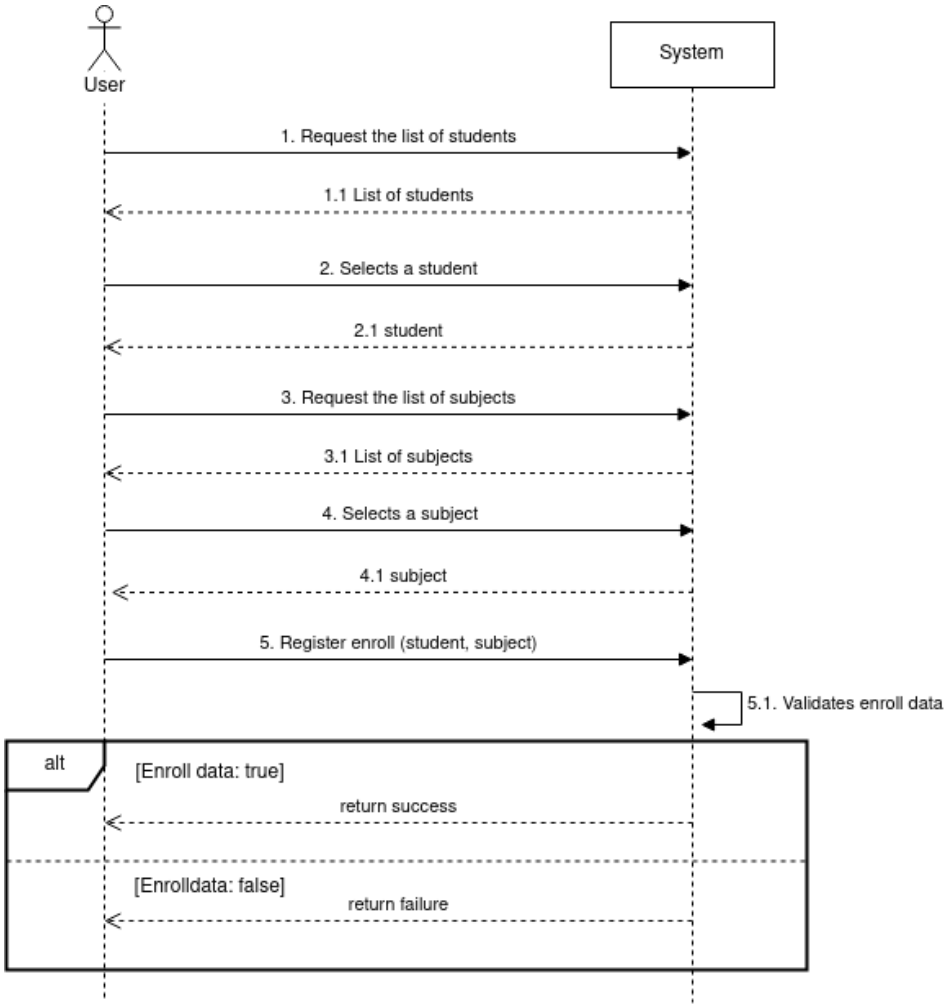
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



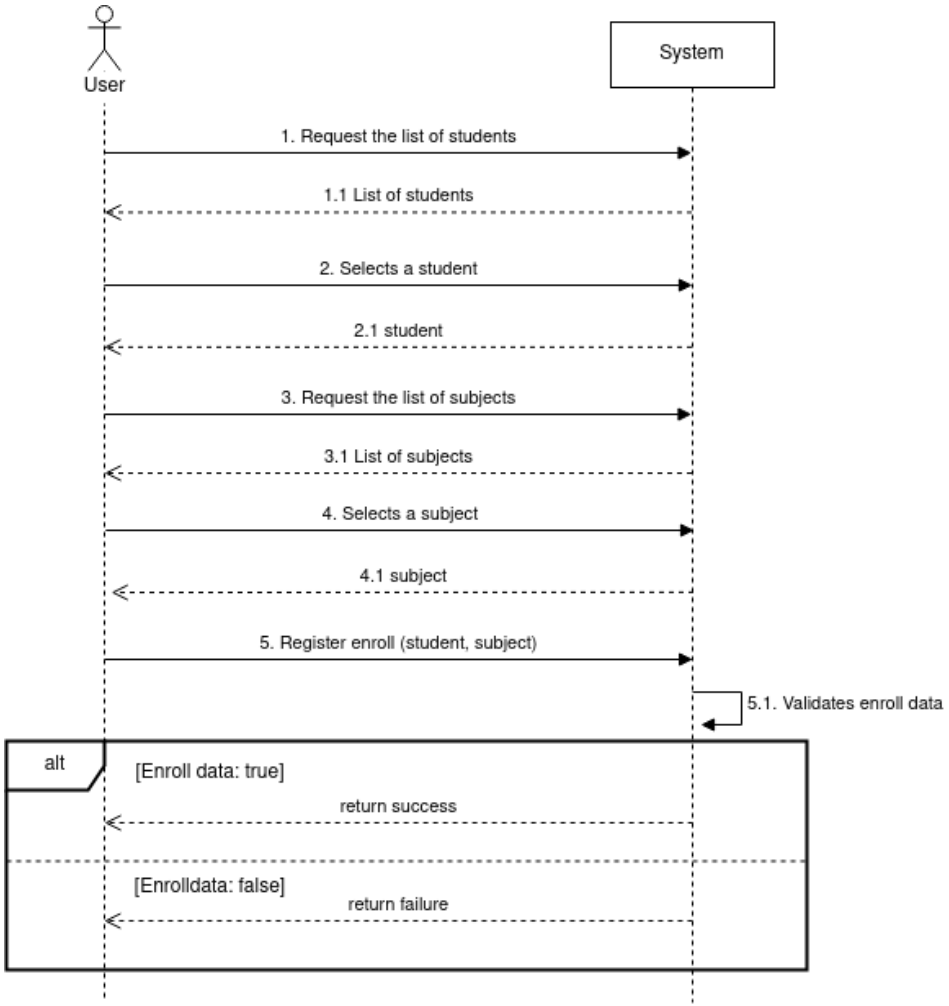
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



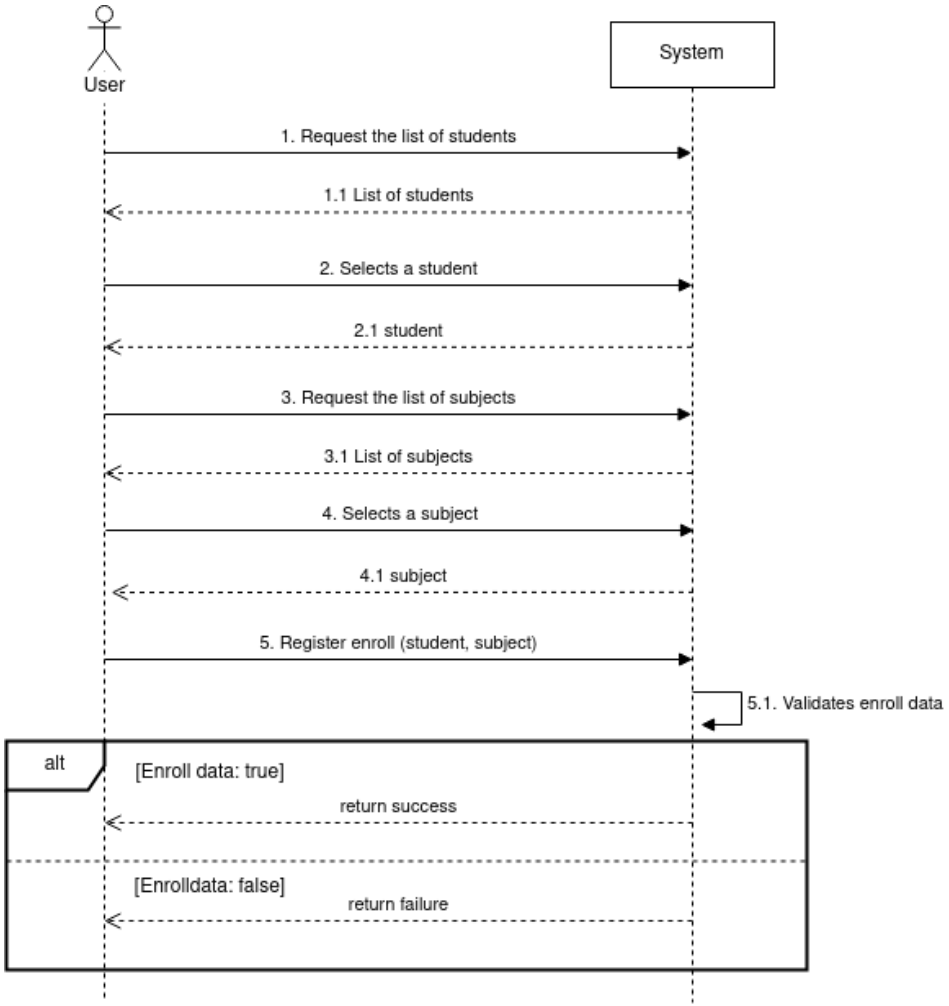
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



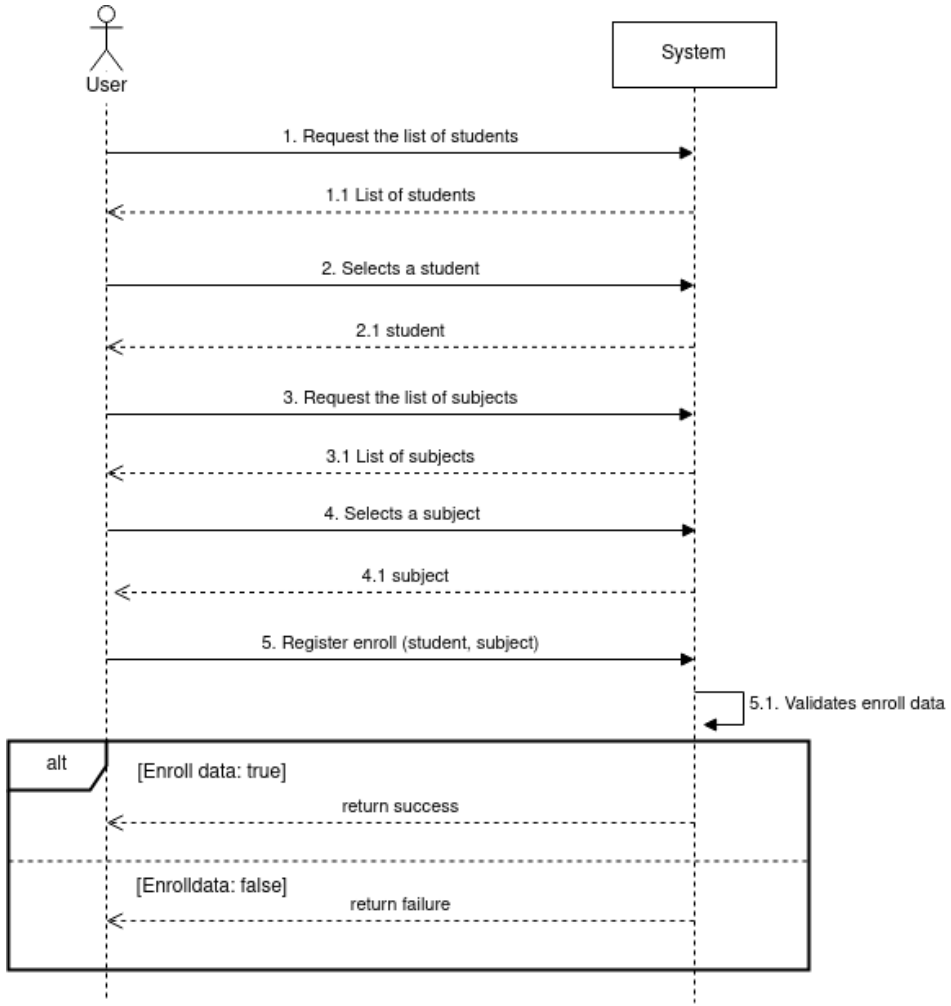
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



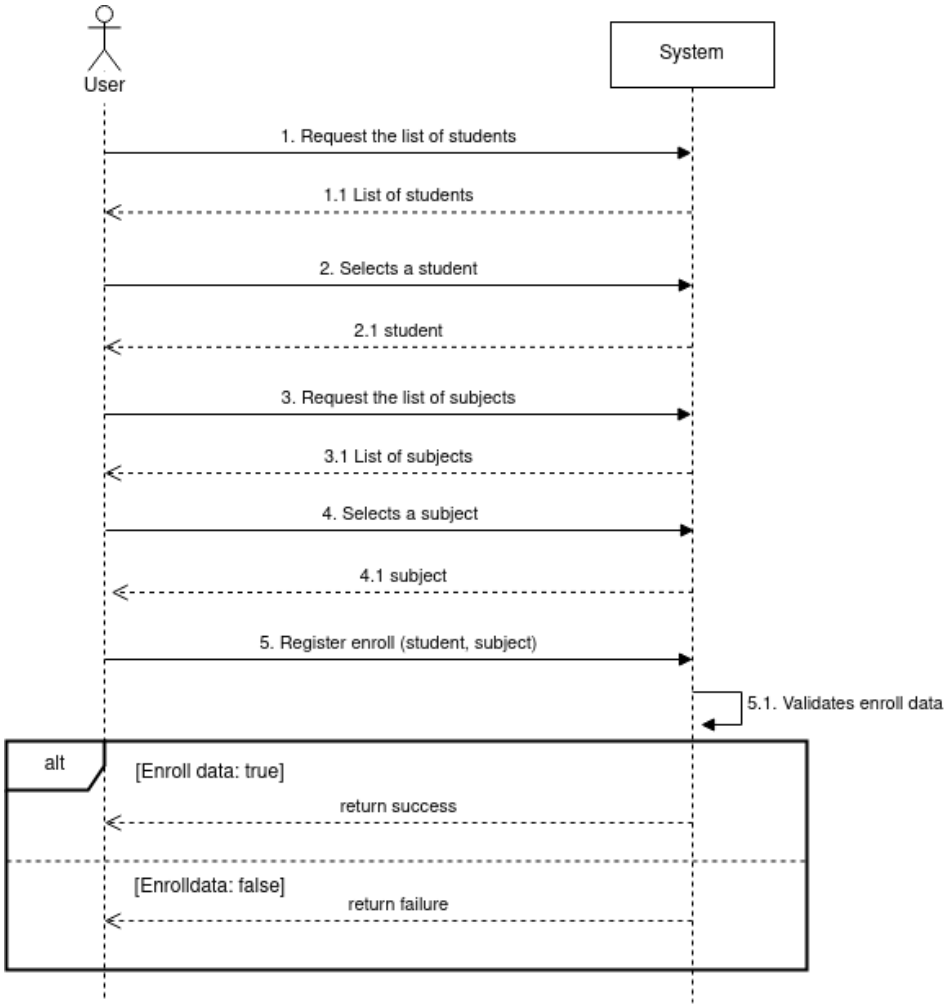
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



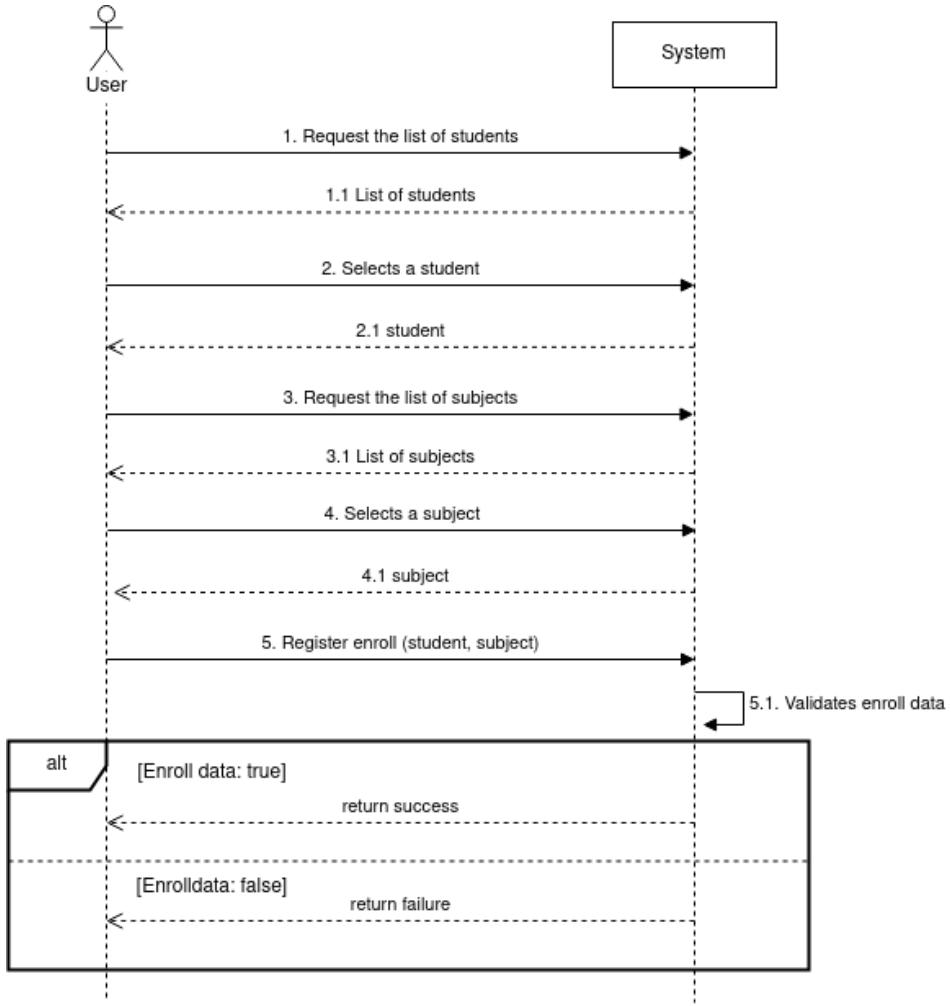
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



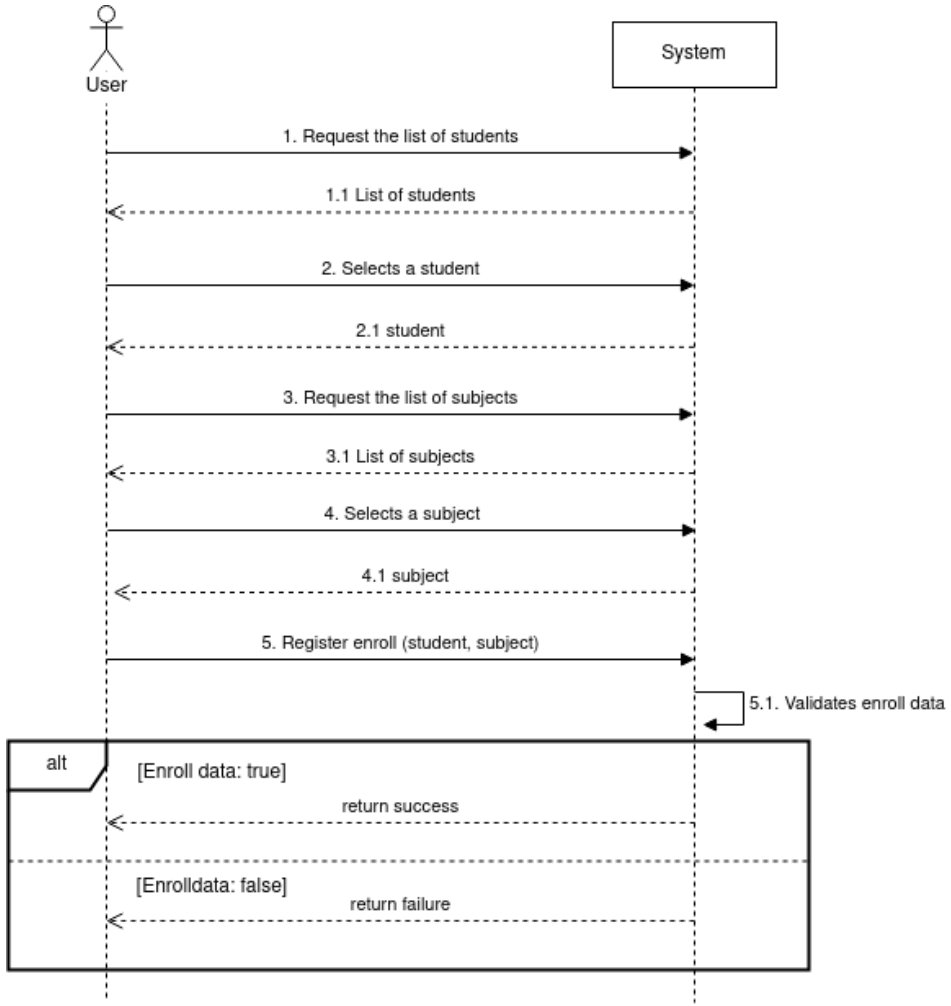
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



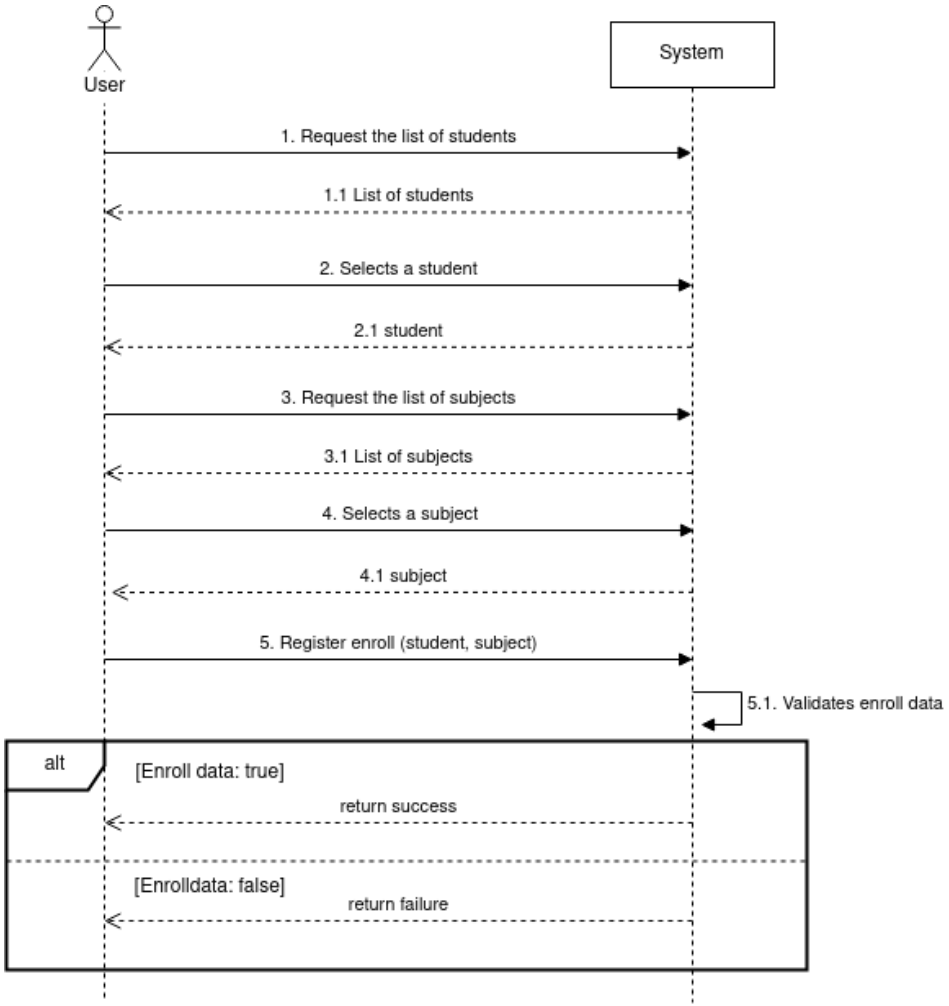
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



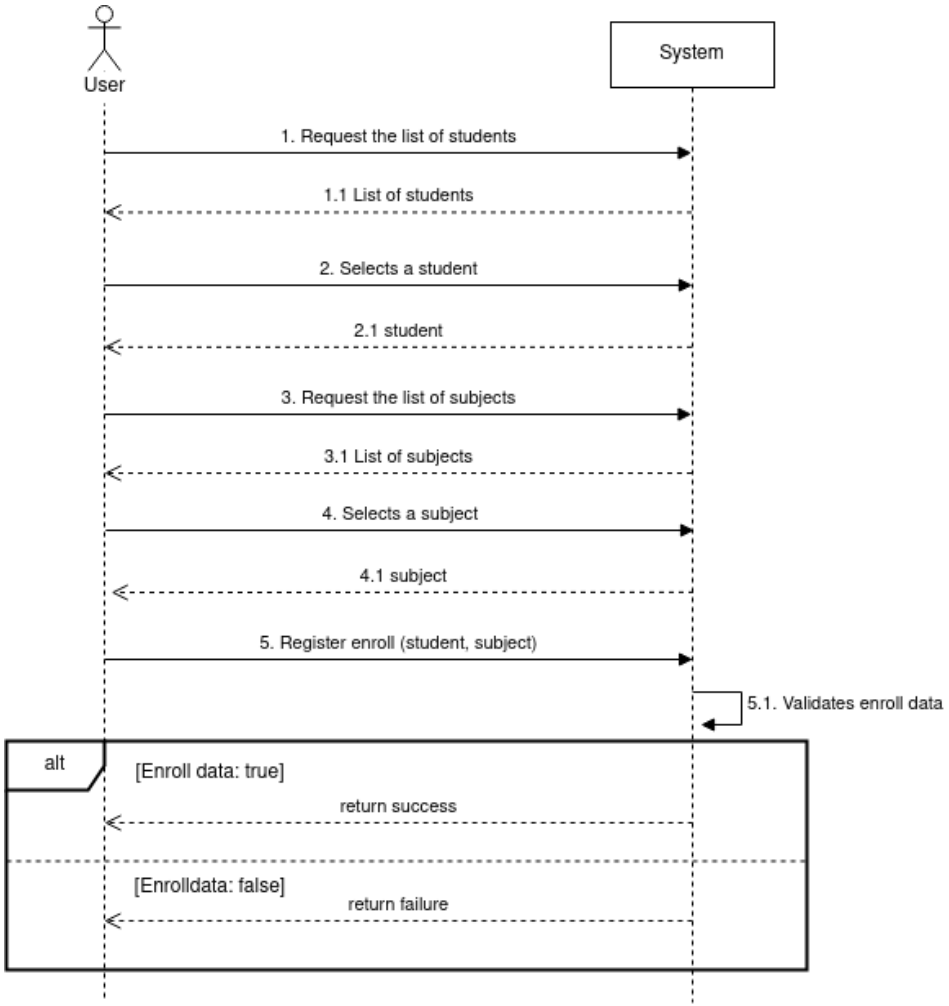
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



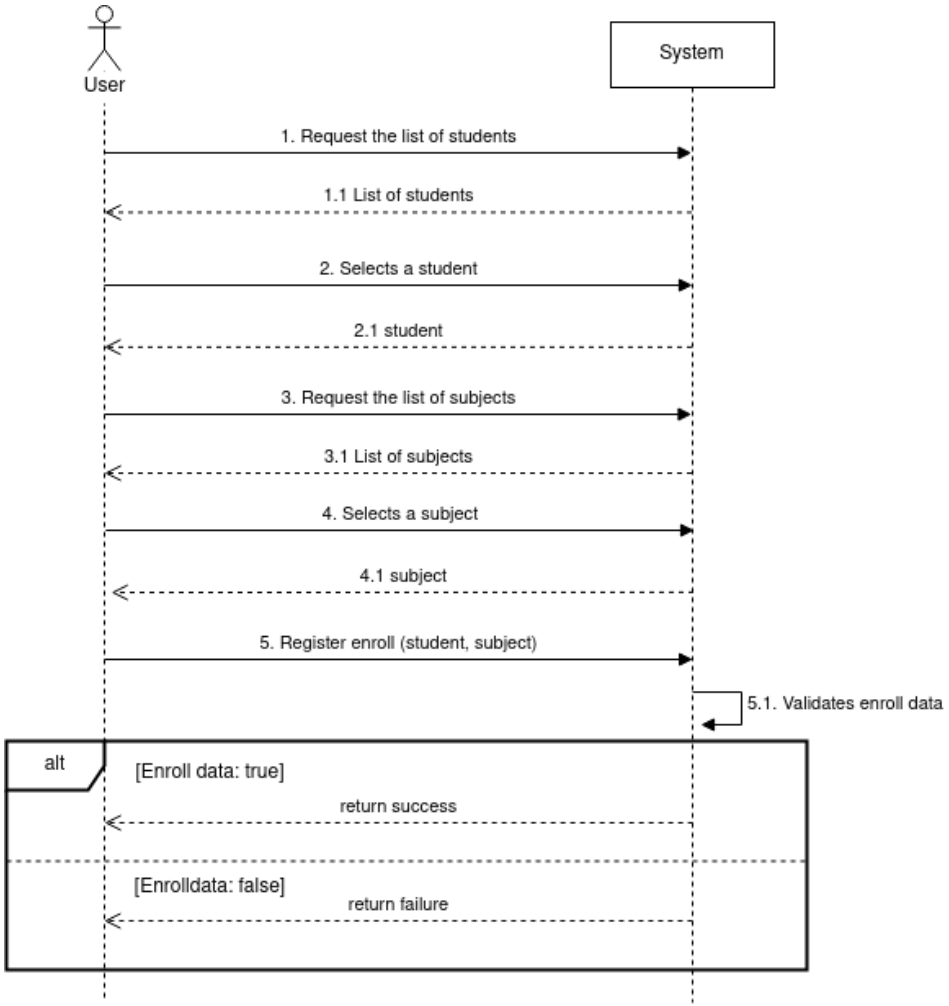
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



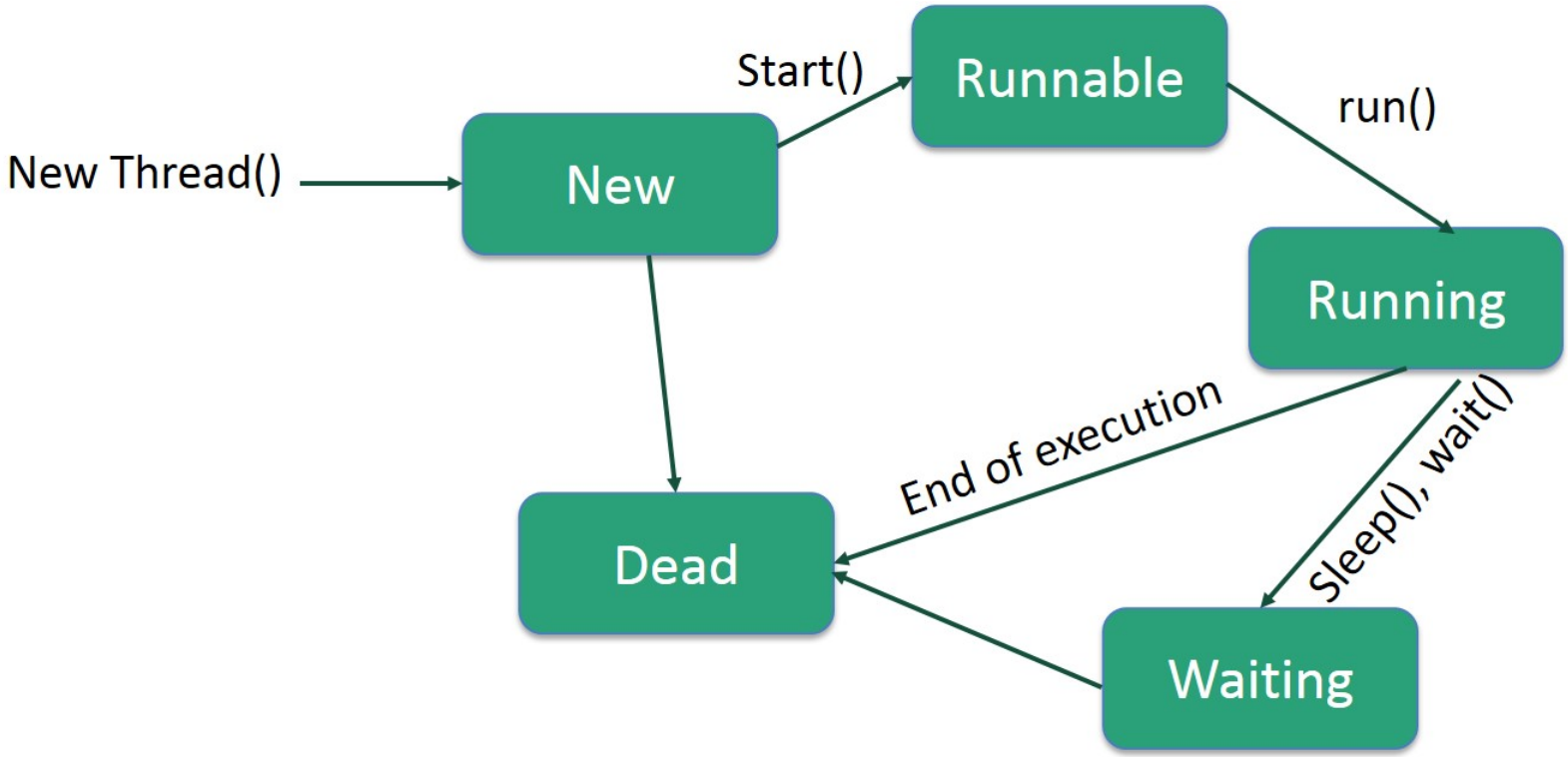
Threads

Multitasking & Multithread

- Java is a **multi-threaded** programming language which means we can develop multi-threaded program using Java.
- A multi-threaded program **contains two or more parts that can run concurrently** and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.
- **Multitasking** is when multiple processes share common processing resources such as a CPU.
- Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads.
- Each of the threads can run in parallel.
  - The OS divides CPU processing time not only among different threads of each application.
- Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Life Cycle of a Thread

- A thread goes through various stages in its life cycle.
  - For example, a thread is born, started, runs, and then dies.



Check: `TP3_09.zip`

Check: `TP3_10.zip`

Example: School Application

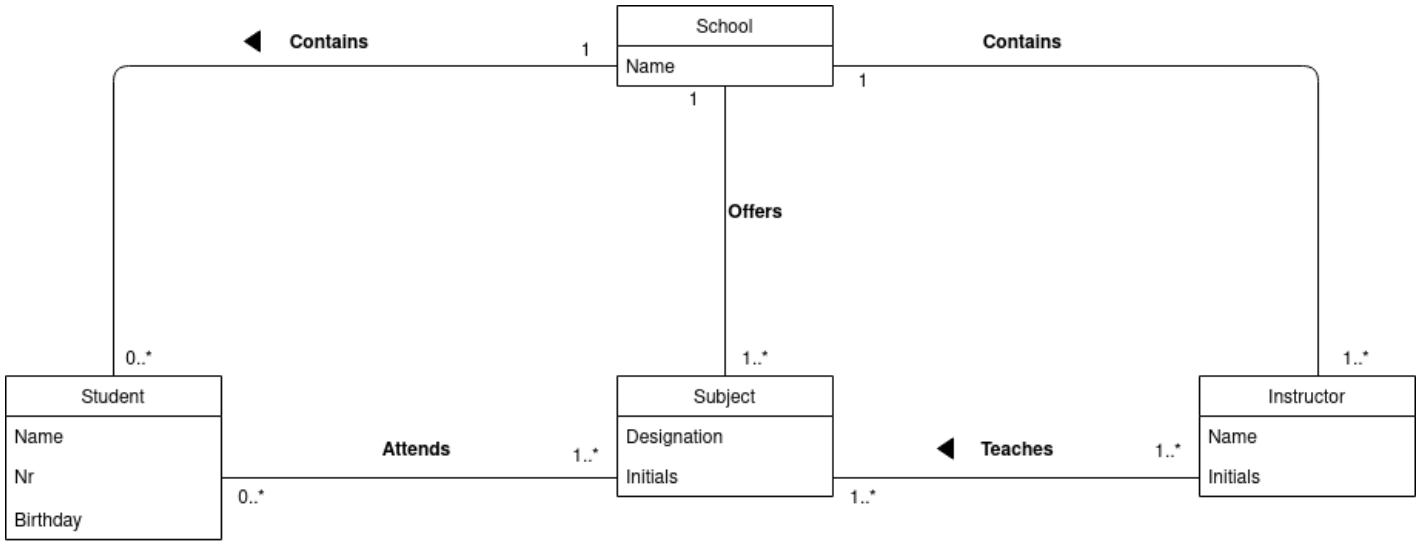
Context/Problem

- Software Application for a school.
- Features
  - School has students, instructors and offers subjects.
  - Instructors can teach many subjects.
  - The same subject can be taught by different instructors.
  - Student can attend to many subjects.
- Requirements
  - Data must be persistent.
  - User Interface must be console-based.
- User Stories
  - As a system user, I want to print (in paper) the student subject marks, so that I can issue the certificate.
  - As a school manager, I want to award the best student, so that I can motivate them.
  - As a school manager, I want to award the youngest student.

Check: `SchoolApplication.zip`

Analysis: Domain Model (I)

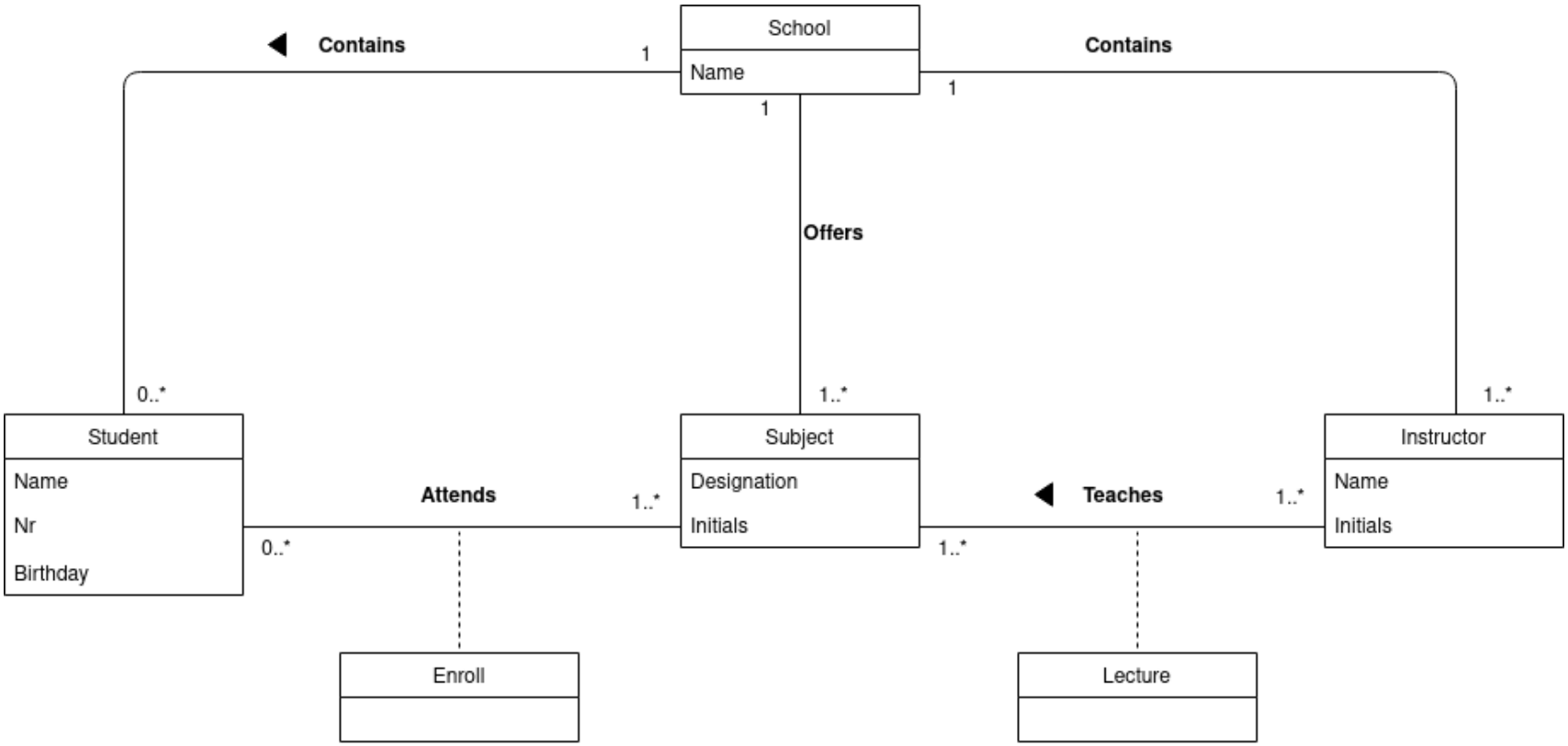
- **Entities** and **Relations**
  - `School` contains `Student`, `Instructor` entities and offers `Subject`.
  - `Instructor` entities can teach many `Subject` entities.
  - The same `Subject` entity can be taught by different `Instructor` entities.
  - `Student` entity can attend to many `Subject` entities.



- However, this model has **many-to-many** relations

Analysis: Domain Model (II)

- Breaking **many-to-many relation** with an associative class.
  - `Enroll` for breaking `Student` and `Subject`
  - `Lecture` for breaking `Instructor` and `Subject` relation



Analysis: Non-Functional Requirements (I)

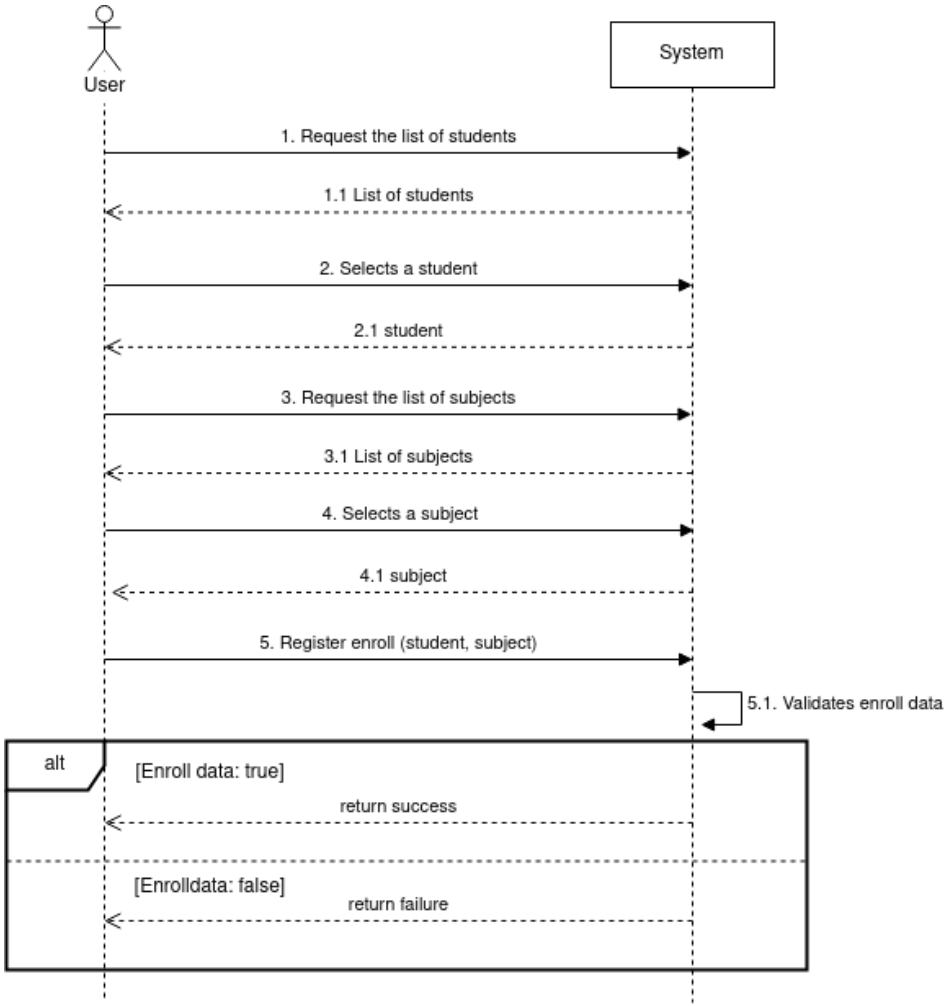
- **Usability**
  - Console-based user interface
- **Reliability**
  - No
- **Performance**
  - No
- **Supportability**
  - Huge amount of tests (unit tests)

Analysis: Non-Functional Requirements (II)

- **+**
  - **Design constraints**
    - Data persistent in binary files
  - **Implementation constraints**
    - Java language
  - **Interface constraints**
    - No

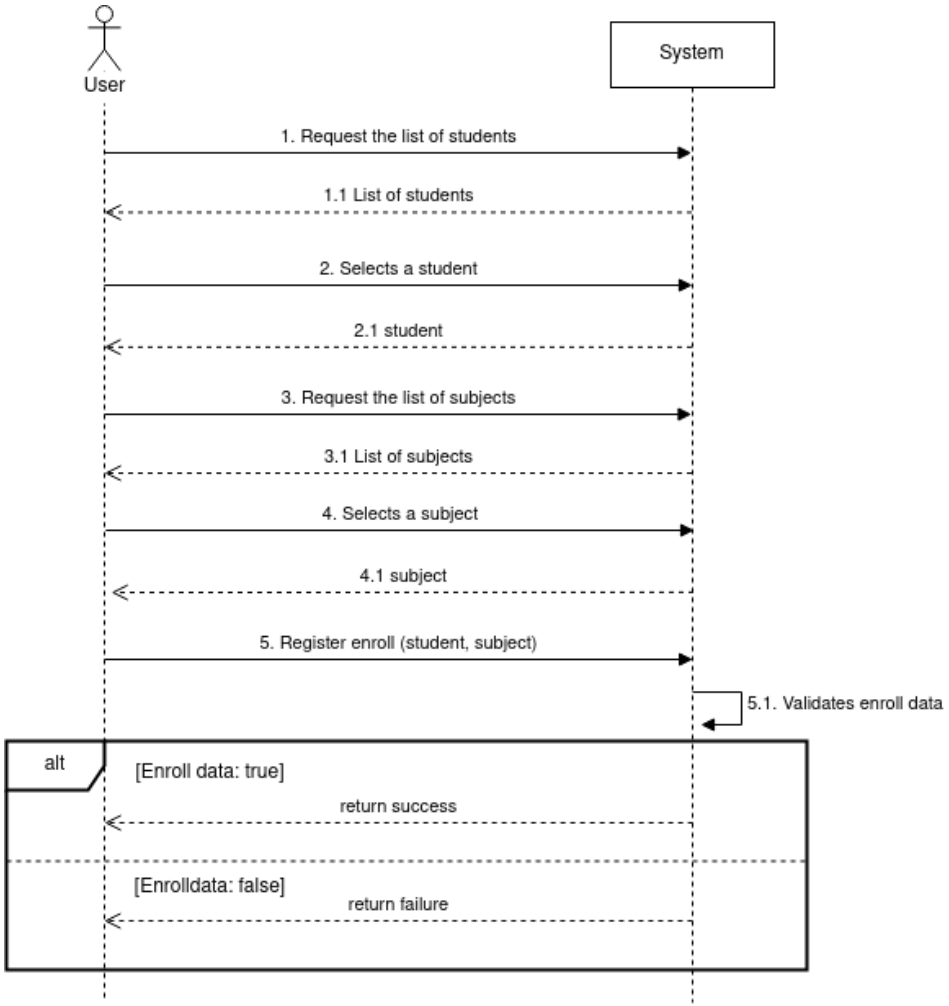
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



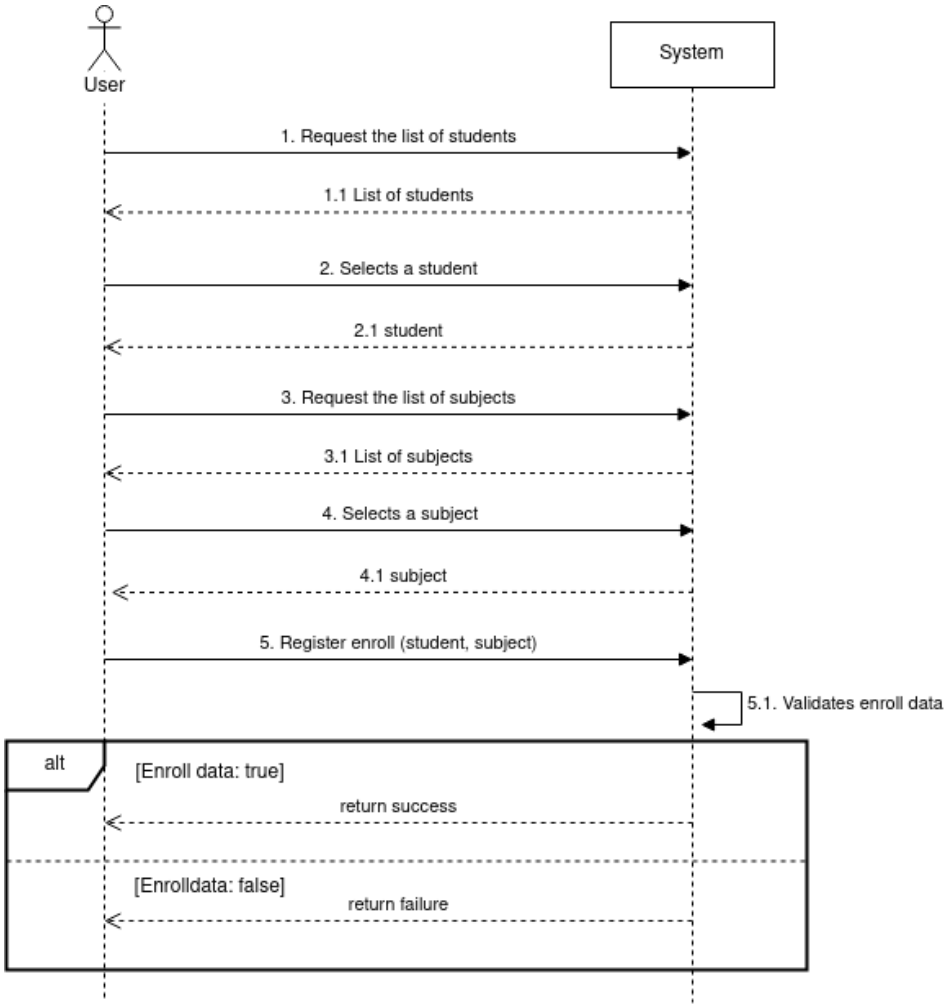
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



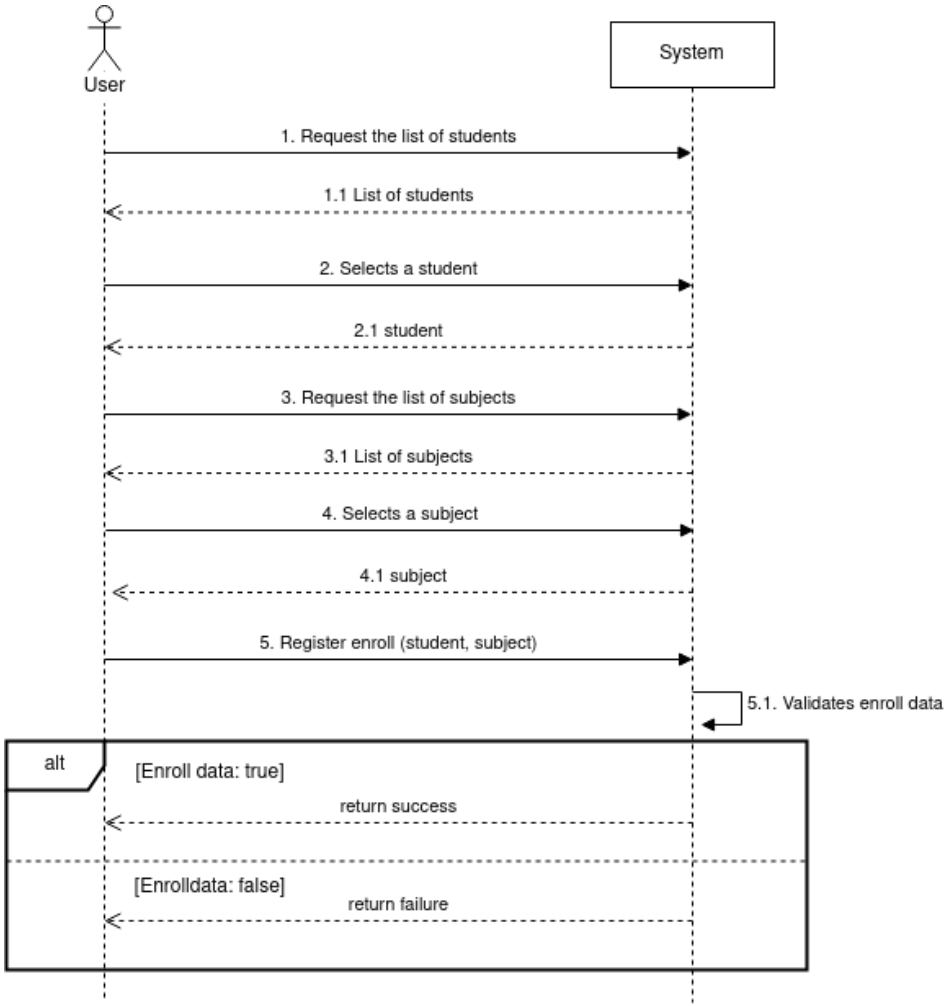
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



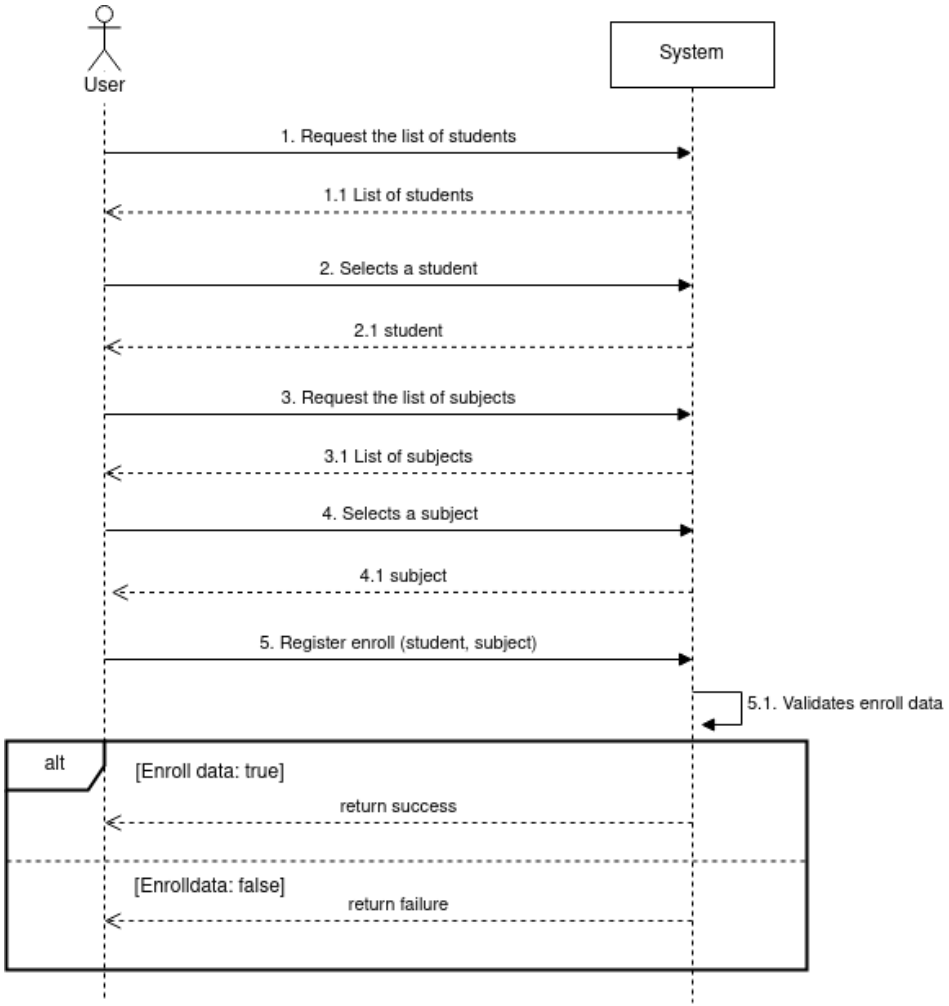
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



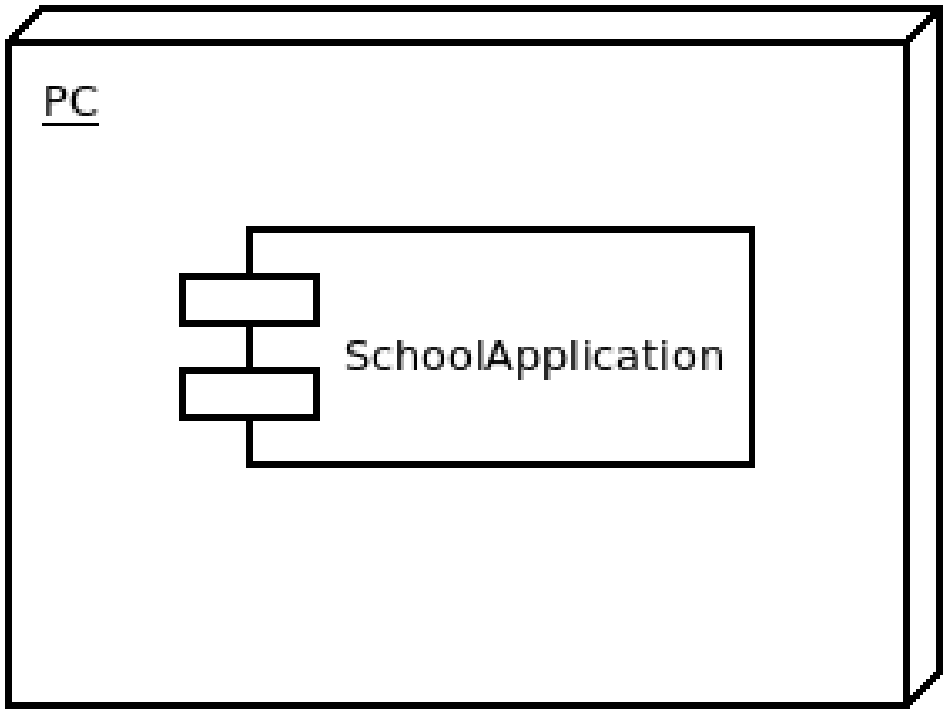
Analysis: Functional Requirements (V)

- SSD: UC 10 - Create Enroll



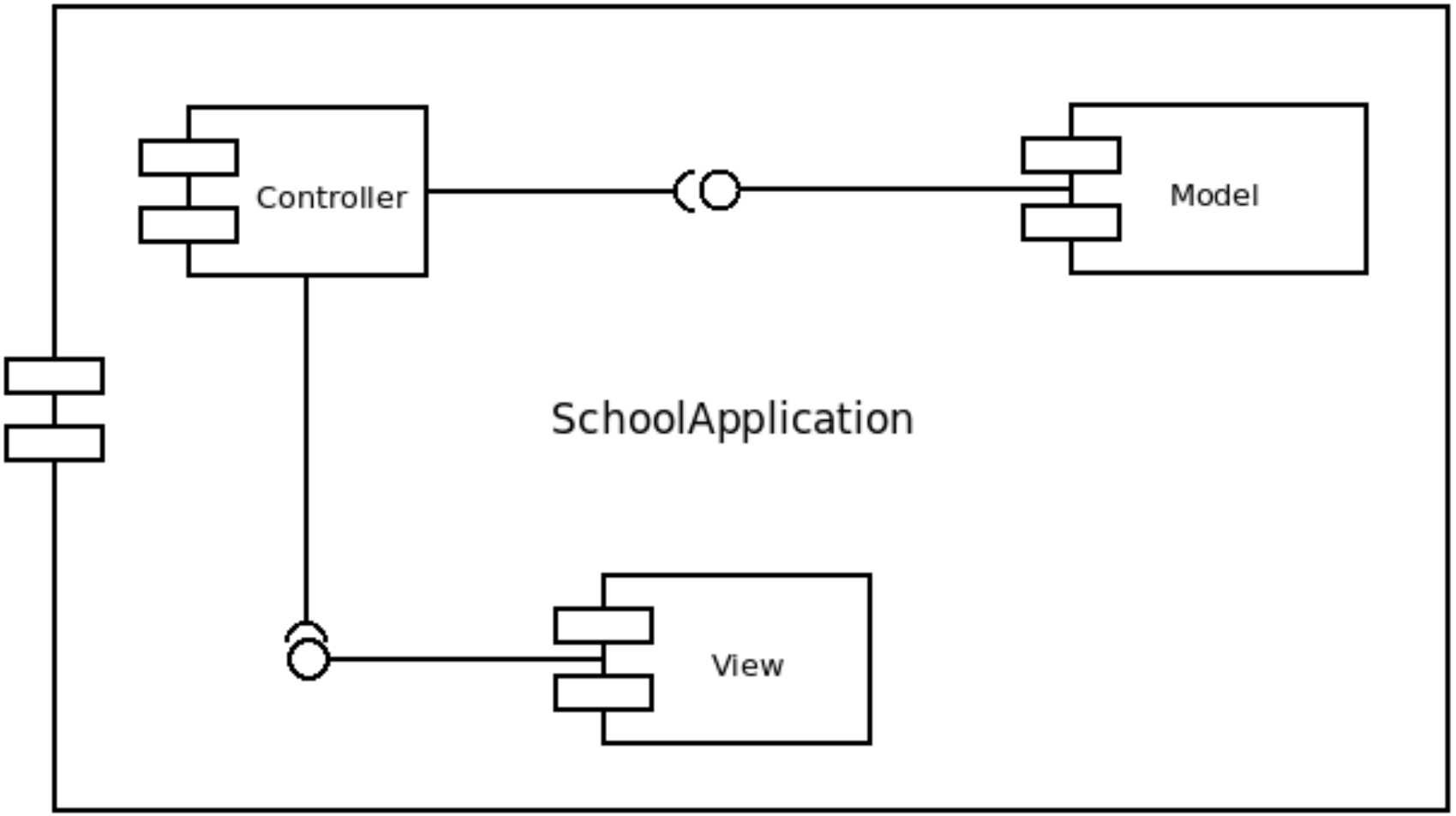
Design: Physical Architecture

- Usage of deployment diagram



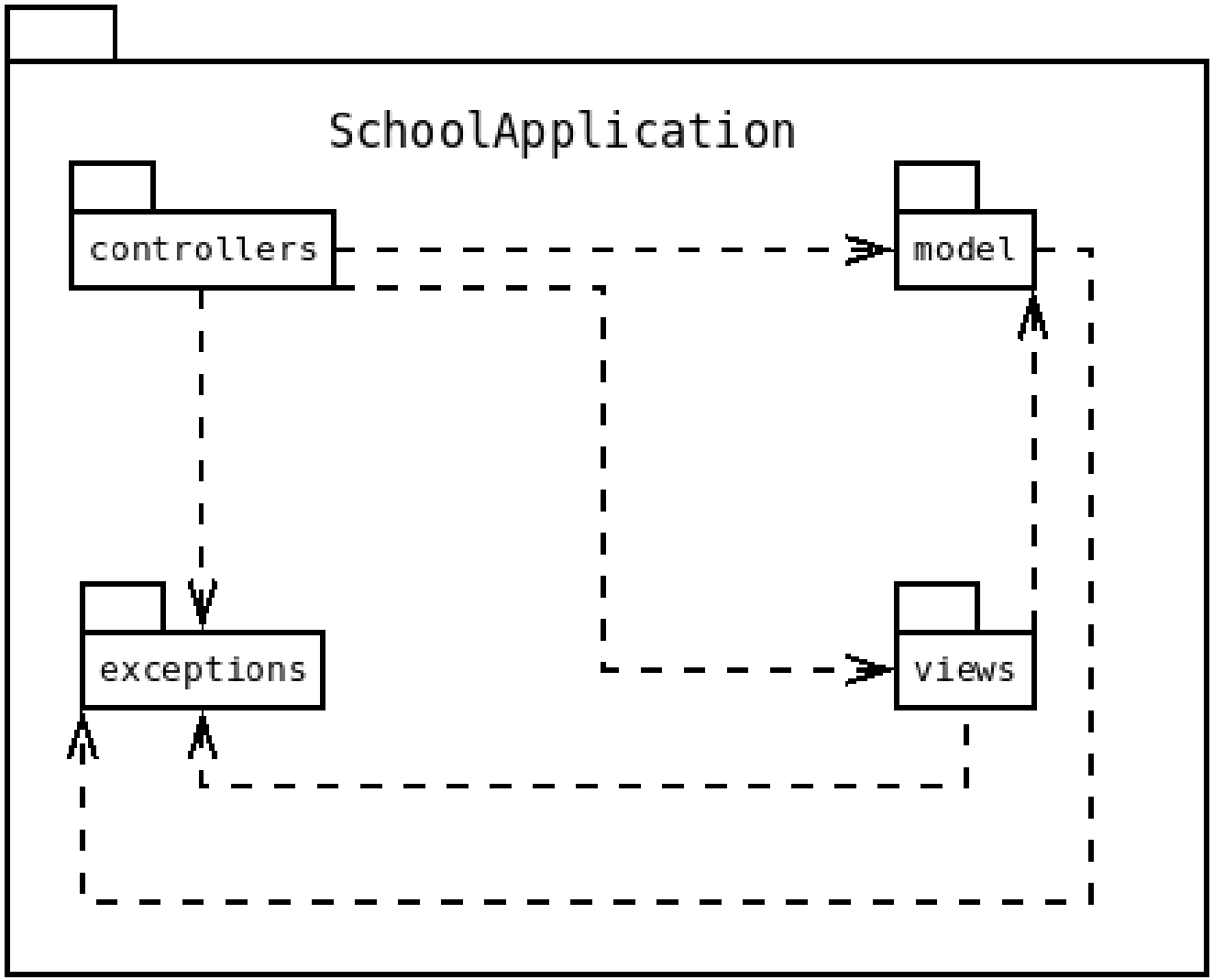
Design: Logical Architecture

- Usage of component diagram



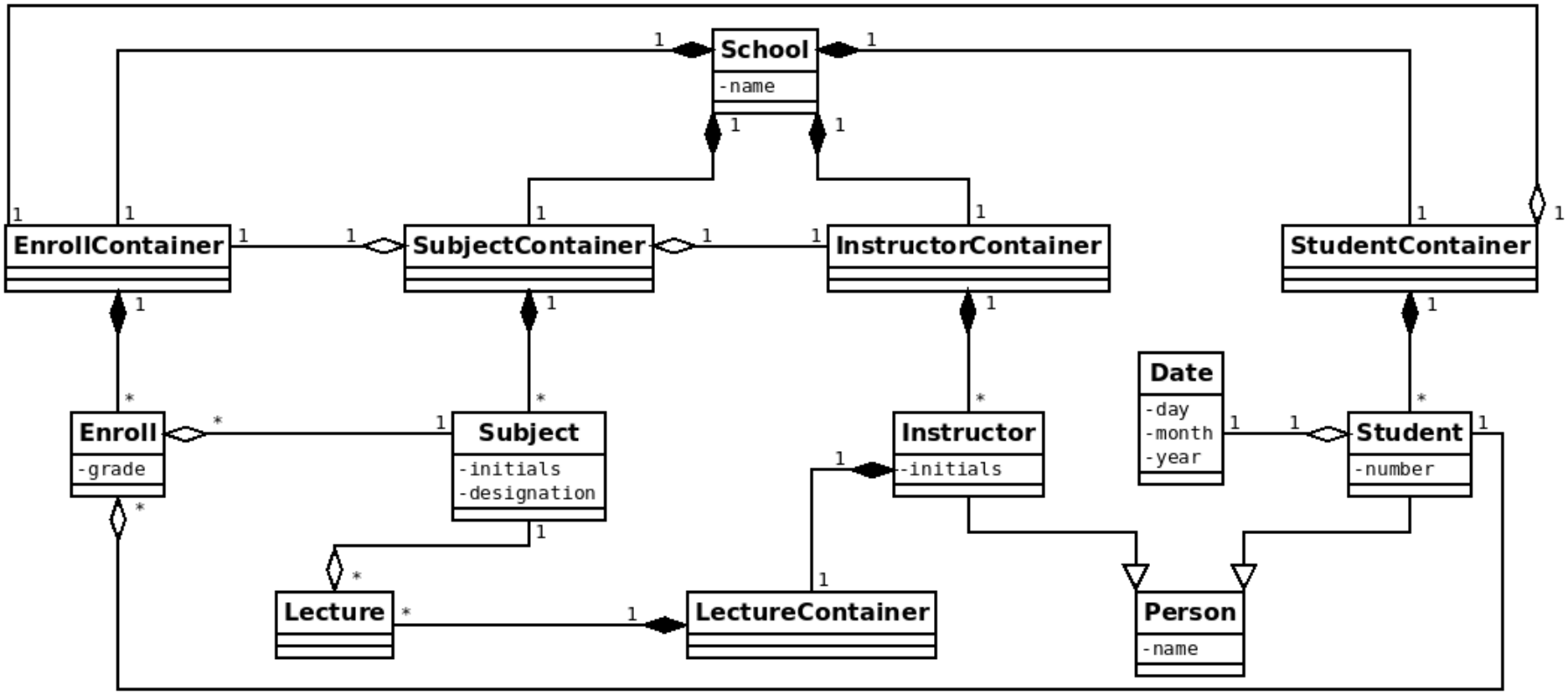
Design: Code Organization

- Usage of package diagram



Design: `model` Class Diagram

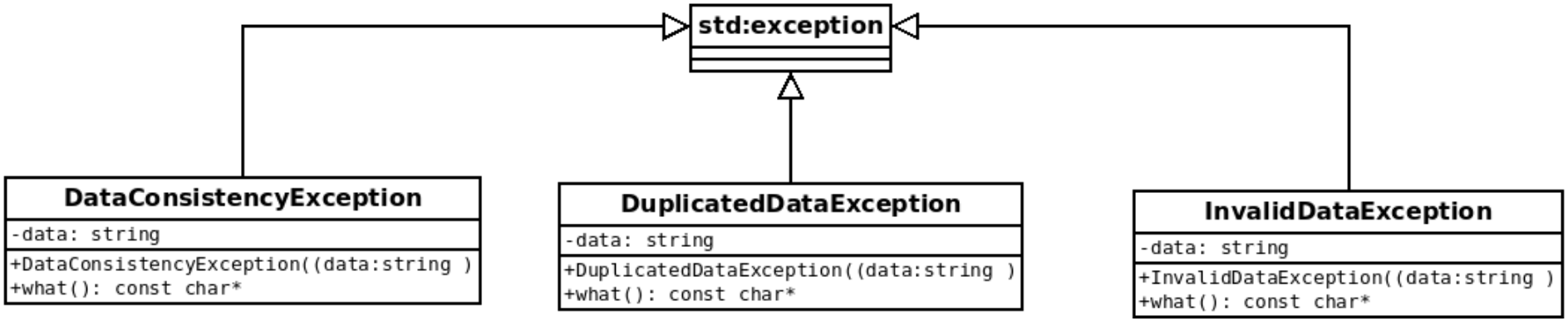
- Usage of class diagram



- It is missing:
  - Functions and attribute types.
  - Relations with exceptions' package classes.

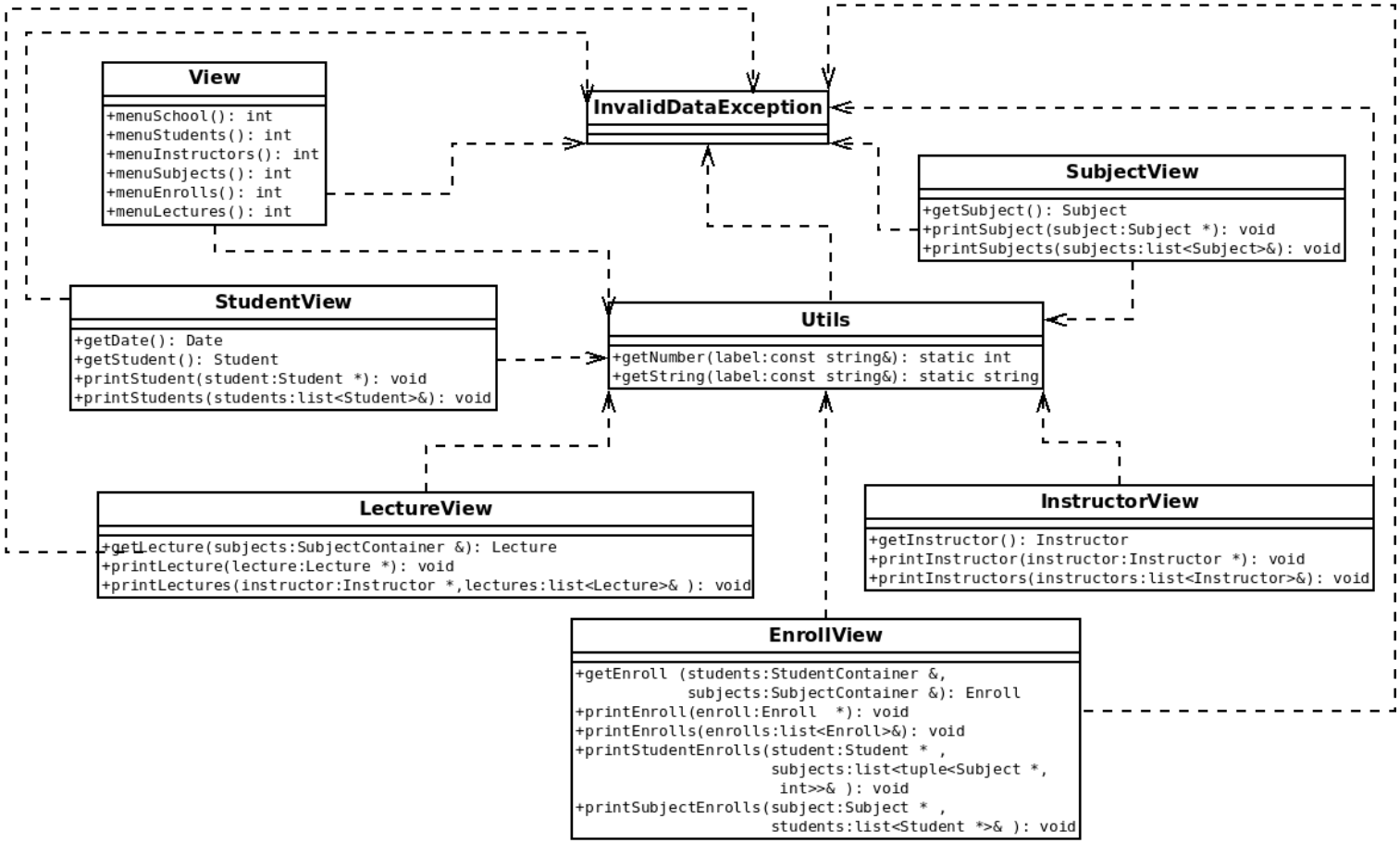
Design: `exceptions` Class Diagram

- Usage of class diagram



Design: `views` Class Diagram

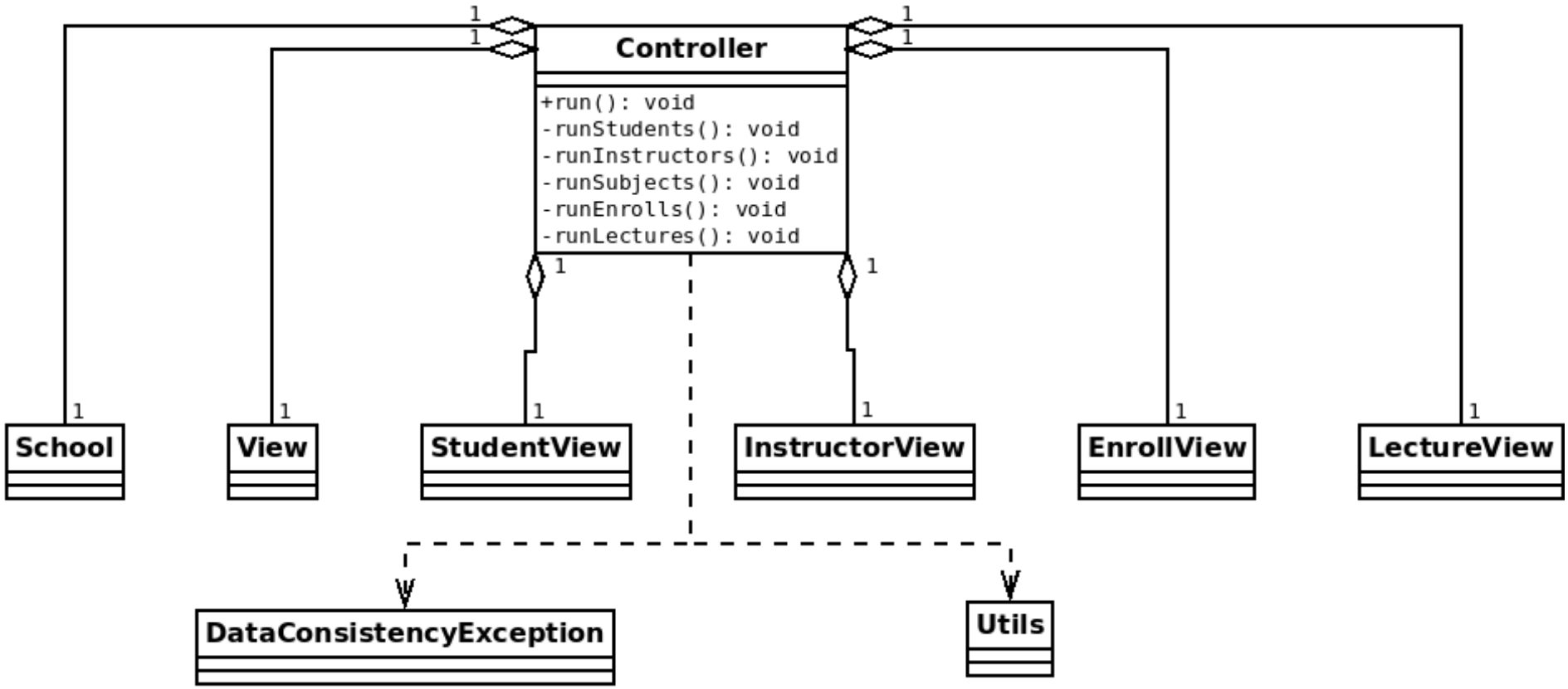
- Usage of class diagram



- It is missing:
  - Relations with model's package classes.

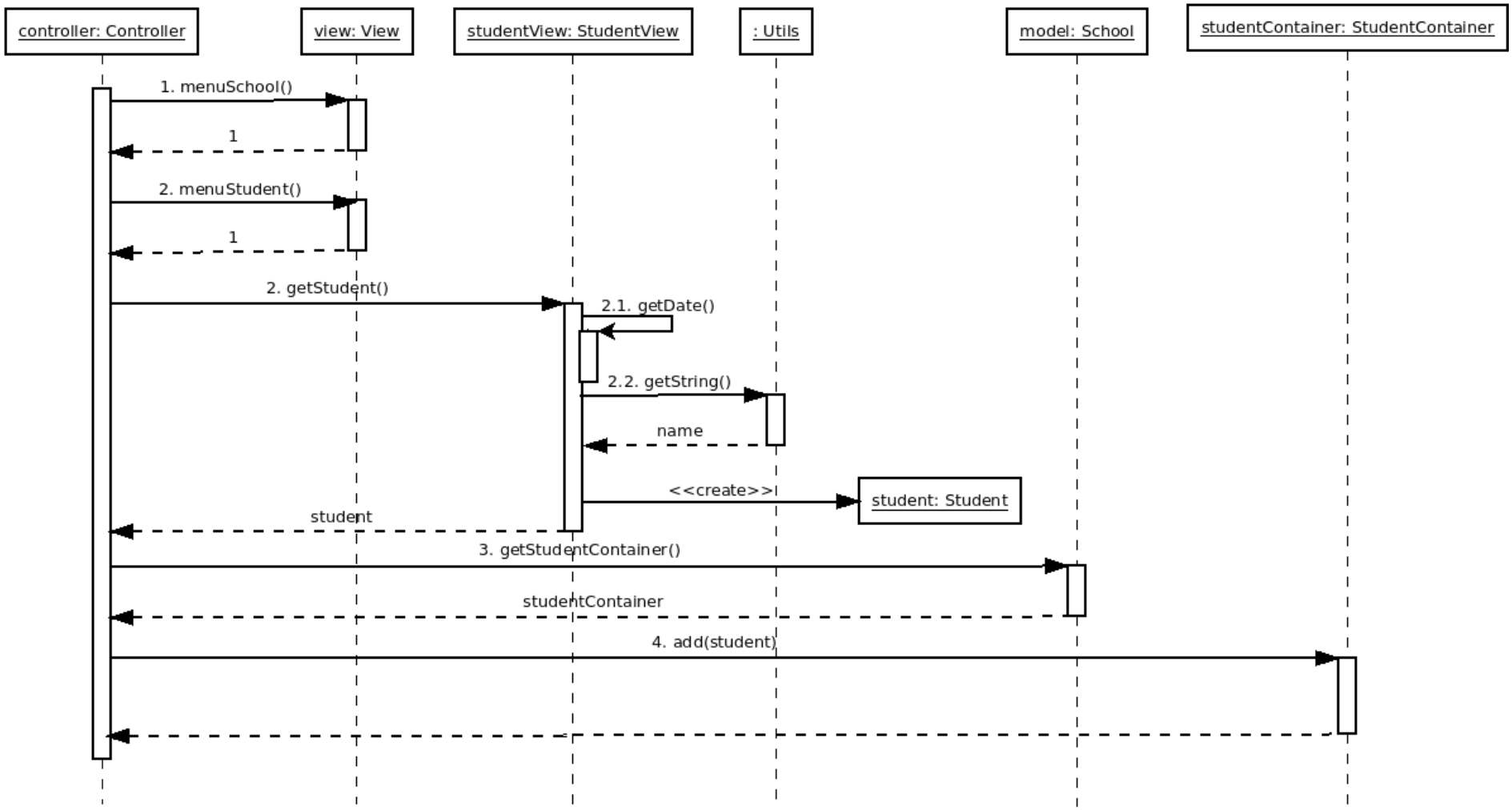
Design: `controllers` Class Diagram

- Usage of class diagram

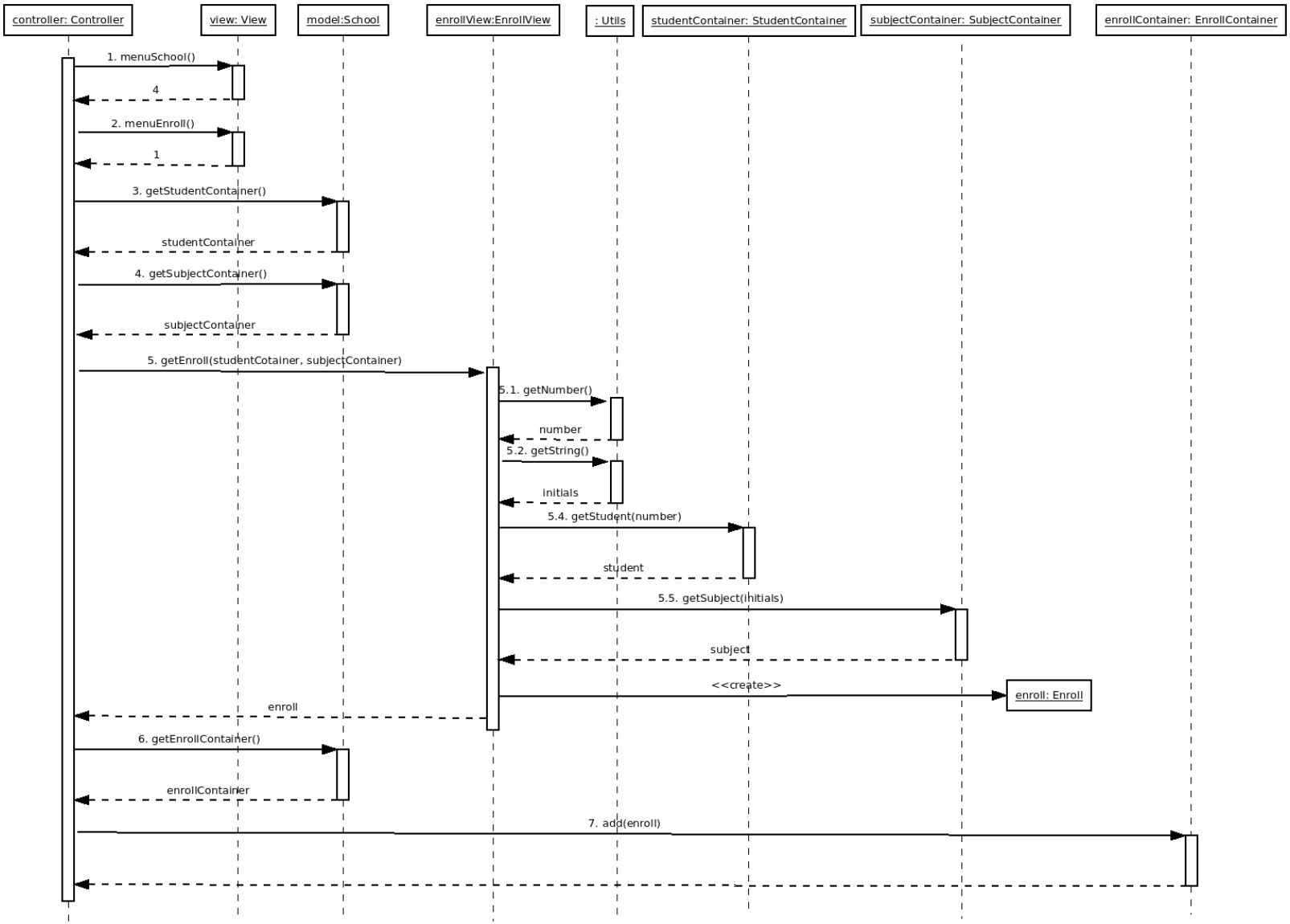


Design: UC 1 - Create Student

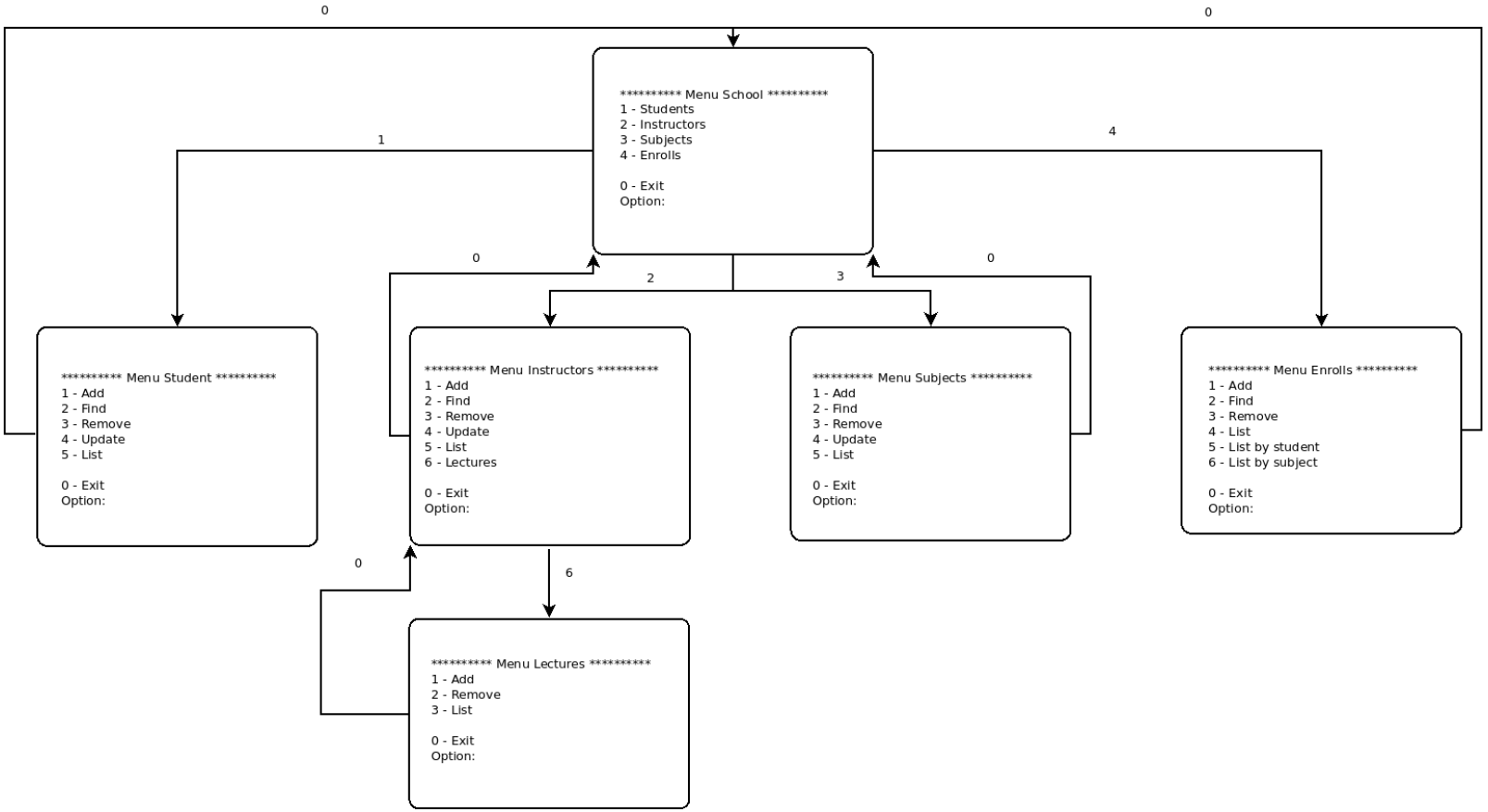
- Usage of sequence diagram



Design:UC 10 - Create Enroll



Design:User Interface



Bibliography

Resources

- "Big Java: Early Objects", 6th Edition by Cay S. Horstmann
- "Java™:The Complete Reference", 7th Edition,Herbert Schildt
- "Java™Programming", 7th Edition, Joyce Farrell
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html
- http://beginnersbook.com/java-tutorial-for-beginners-with-examples/
- https://www.leepoint.net/index.html
- https://junit.org/junit5/docs/current/user-guide/