<Ticket selling system for the Untold festival>

Analysis and Design Document

Student: Marin Andreea

**Group: 30431**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The objective of this assignment was to allow us to become familiar with layers architectural pattern, repository, services and unit tests by designing and implementing a ticket selling system for the Untold festival. The application presents two types of users (admin and cashier) which have to provide a username and a password in order to use the application.

# Functional Requirements

The application presents the following functional requirements:

* Admin users can perform CRUD operations on cashiers’ information
* Admin users can perform CRUD operations on concerts’ information
* Admin users can perform CRUD operations on bands’ information
* Admin users can perform CRUD operations on concertbands’ information
* Admin users can export all tickets that were sold for a certain concert to a .csv file
* Cashier users can perform CRUD operations on tickets’ information

# Non-functional Requirements

The application presents the following non-functional requirements:

* The system requires authentication
* The system is easy to work with

2. Use-Case Model

*[Create the use-case diagrams and provide one use-case description (according to the format below).*

*Use-Case description format:*

*Use case: <use case goal>*

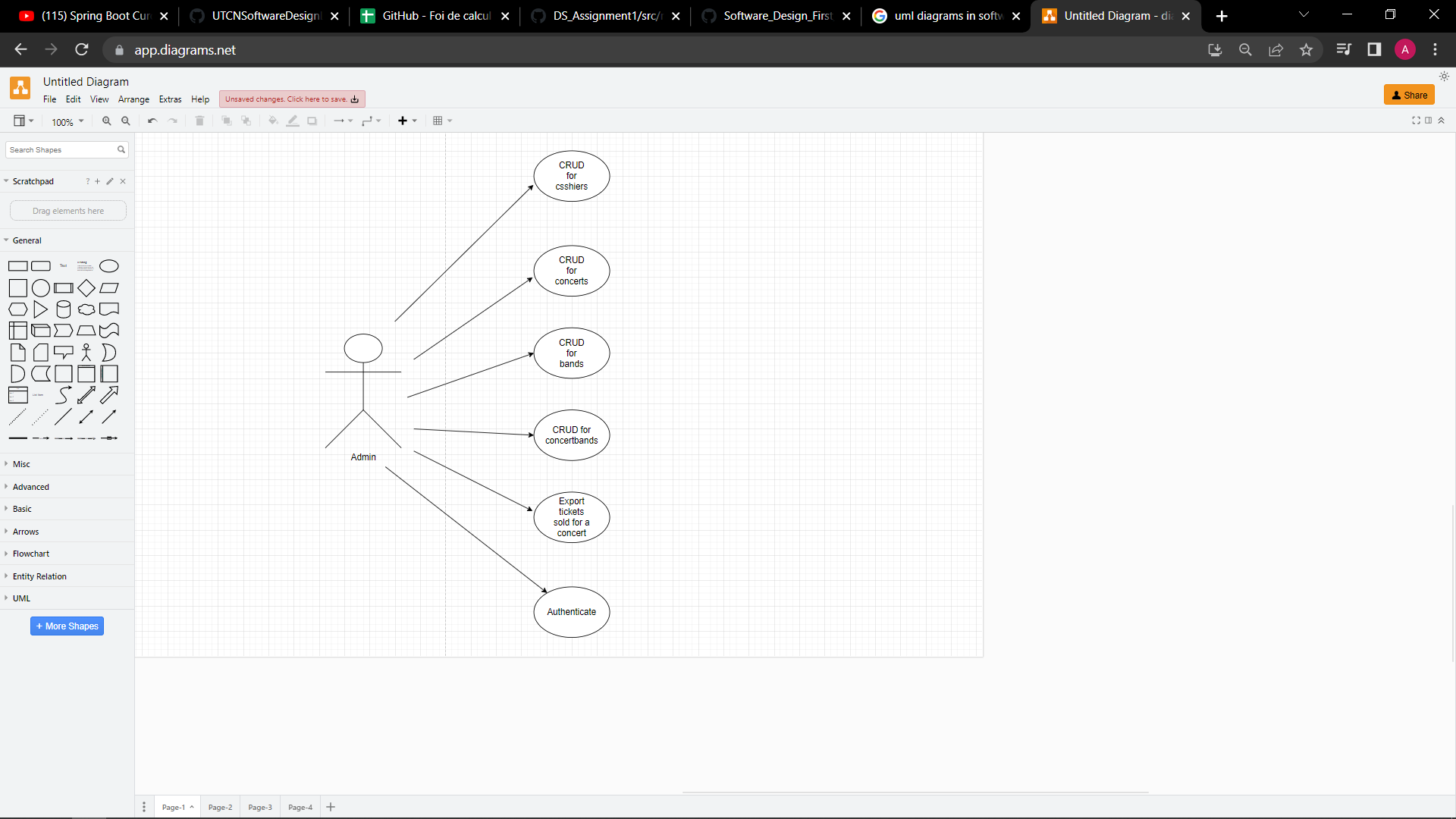
*Level: <one of: summary level, user-goal level, sub-function>*

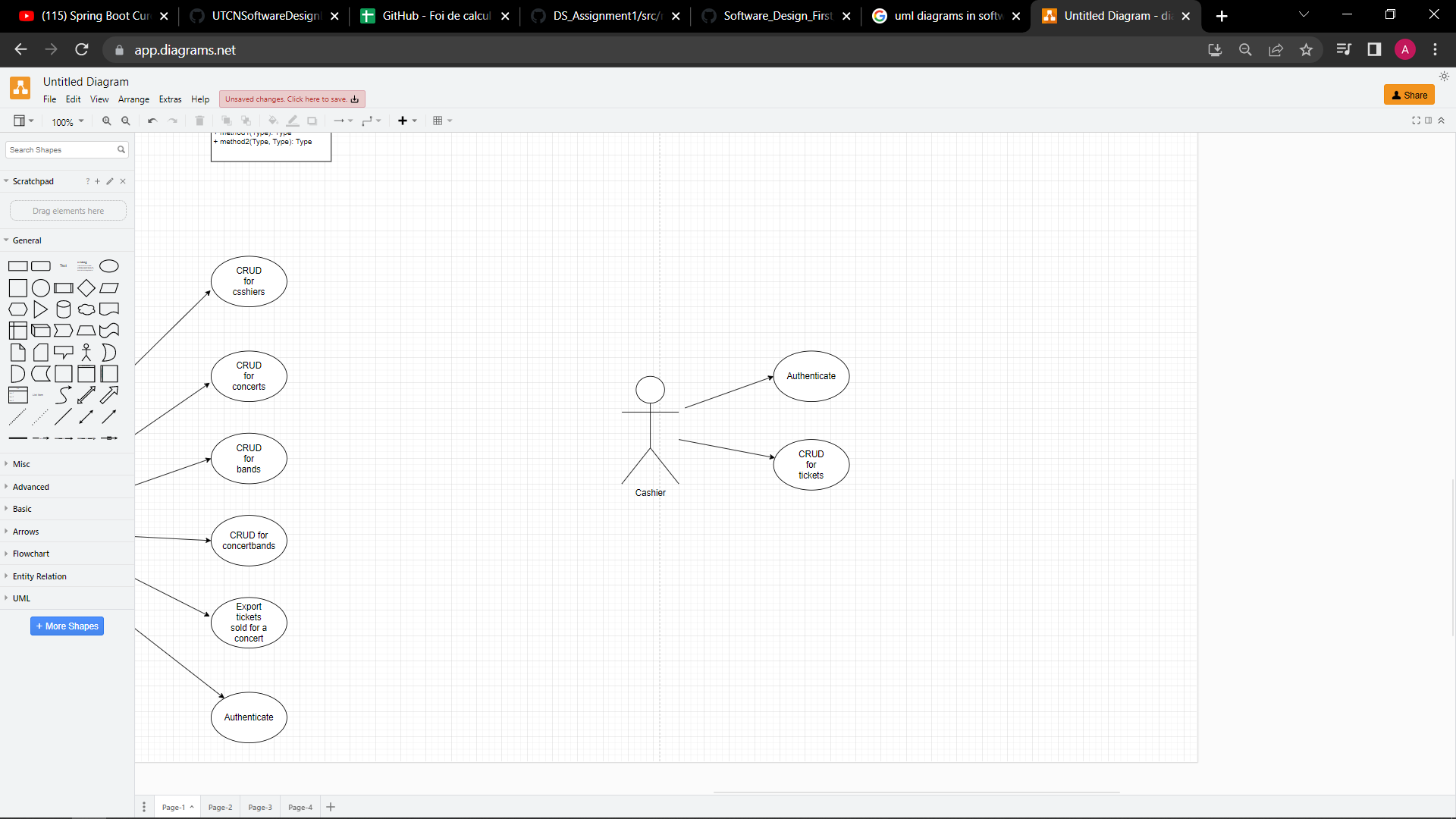
*Primary actor: <a role name for the actor who initiates the use case>*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*

*]*





**Use-case:** Create a new cashier

**Level:**

**Primary actor:** Admin

**Main success scenario:** The admin introduces his credentials. If the login was successful, the admin is able to create a new cashier by introducing the credentials of the new cashier.

**Extensions:** The system will notify that the credentials introduced by the admin are wrong and the login was unsuccessful.

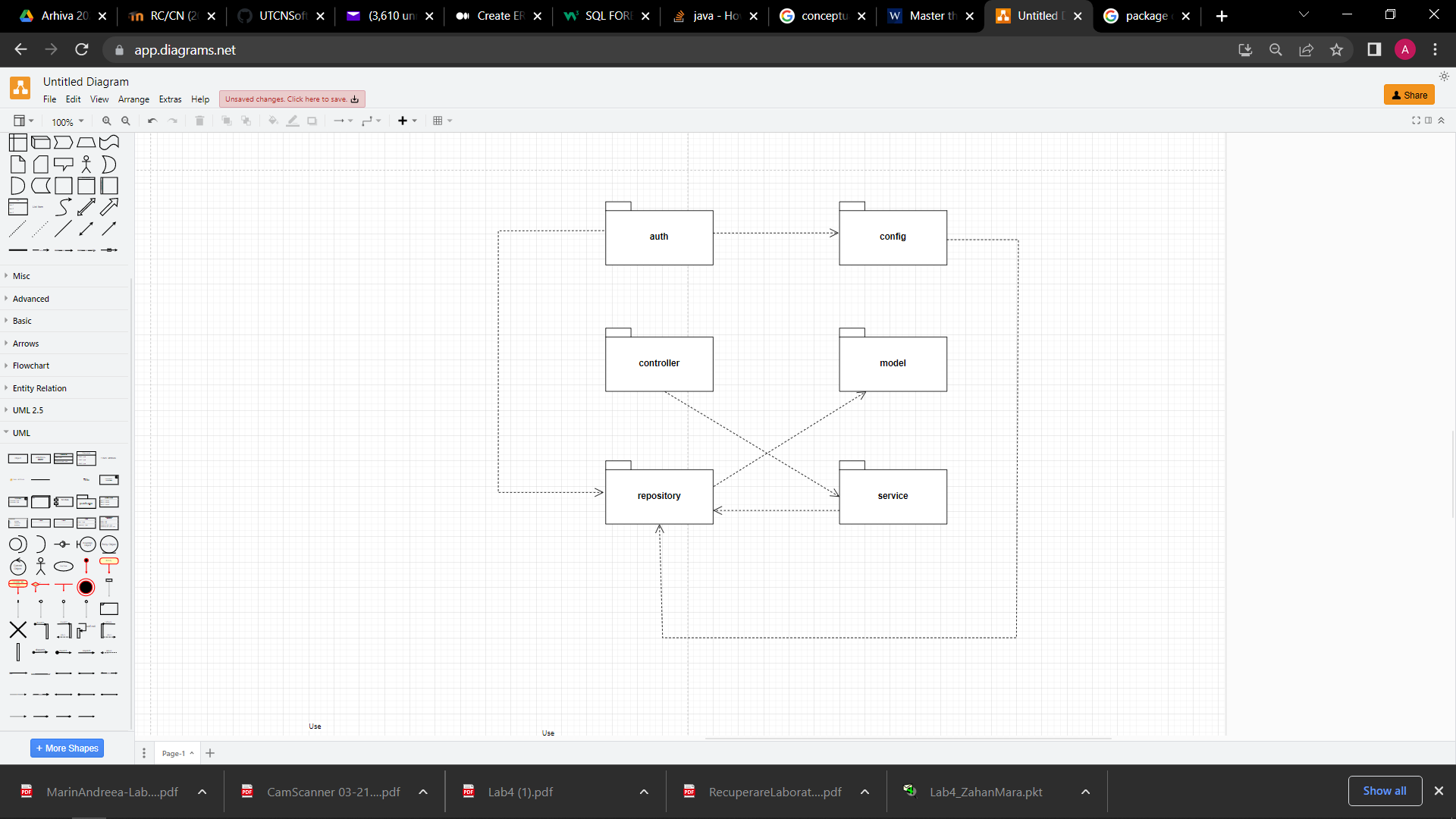
3. System Architectural Design

**3.1 Architectural Pattern Description**

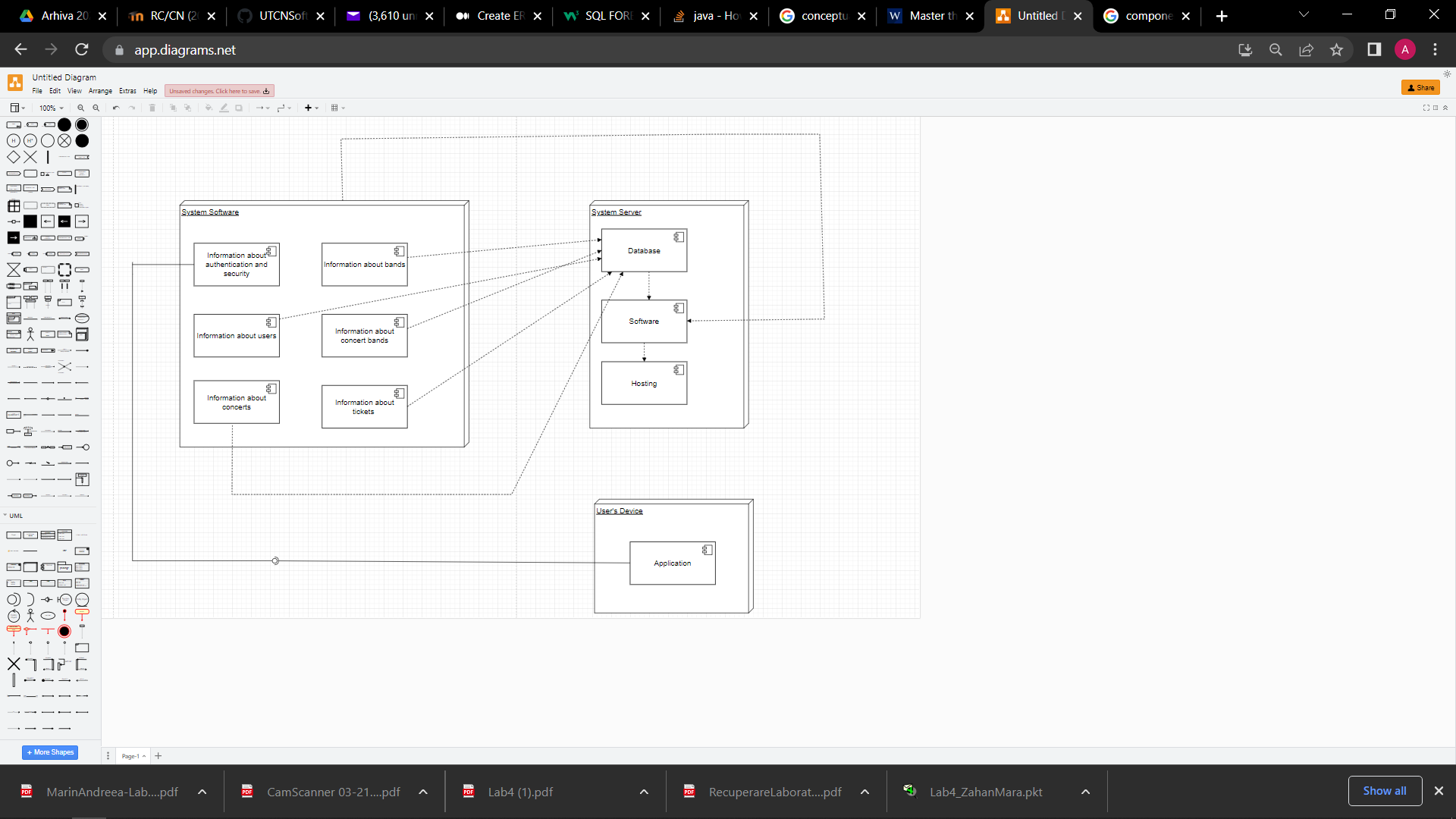
The architectural pattern used for this project is layered architecture. It is also known as an n-tier architecture and describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. A major characteristic of this framework is that layers are only connected to the layers directly below them. It comes with the advantage that we can make changes in one layer without affecting the other layers.

**3.2 Diagrams**

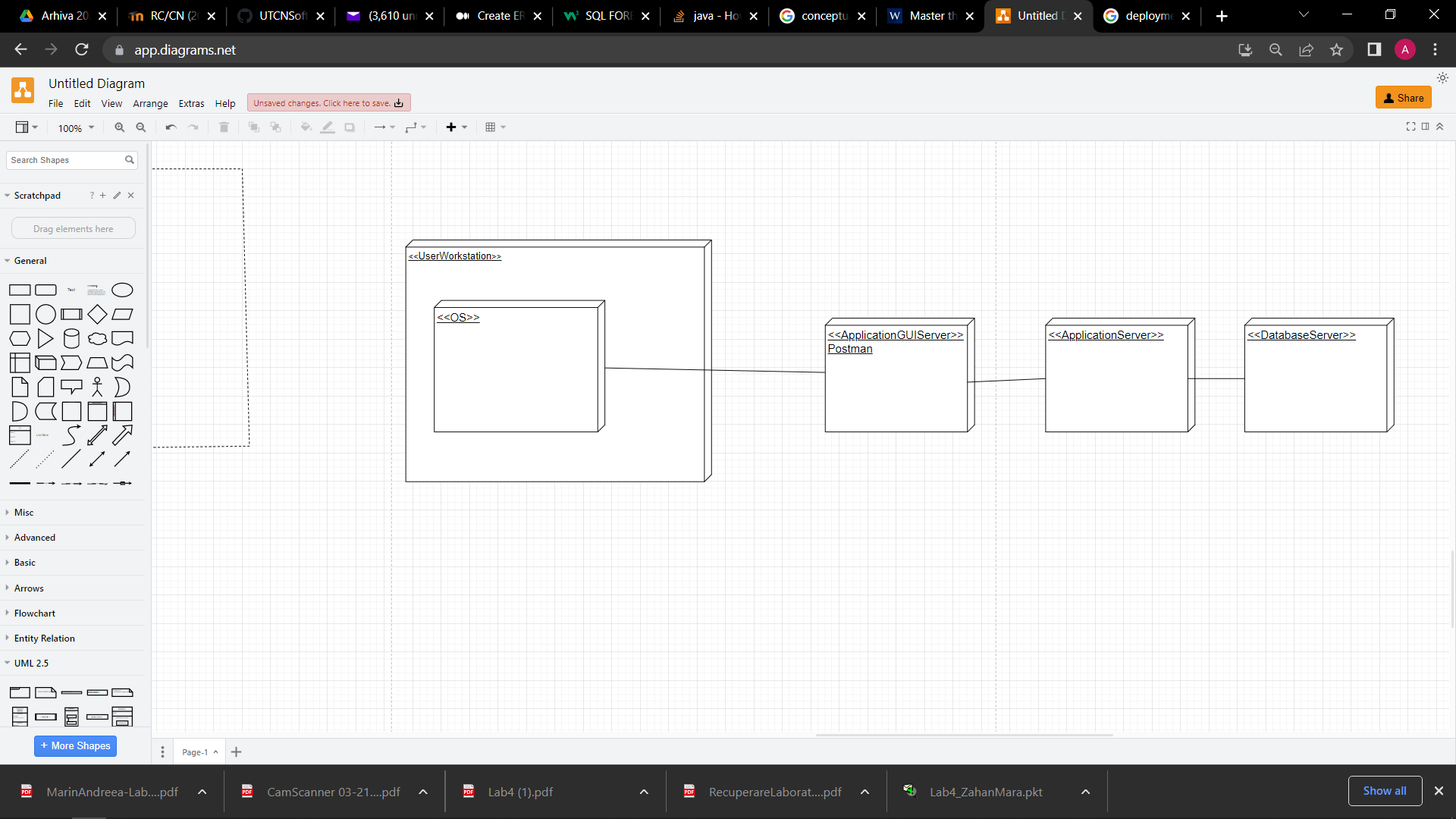
* Package Diagram



* Component Diagram

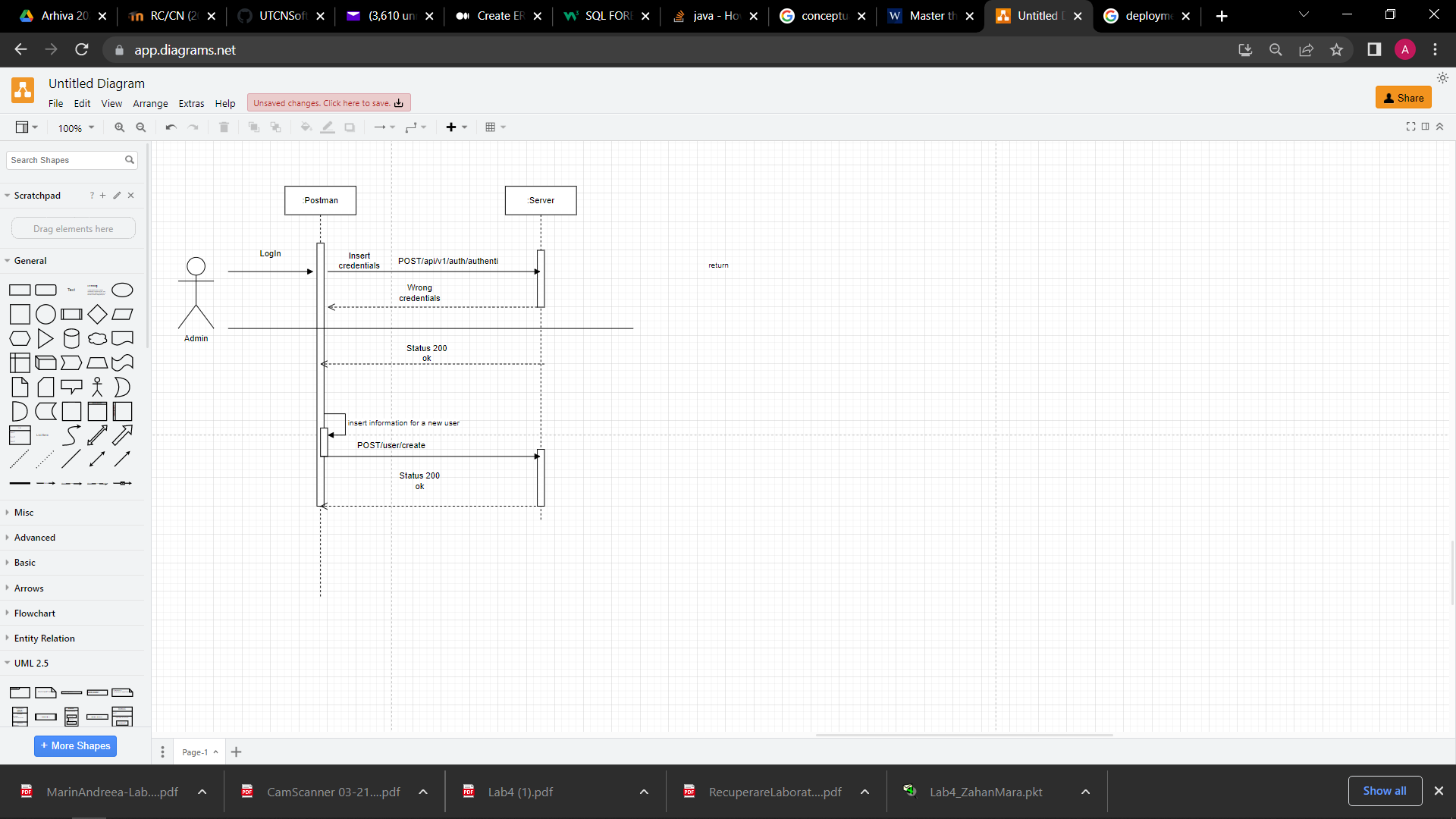


* Deployment Diagram



4. UML Sequence Diagrams

UML Sequence Diagram for creating a new user.



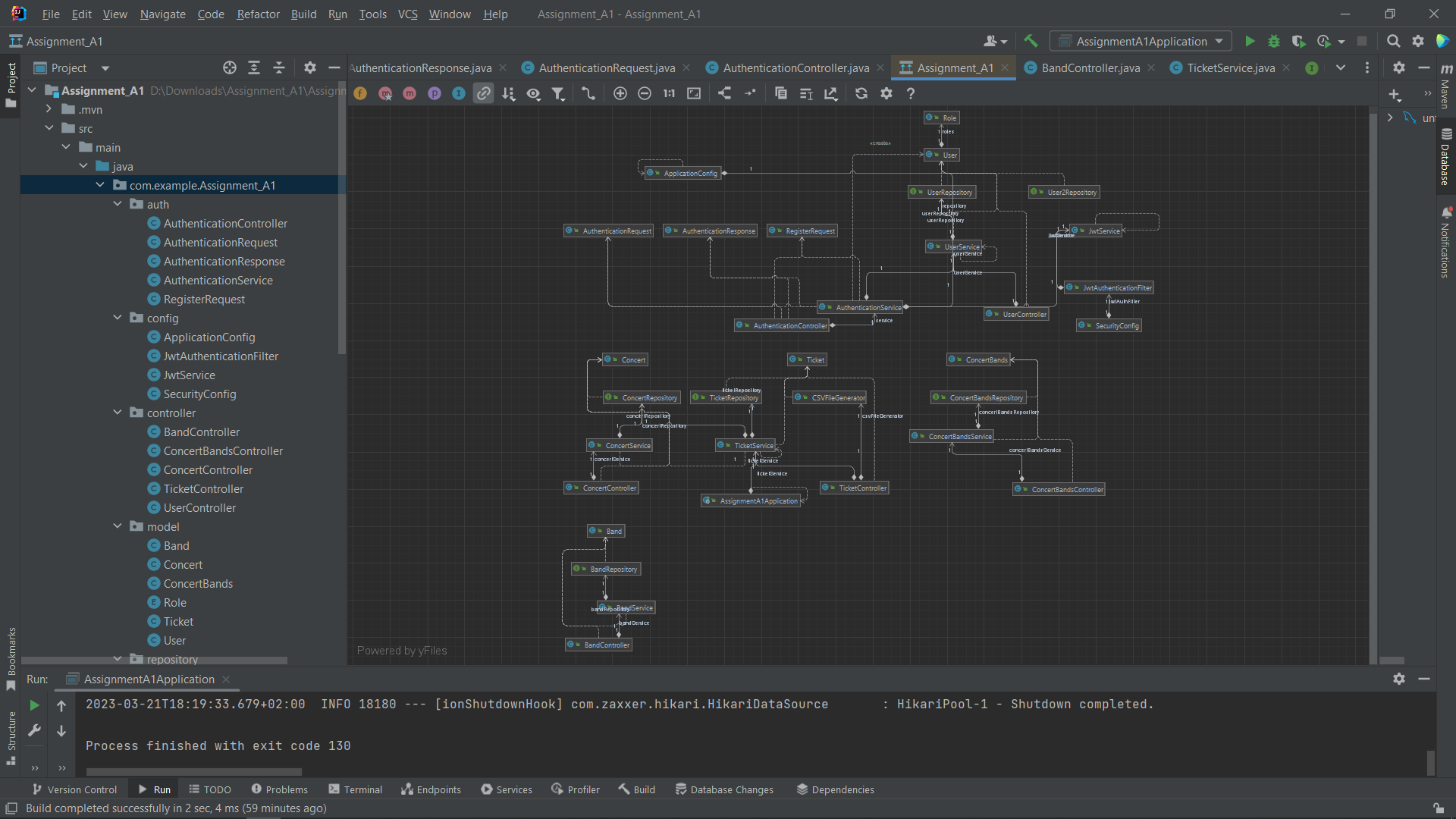
5. Class Design

**5.1 Design Patterns Description**

In software engineering, dependency injection is a design pattern in which an object or function receives other objects or functions that it depends on. A form of inversion of control, dependency injection aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs.

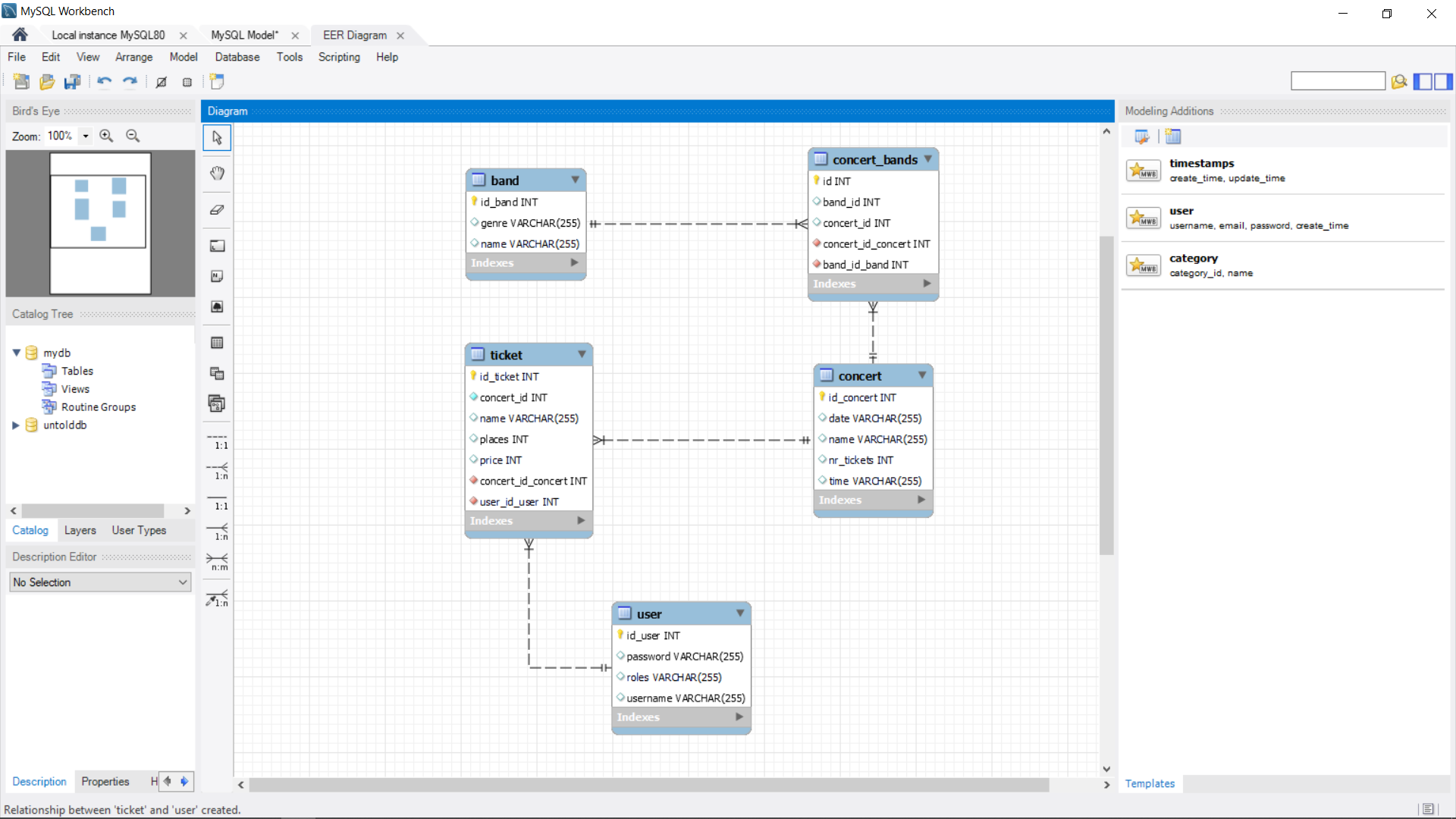
**5.2 UML Class Diagram**

The dependency injection pattern ensures that an object or function which wants to use a given [service](https://en.wikipedia.org/wiki/Service_(systems_architecture)) should not have to know how to construct those services.



6. Data Model

The system implements five data models: User, Ticket, Concert, Band and ConcertBands. The User data model contains an id, username, password and a role(CASHIER or ADMIN). The username and password will be used for authentication. The ticket data model contains an id, a concert id, name, price and number of places. CRUD operations on tickets can be performed only by users having the CASHIER role. CRUD operations for concert, band, and concertBands are performed only by admin.



7. System Testing

For testing the system I used JUnit. The system provides tests for password encryption and for the number of tickets for a concert exceeded validation.

8. Bibliography

* <https://www.youtube.com/playlist?list=PLqq-6Pq4lTTbx8p2oCgcAQGQyqN8XeA1x>
* <https://www.youtube.com/watch?v=KxqlJblhzfI>
* <https://www.youtube.com/watch?v=9SGDpanrc8U>
* https://www.baeldung.com/spring-boot-data-sql-and-schema-sql