Student: Sebastian George Urda

**Group:30433**

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

# Since the Ping-Pong Tournaments application is such a big success, the owners of the application wish to add a new feature to it: Paid Tournaments. They differ from the free tournaments available until now by the fact that they require an enrollment fee and offer a cash prize to the winner. For the moment, the player that finishes on the 1st position receives all the money from the enrollment fees of the tournament. From the beginning of the tournament until a player wins 1st place, the prize money is kept in the account of the Ping-Pong Association or in the account of the Tournament itself. Based on Assignment A1, adapt your application to fit the new requirements. Functional Requirements

# Non-functional Requirements

The regular user can perform the following additional operations:

* Enroll into upcoming Tournaments, by paying the enrolment fee out of their account;
* View Tournaments by category: Free Paid
* Search Tournaments by name
* Update their data

The administrator can perform the following additional operations:

* Create Tournaments
* View Tournaments by category: Free Paid
* Search Tournaments by name and
* Create Account
* Delete Tournaments
* Delete Account
* Update Account including balance

Application Constraints and Technical Requirements:

* Use the Model View Controller Pattern for all views;
* The Data Access Layer (DAL) will be re-implemented using an ORM framework;

2. Use-Case Model

Use case: Modify your own data

Primary actor: user

Main success scenario: login with your account->select the modify your data option-> get data modified

Extensions: fail login

Use case: Create Tournament

Primary actor: admin

Main success scenario: login with admin account->select the create account option-> get the success message

Extensions: fail login, fail to create

Use case: View Tournaments

Primary actor: admin/user

Main success scenario: login with admin/user account->select in the choice box the tournament -> see the tournament window

Extensions: fail login

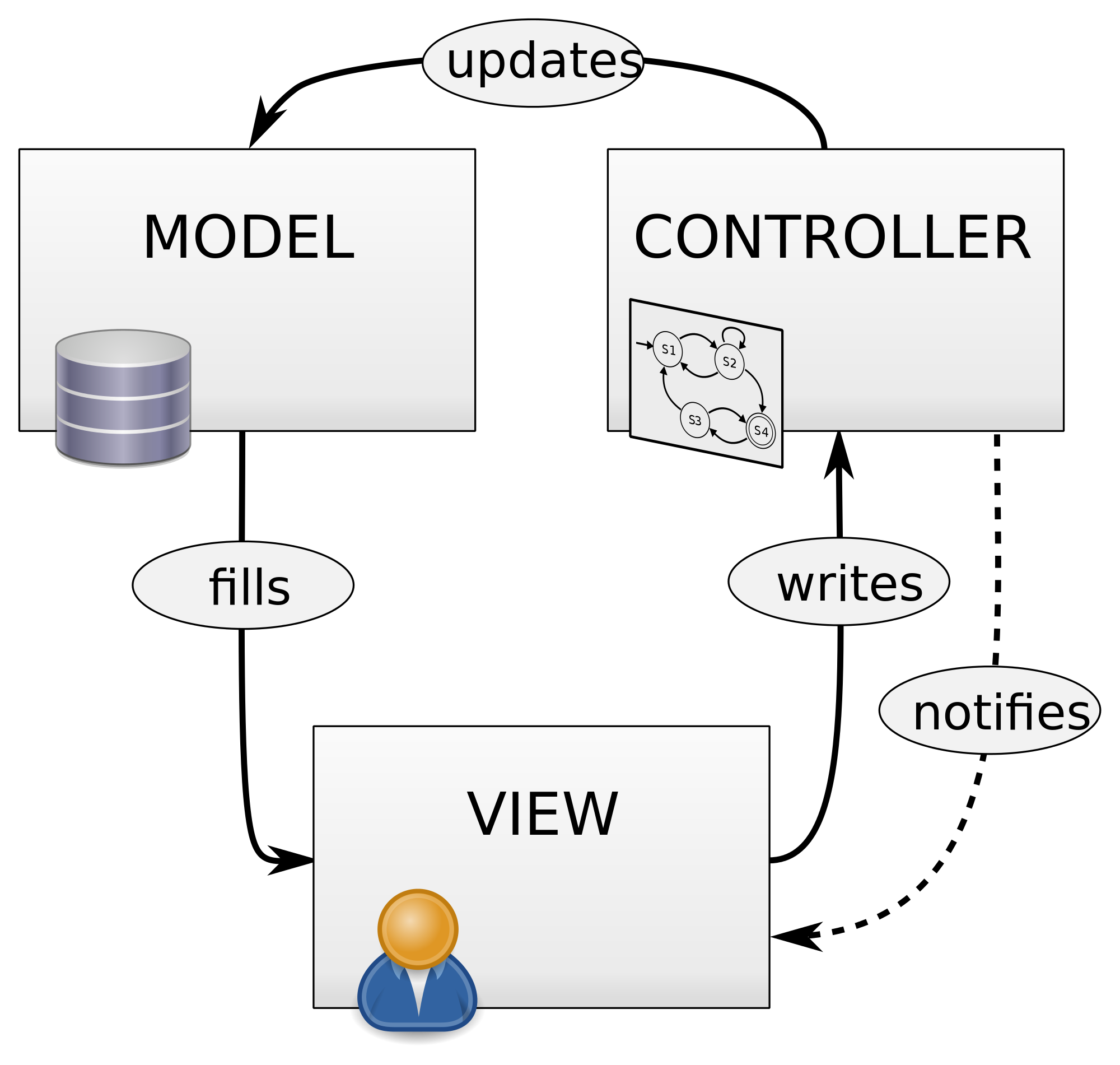
3. System Architectural Design

**3.1 Architectural Pattern Description**

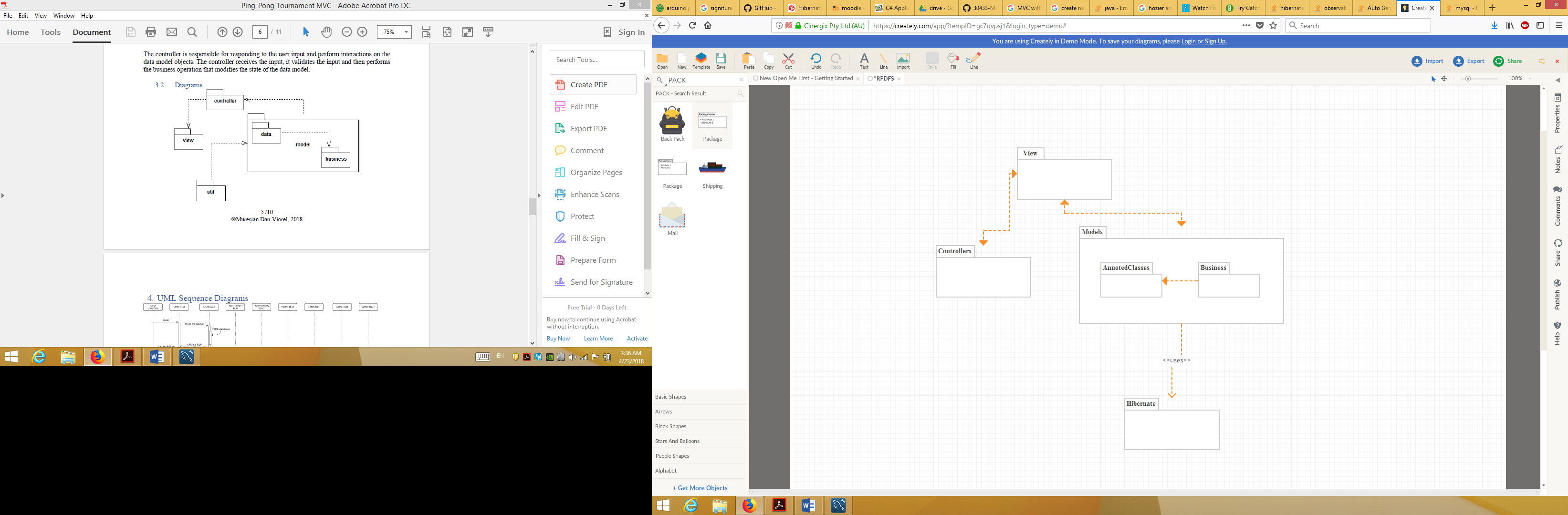
Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

* **Model** − The lowest level of the pattern which is responsible for maintaining data;
* **View** − This is responsible for displaying all or a portion of the data to the user;
* **Controller** − Software Code that controls the interactions between the Model and View.

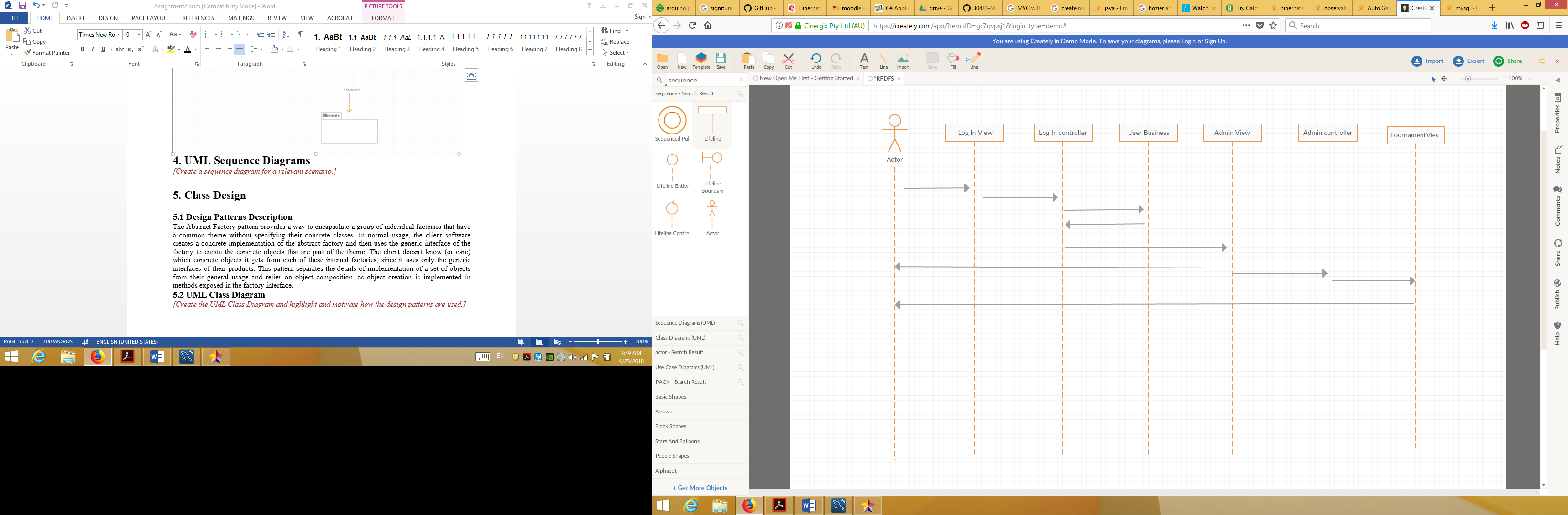
MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.



**3.2 Diagrams**



4. UML Sequence Diagrams

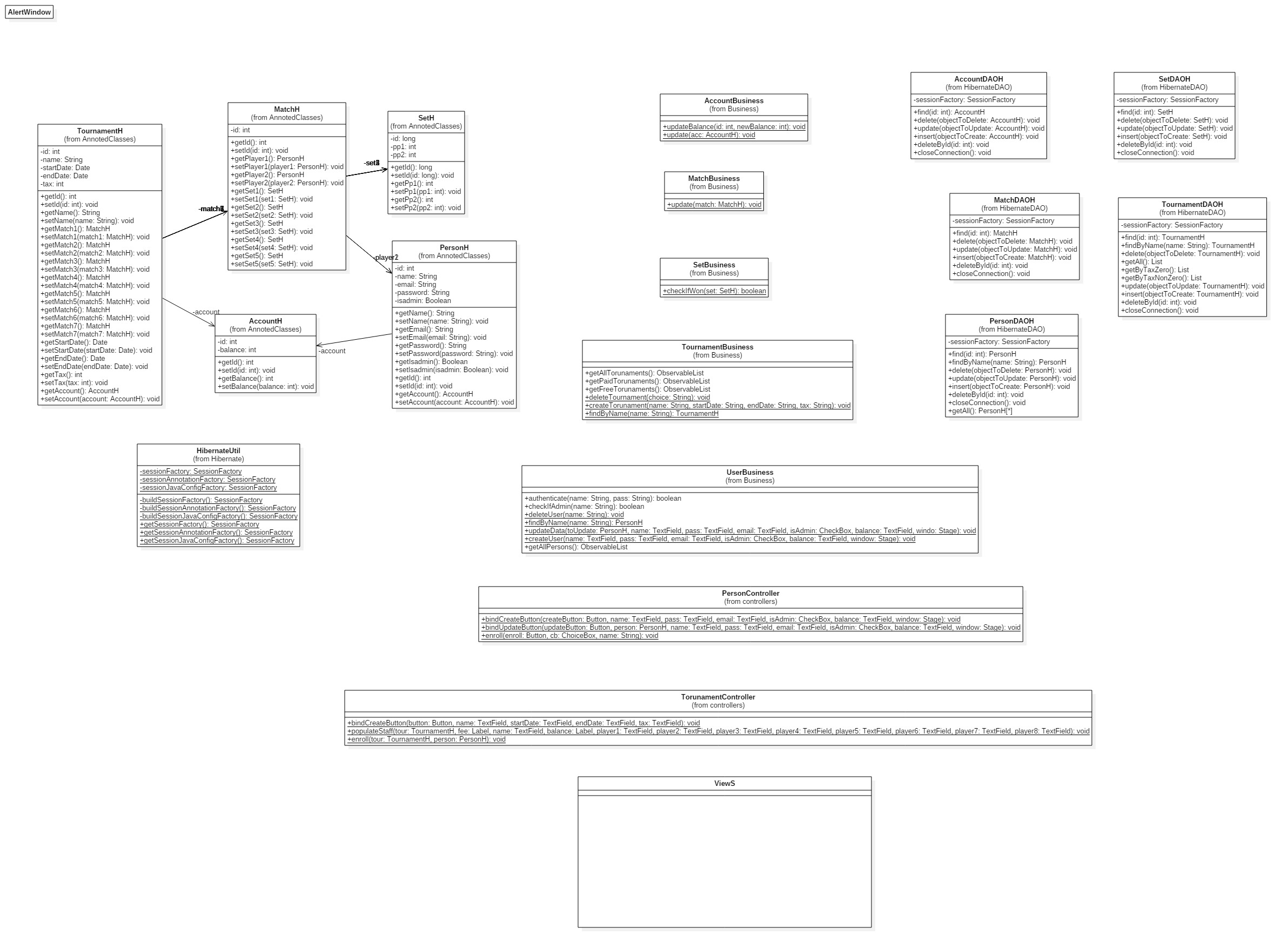


5. Class Design

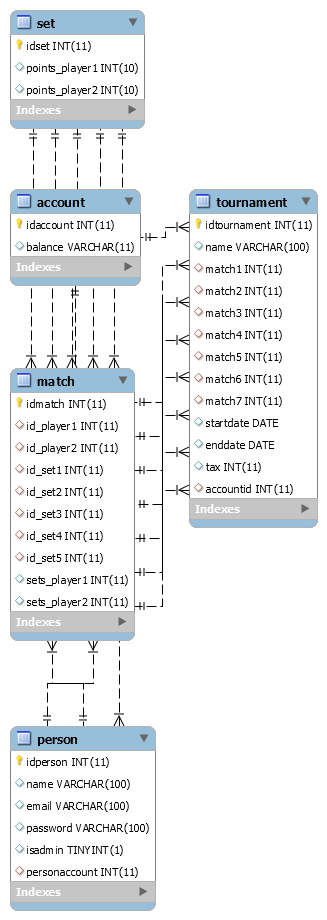
**5.1 Design Patterns Description**

The Abstract Factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes. In normal usage, the client software creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create the concrete objects that are part of the theme. The client doesn't know (or care) which concrete objects it gets from each of these internal factories, since it uses only the generic interfaces of their products. This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface.

**5.2 UML Class Diagram**



6. Data Model

**

7. System Testing

2 Junit Tests were created in order to test the insertions into the Account, Person and Tournament Table.

8. Bibliography

1. https://pongworld.com/table-tennis-sport/rules;

2. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller;

3. https://www.journaldev.com/2954/hibernate-query-language-hql-example-tutorial;

4. https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html;

5. https://en.wikipedia.org/wiki/Abstract\_factory\_pattern;

6. https://www.journaldev.com/3793/hibernate-tutorial.