Sport Event Application “Fun&Health”

Analysis and Design Document

Student:Casap Dumitru

**Group:30239**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

*Sport event application “Health&Fun” is a web application which will run on a server. The app is proposed to all people who like sport. The application has two types of users ( a regular user represented by the clients who want to do sport and an administrator user) which have to provide a username and a password in order to use the application.*

*The regular user can perform the following operations:*

* *Create a sport event*
* *View all/near sport events*
* *Update/delete his sport events.*
* *Edit his account.*
* *Select favorite sports.*
* *Create a sport event.*
* *Going/Ungoing to a sport event.*
* *Add comment to a sport event.*
* *Add to friend/ chat with other users*

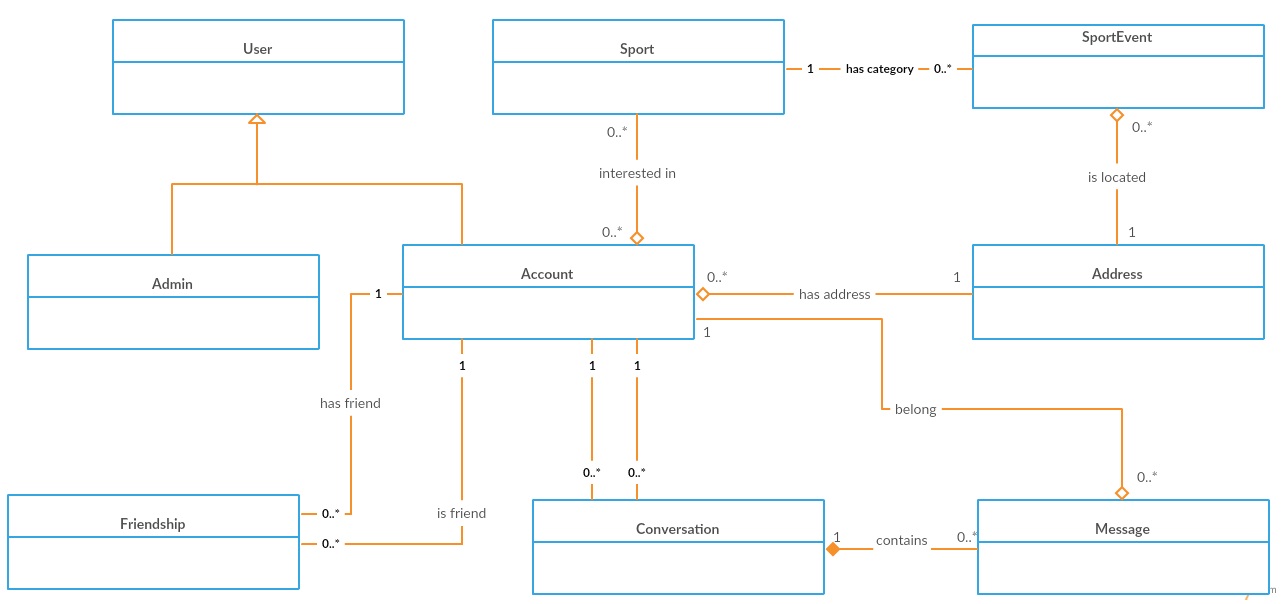
*The administrator can perform the following operations:*

* *Delete sport events*
* *Delete accounts*
* *CRUD on type of sport*
* *Add sport news*

# Elaboration – Iteration 1.1

# Domain Model

A domain model is a system of abstractions that describes selected aspects of a sphere of knowledge. It is a representation of meaningful real-world concepts pertinent to the application that need to be modeled in software. In the case of this application, the domain model are represented by model classes : User, Admin, Account, Sport, SportEvent, Friendship, Conversation, Message, Address and relations between them. The main actor is account, which all be the client of this application. All models are related with Account model. The domain model of the sport event application “Health&Fun” are better explained by the conceptual class diagram.



# Architectural Design

## Conceptual Architecture

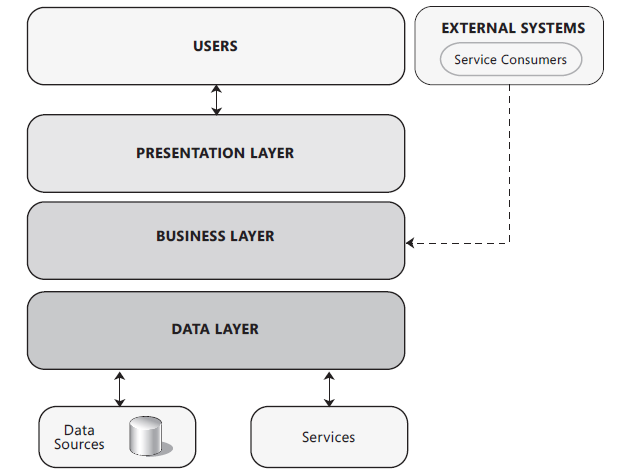
The architectural design pattern used in this project is the layers architectural pattern (Figure 3). An application can consist of a number of basic layers and the common three-layer design consists of the following layers:

**Presentation layer**–  is responsible for the delivery and formatting of information to the application layer for further processing or display. The presentation layer is the lowest layer at which application programmers consider [data structure](https://en.wikipedia.org/wiki/Data_structure) and presentation

**Business layer**– implements the core functionality of the system and encapsulates the relevant business logic

**Data layer** – provides access to data hosted within the boundaries of the system

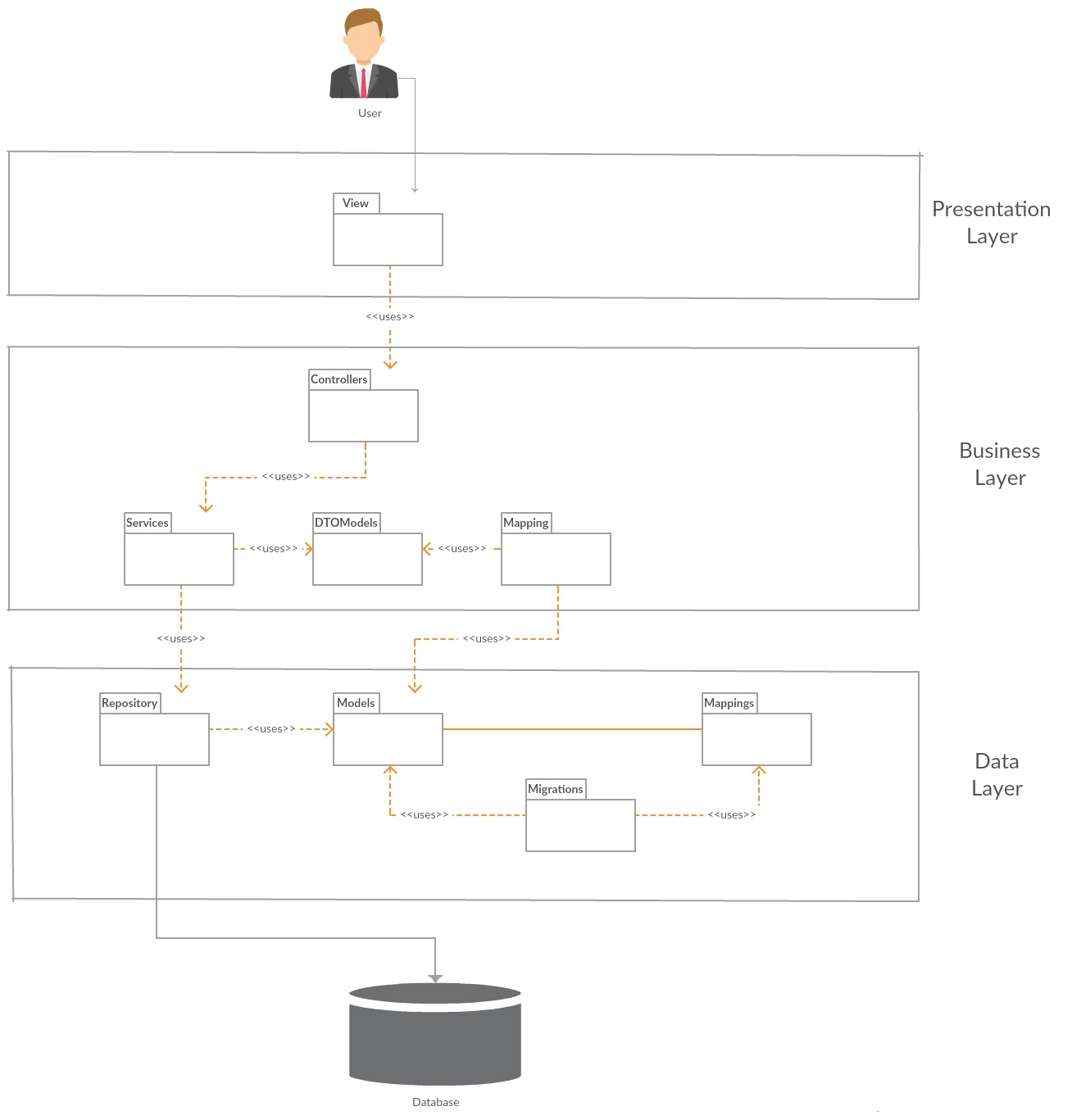
Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data ,and pass that information up to the presentation layer.

**

The layered architecture pattern is a solid general-purpose pattern, making it a good starting point for most applications, particularly when you are not sure what architecture pattern is best suited for your application.

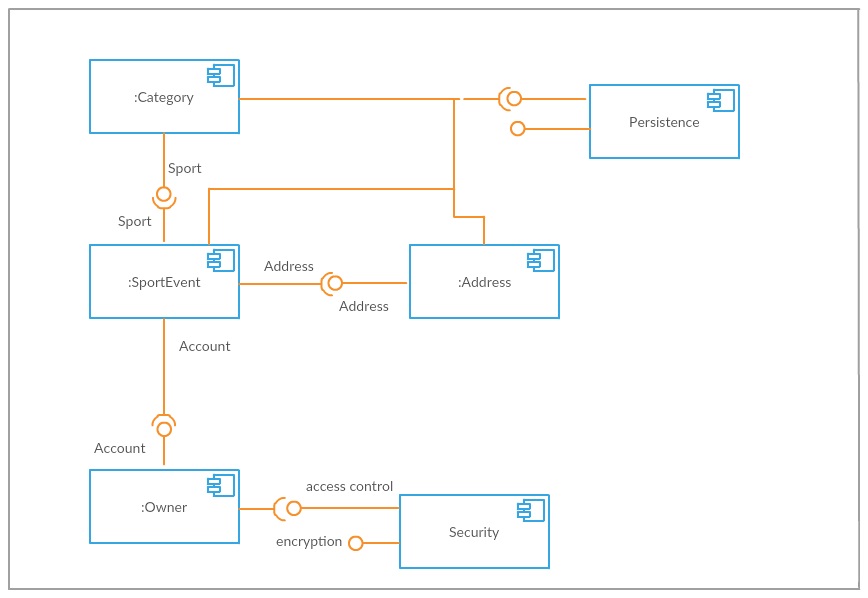
One of the powerful features of the layered architecture pattern is the separation of concerns among components.

## Package Design

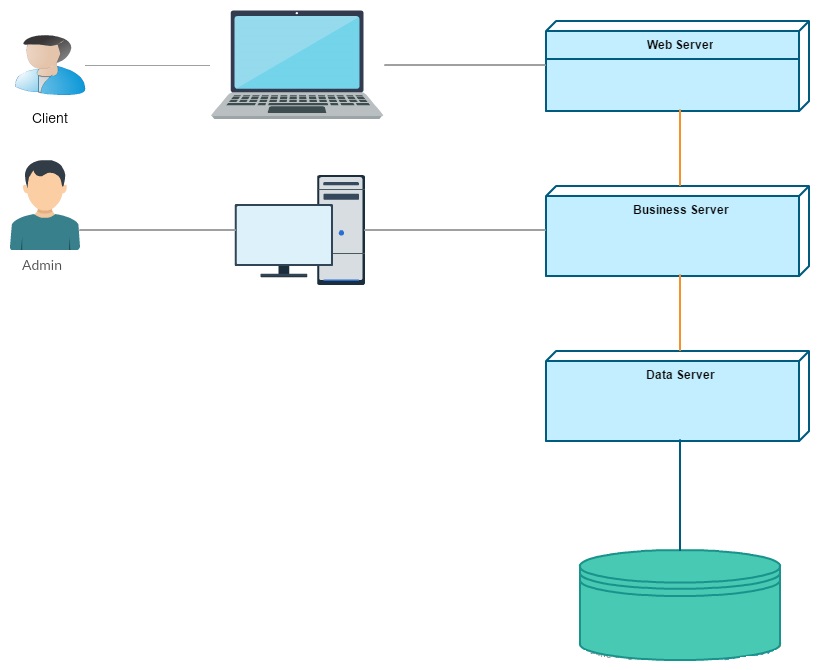


## Component and Deployment Diagrams

Component Diagram



Deployment Diagram

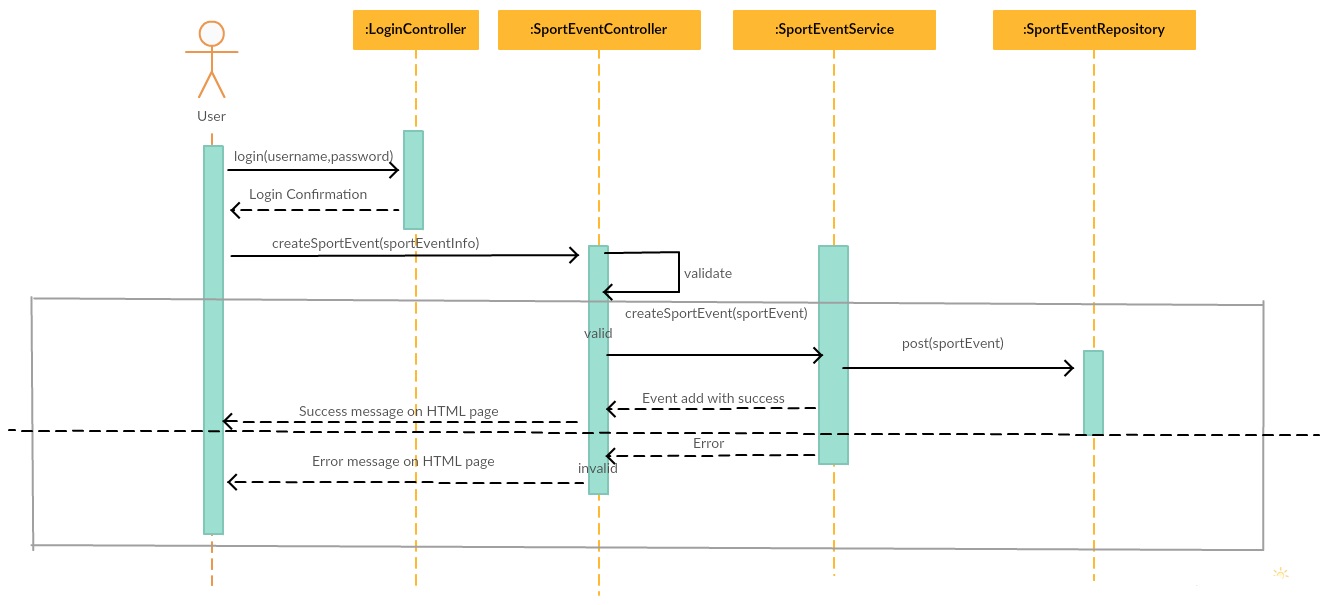


# Elaboration – Iteration 1.2

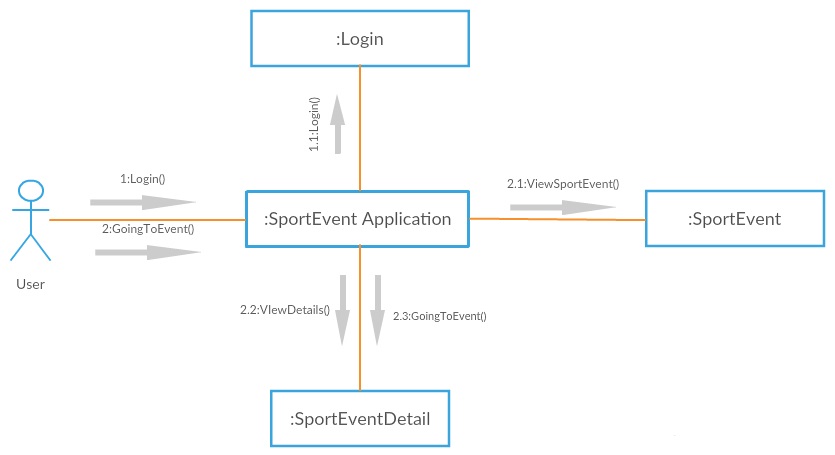
# Design Model

## Dynamic Behavior

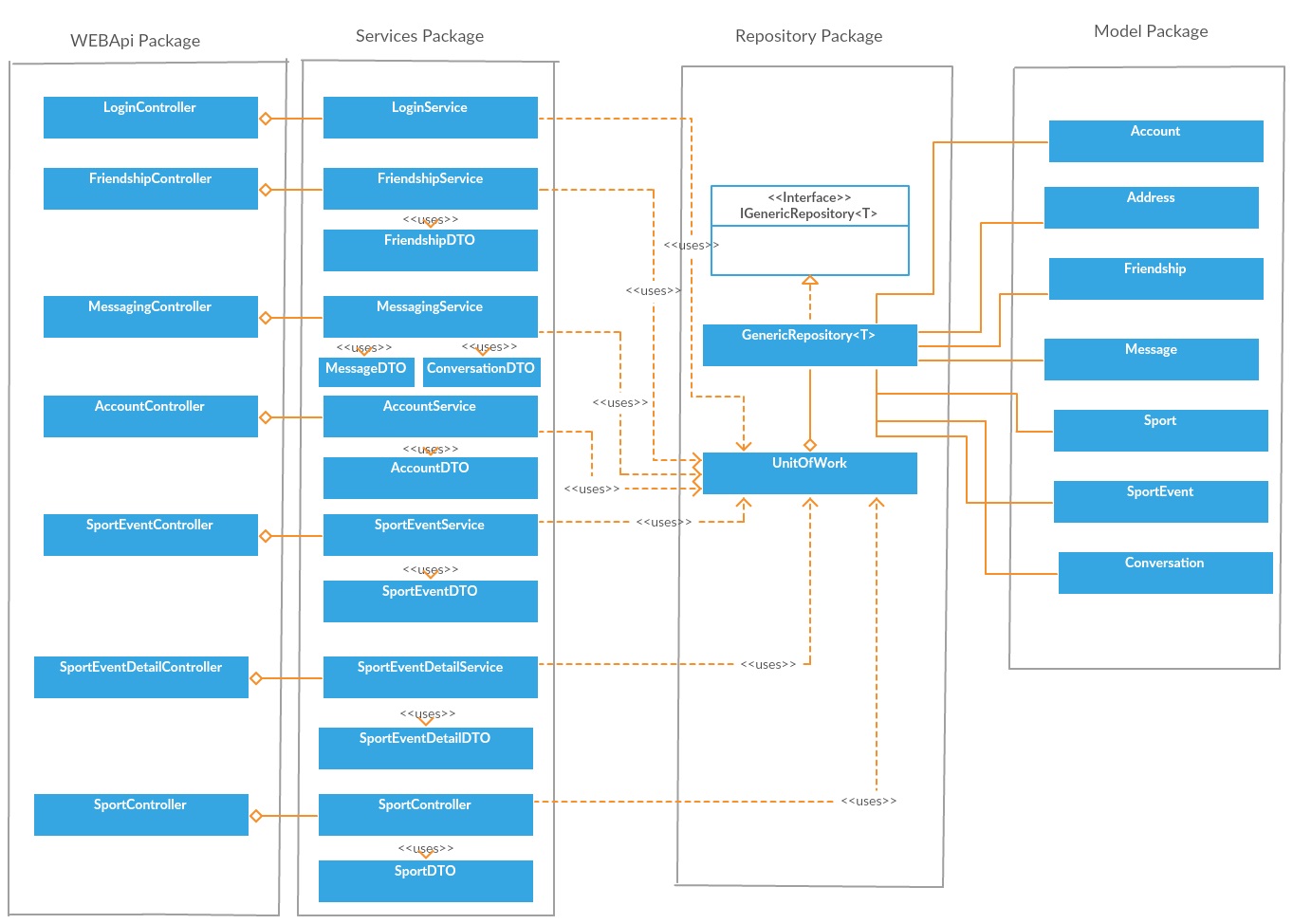
Sequence diagram for creating a sport event.



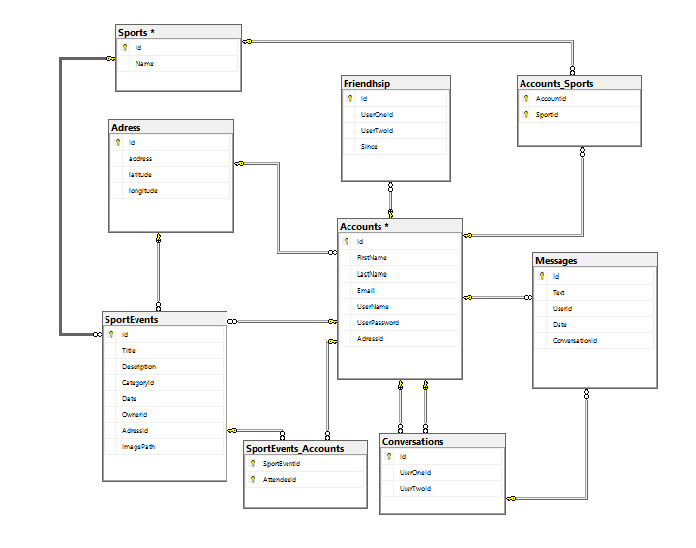
Component diagram for going to a sport event.



## Class Design



# Data Model



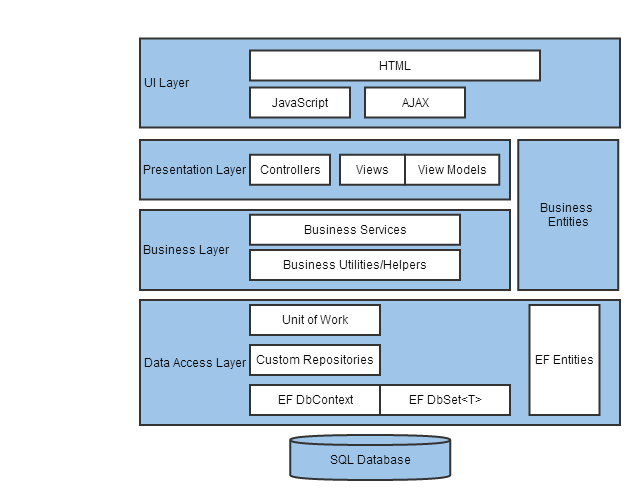
# Unit Testing

Each method from repository( Add, Edit, Delete, GetById, GetAll, FindBy) was tested for all entities from database. Each test case is independent from the others. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. The result are verified with assert instruction.

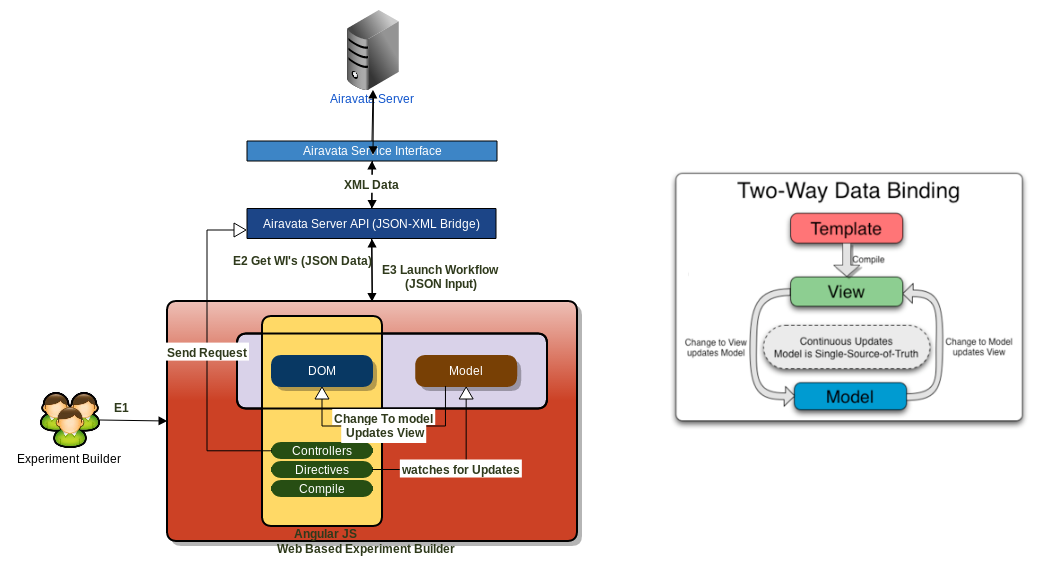
# Elaboration – Iteration 2

# Architectural Design Refinement

For the backend side, it was preserved the layered architecture, where application is organized into horizontal layers and each layer performs a specific role within the application. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request.



For the fronted side, Angular 2 framework was used. The architectural style used by angular is presented in the figure below.

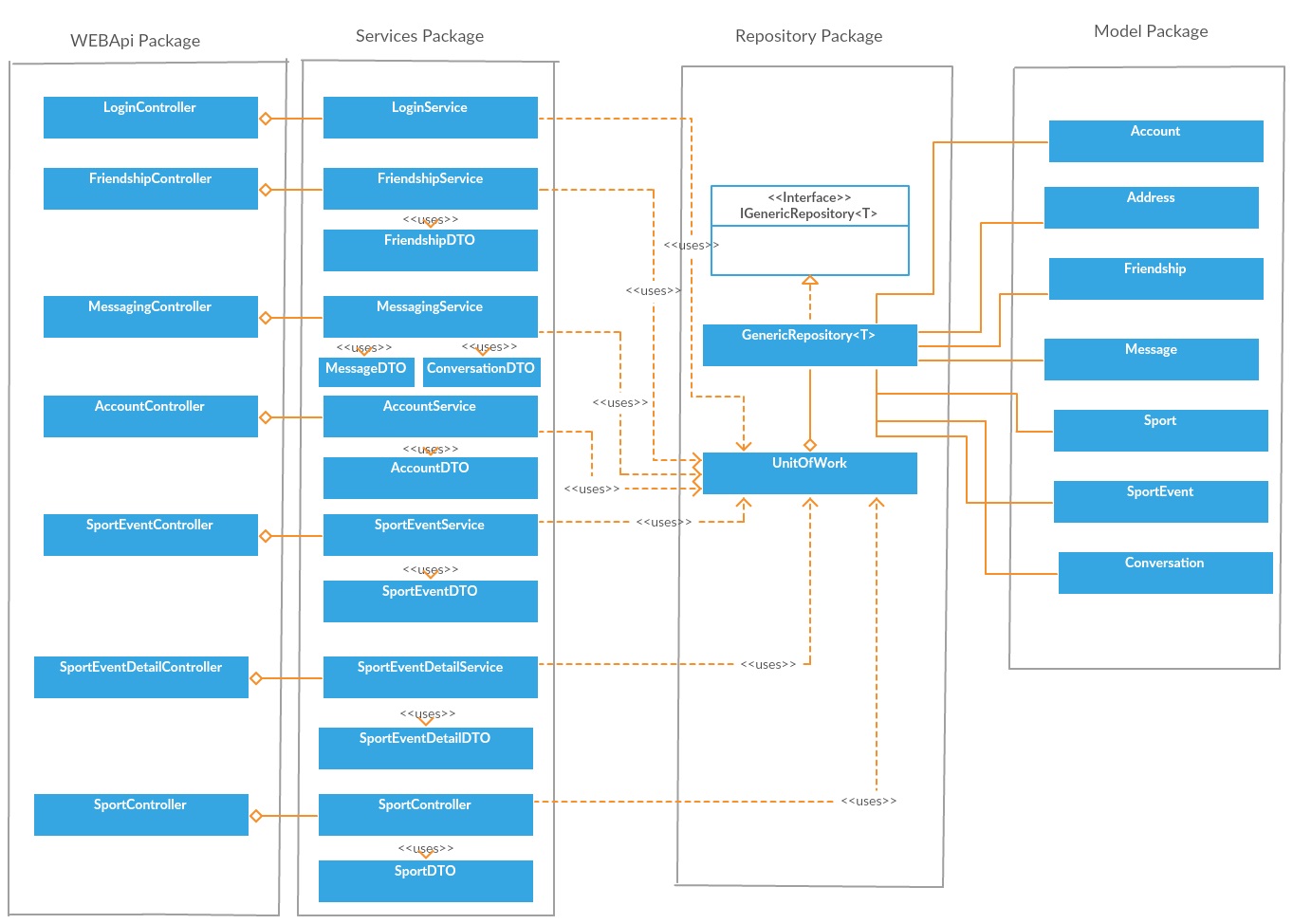
**

architectural style of Angular 2 framework

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

The class diagram has not undergone major changes, because the object oriented principles like SOLID and GRASP were applied at the first iteration .

****.

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

After all small modules was tested with unit testing, it is time to combine them, ant to test the whole application. It were created 4 classes for test ( AccountsTest, SportEventTest, FriendshipTest and MessengerTest). In all these classes were tested the main functionalities from AccountService, SportEventService, FriendshipService and MessengerService. The method that were tested in AccountsTest are :

1.SignUpAndLogin() – was created a new AccountDTO after that it was added to DB, and the login method from Account service should return the Account , and with assert is verified his name, with the name of user which was added.  
 2. EditAccount() – it is the same as the first method, but after the user was added, the last name and address fields were modified, and was executed an edit method. With assert method is check if the fields were modified.

3. AddAccountWithTheSameUserName() – after was added an user, it is created another user with the same username as the first. The method of addAccount should return null, and user will not be added.

The method that were tested in SportEventTest are :

1.AddSportEvent – a new Sport Event was created and was added in the database, after that it is verified if that user exist in database.

2.UpdateSportEvent – is the same as the first method, but after that the title field was modified and was executed an edit method. With assers is check if title is the new title.

3.GoingSportEvent – after a sport event was created, it was created an user and with method goingSportEvent(sportEventId,userId) the user was added to the attendees of that sport event. After that was checked if that user exist in the list of all users which are goind to that event.

4.NotGoingSportEvent- is the same as goingSportEvent, but after that the notGoingSportEvent is executed and it is checked if that user doesn’t exist in the attendees list.

The method which was tested in FriendshipTest is AddRemoveFriend – Firstly, were created two users which were added in the database. The addFriend method was used from FriendshipService in order to create friendship relationship. With method GetAllFriendship, were optained all friends of user1. With assert it was checked if the user2 is in that list. After that it was removed from friends, and this time it was checked if it doesn’t exist in that list.

The method which was tested in MessengerTest is SendMessage – Firstly , were created two users which were added in the database. Was created a conversation between them and after that a message from user1 with the conversationId. The message was posted in the DB, and after that was optained all message from their conversation, and was verified if that message exist in the list of all messages.

# Future improvements

1. Better design
2. Structure and implementation improvements
3. Admin can add sport news
4. Users can add comments and give a note to sport events.
5. Users can share events to/from facebook
6. Encrypted password
7. Profile image for users
8. Share events, news on friends landing page

# Bibliography

Entity Framework - <http://www.entityframeworktutorial.net/code-first/entity-framework-code-first.aspx>

Web API - <https://www.asp.net/web-api>, https://docs.microsoft.com/en-us/aspnet/web-api/

Angular 2 JS - <https://angular.io> , <http://www.angular2.com/> , <https://www.tutorialspoint.com/angular2/>

Design Patterns - <http://www.dofactory.com/net/design-patterns>, <https://sourcemaking.com/design_patterns>

SQL Database - <https://www.w3schools.com/sql/sql_intro.asp>

Designing Web Applications - <https://msdn.microsoft.com/en-us/library/ee658099.aspx>