**Final Project Template for Alpha .NET Students**

**ASP.NET Core MVC Final Project**

This document describes the **final project assignment** for the **ASP.NET Core MVC** course at Telerik Academy.

**Project Description**

You are tasked to create a simple **casino website** where users can **register**, **deposit funds** using credit cards and play one of three available **slot games**.

Design and implement the application using **ASP.NET Core 2.1 MVC**.

The application should have:

* **public part** (accessible without authentication)
* **private part** (available for registered users)
* **administrative part** (available for administrators only)

**Note:** The document contains mocks that are used for visualizing the described functionality. Even though the team might choose to implement a similar design, the mocks should not be interpreted as design requirements and constraints.

**Public Part**

The **public part** of your projects should be **visible without authentication**. The public part consists of the **home page**, **login page** and **register page**.

**Private Part (Users only)**

**Registered users** should have private part in the web application accessible after **successful login**.

The private part of the application consists of:

* Game pages
* Deposit page
* Profile page

**Administration Part**

**System administrators** should have administrative access to the system.

The admin part of the application consists of:

* Transaction history page
* User administration page

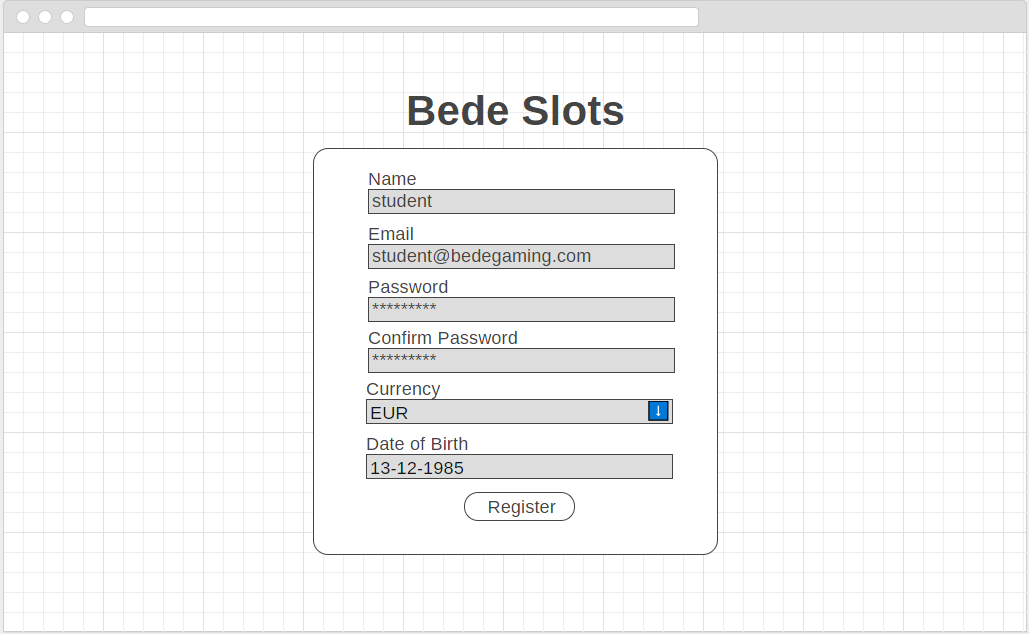
**General Requirements**

Your Web application should use the following technologies, frameworks and development techniques:

* Use **ASP.NET Core 2.1 MVC** and **Visual Studio 2017**
* You should use **Razor** template engine for generating the UI
  + You may use any JavaScript library of your choice
  + Use **sections** and **partial views**
  + Use **tag helpers**
* Use **MS SQL Server** as database back-end
  + Use **Entity Framework 2.1** to access your database
  + Using **repositories and/or service layer** is a must
* Use at least **2 areas** in your project (e.g. for administration)
* Create **tables with data** with **server-side paging and sorting** for a model entity
  + You can use Kendo UI grid, jqGrid, DataTables or any other library or generate your own HTML tables
* Create beautiful and responsive UI
  + You may use **Bootstrap** or **Materialize**
  + You may change the standard theme and modify it to apply own web design and visual styles
* Use the standard **ASP.NET Identity System** for managing users and roles
  + Your registered users should have at least one of the two roles: **user** and **administrator**
* Use **AJAX form** communication in some parts of your application
* Use **caching** of data where it makes sense (e.g. starting page)
* Apply **error handling** and **data validation** to avoid crashes when invalid data is entered (both client-side and server-side)
* Prevent yourself from **security** holes (XSS, XSRF, Parameter Tampering, etc.)
  + Handle correctly the **special HTML characters** and tags like <script>, <br />, etc.
* Create **unit tests** for your "business" functionality following the best practices for writing unit tests (**at least 80% code coverage**)
* Use **Dependency Inversion** principle and **Dependency Injection** technique following the best practices (optional)
* Integrate your app with a **Continuous Integration server** (Jenkins, AppVeyor or other) - configure your unit tests to run on each commit to your master branch
* Use GitHub and take advantage of the **branches** for writing your features.
* **Documentation** of the project and project architecture (as .md file, including screenshots)

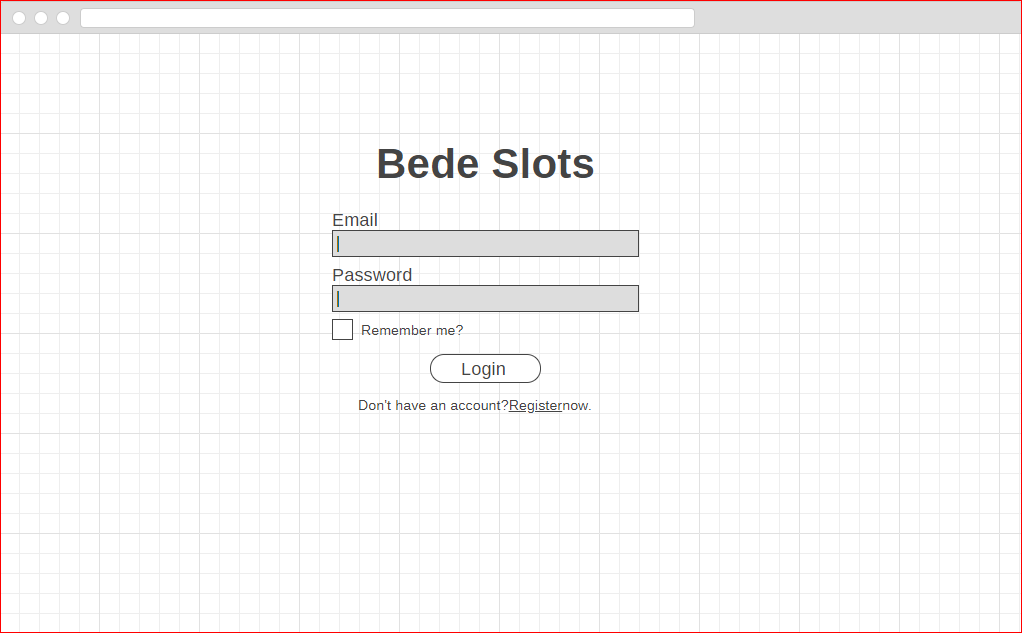
# Public part

## Registration (public)



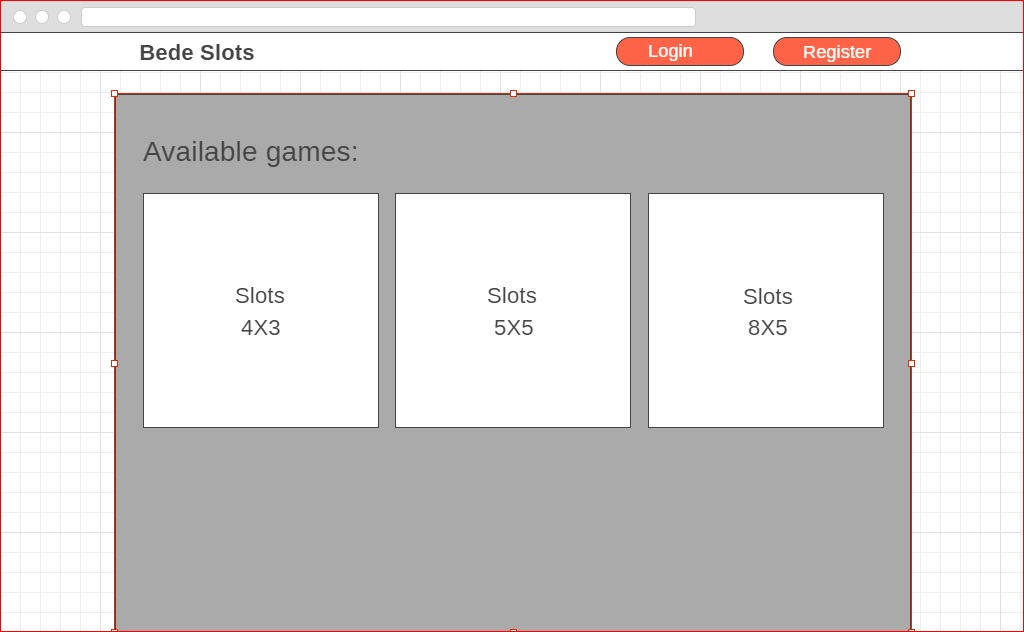
* Any user should be able to register and create new non-admin account.
* The registration should have the following fields (all required): **email, name, date of birth, preferred currency, password, confirm password**
* The currencies should be selected via a dropdown from a predefined list of 4 currencies - EUR, USD, BGN, GBP
* Date of birth selector can take any form - text field, three separate input fields for day, month and year or a calendar selector
* Appropriate validation must be present for all fields

## Login (public)



* The user should be able to login when correct email/password combination is provided.

## Home page (public)



* The home page should be available publicly .
* When user is not logged-in, the top navbar should contain links to the login and register pages
* The homepage should display links to all 3 available games

# Private area

The private part of the website should be available only to logged-in users. If a anonymous user tries to access any of these pages, they should be redirected to the login page.

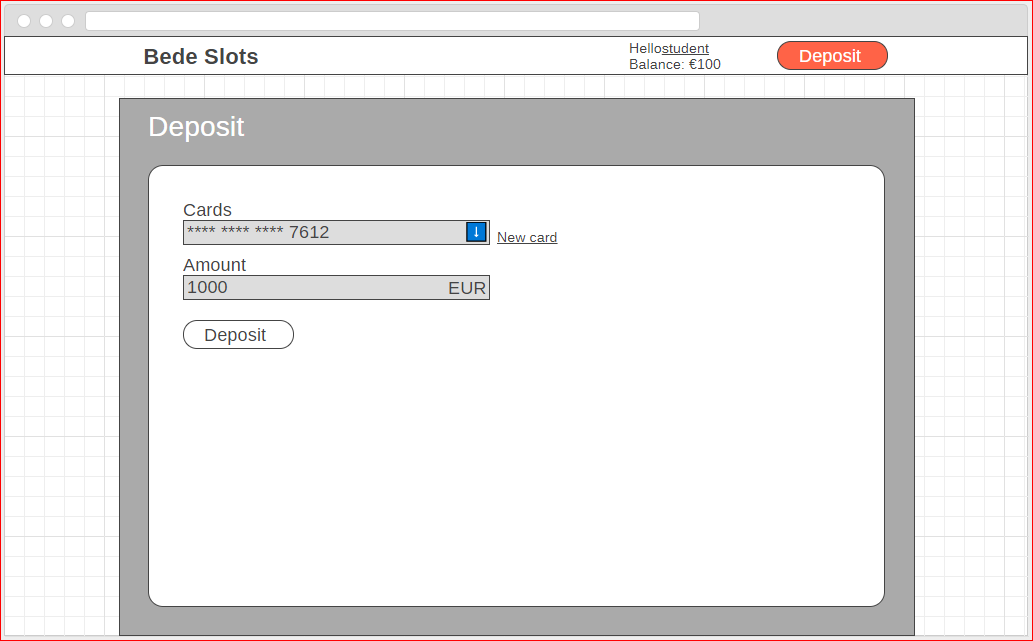
It should have a top navbar on all pages that displays the following information:

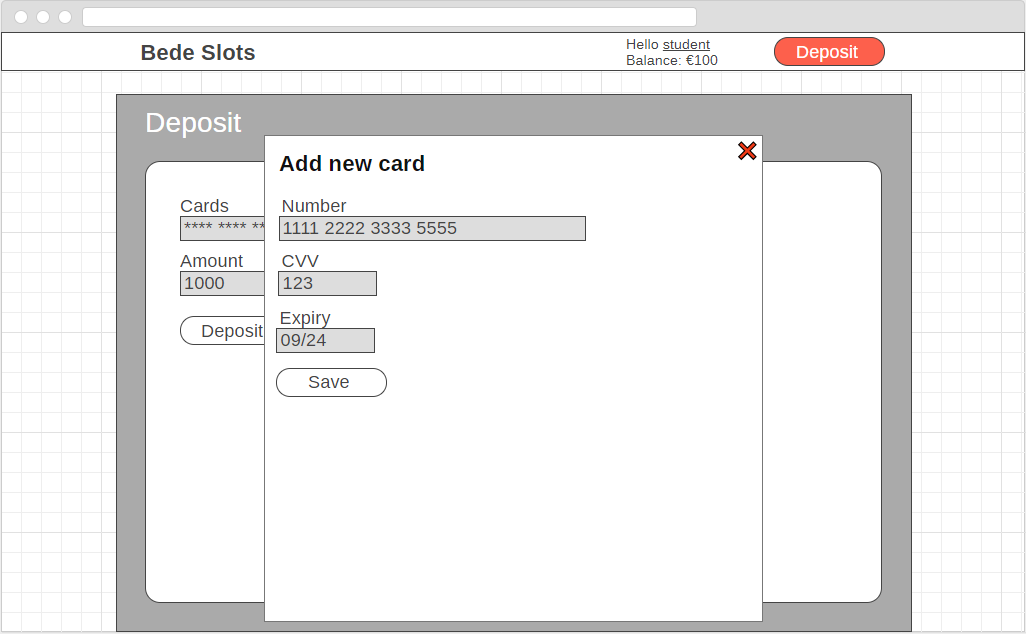
* The name of the currently logged-in user. The name should also function as a link to the profile page of the user. You can use the default user profile page.
* The current balance of the logged-in user in their preferred currency (currency should not be editable).
* A link that points to the **Deposit** page
* Admin users should also see links to the Transactions and User management pages

## Home page (private)

* The only difference in the home page when the user is logged-in compared to the public mode is the top navbar

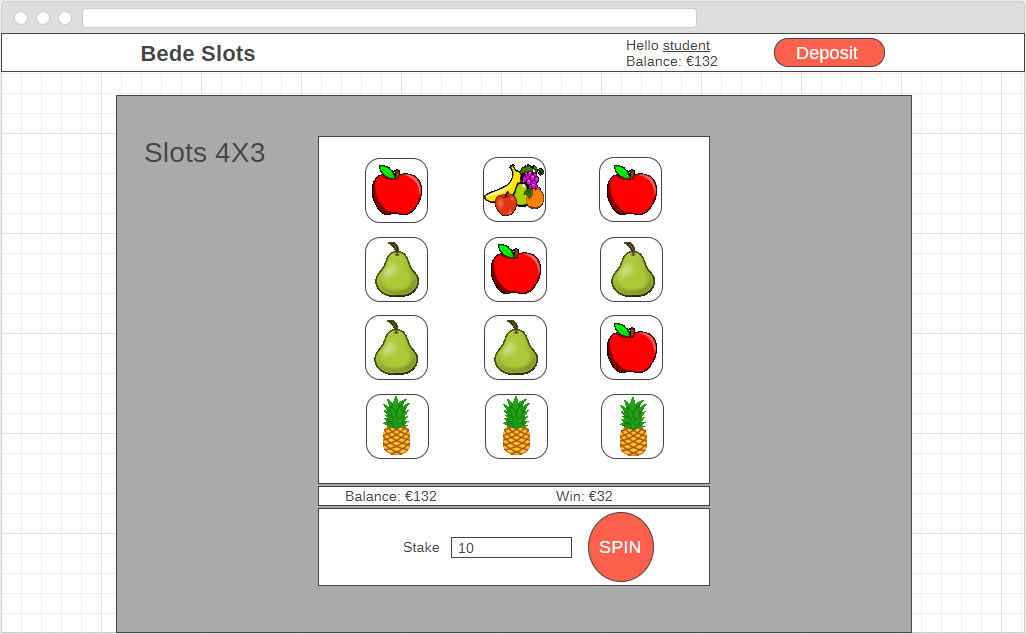
## Deposit page(private)





* On the deposit page the user can deposit funds using any of the already saved credit cards
* The deposit page contains a link to a add a new card. It can be a modal window or a new page
* Credit card details should be validated (example: card number should contain only numbers and be exactly 16 digits)
* Credit card details should be persisted in the database when the save button is clicked
* Returning back to the deposit page - the user chooses one of the cards and inputs the amount of cash they will deposit.
* The currency displayed on the page is the player’s currency selected upon registration. It is not editable.

## Game page(private)



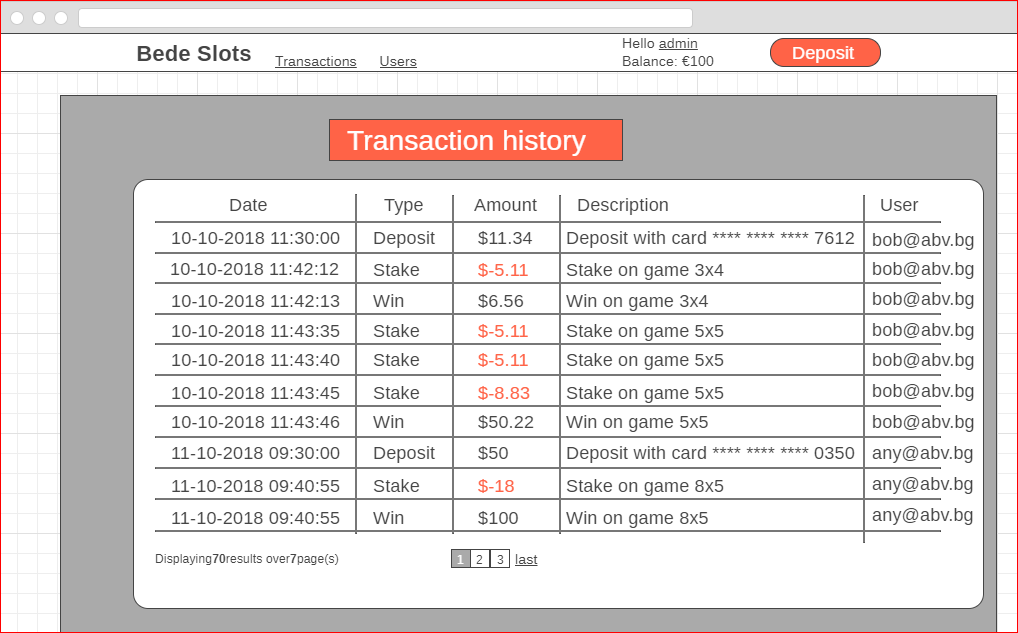
* The game algorithm will be explained in a separate section below.
* When first loading the game - the grid is randomly generated with icons. The initial random grid shouldn’t be scored and no winnings should be awarded.
* The user can choose the amount of cash to stake.
* The user cannot stake more than their current balance.
* The user clicks the spin button, the icons are generated again and result is evaluated. It is not required to create spinning animations, although you are free to do so if you wish.
* Each stake and each win should be persisted in the database as individual transactions. (example: player stakes 5$, which results in total winnings of 3.5$. This should result in two separate transactions - Stake 5$ and Win 3.5$)

# Admin area

## User management

* Admins should be able to assign admin role to users
* You are free to create any suitable UI for this page.

## Transaction history



* Admins should be able to see all deposits, stakes and wins of all users.
* The transactions should be displayed in USD
* Use <https://fixer.io/> or <https://exchangeratesapi.io/> to convert from player’s currency to USD (if necessary). When converting currencies use current rates, there is no need to use historical data.
* Currency rates should be refreshed from the 3rd party API at least once every 24 hours.
* Use server-side pagination and sorting for the table data.
* (Optional) Use server-side filtering for the grid data.

# Game algorithm

* The site should support 3 types of slot games:
  + Game 1: 4 rows and 3 columns
  + Game 2: 5 rows and 5 columns
  + Game 3: 8 rows and 5 columns
* The algorithm is exactly the same for all games.
* The examples below follow Game 1 with 4 rows and 3 columns.
* The games support the following symbols:

|  |  |  |
| --- | --- | --- |
| Symbol | Coefficient | Probability to appear |
| Apple (**A**) | 0.4 | 45% |
| Banana (**B**) | 0.6 | 35% |
| Pineapple (**P**) | 0.8 | 15% |
| Wildcard (**\***) | 0 | 5% |

* The symbols are placed randomly respecting the probability of each item. For example: there is 5% chance that a Wildcard will be placed in a cell and there is 45% chance for an Apple.
* The player will win only if one or more horizontal lines contains only matching symbols. *Wildcard (\*) is a symbol that matches any other symbol (****A****,* ***B*** *or* ***P****).*
* The won amount should be the sum of the coefficients of the symbols on the winning line(s), multiplied by the stake amount.
* If two or more lines are ‘winning’ lines, the coefficients of each of the lines are summed.

## Examples:

BAA // 0

AAA // 0.4 + 0.4 + 0.4 = 1.2 coefficient

A\*B // 0

\*AA // 0 + 0.4 + 0.4 = 0.8 coefficient

Player has opening balance of 200.

Player has staked 10.

Player balance becomes 190.

Winning coefficient is 1.2 + 0.8 = 2 so win is: 10 \* 2 = 20.

The won amount is then added to the current balance of the player 190+ 20 = 210.

***Win calculation examples:***

|  |  |  |  |
| --- | --- | --- | --- |
| Win combinations | | | Calculation of win |
| \* | P | \* | (0 + 0.8 + 0)\*10 = 8 |
| A | A | A | (0.4 + 0.4 + 0.4)\*10 = 12 |
| B | B | B | (0.6 + 0.6 + 0.6)\*10 = 18 |
| P | P | P | (0.8 + 0.8 + 0.8)\*10 = 24 |
| A | B | P | No matching symbols |
| \* | A | B | No matching symbols |

### Partner Evaluations and Criteria

Partner representatives should evaluate the project according to the following criteria:

1. Feature Completeness and Quality
   * Have all requirements been met? Have optional requirements been implemented?
   * Are there any issues with performance, bugs, or malfunction?
2. Code Architecture and Design Quality
   * Loose coupling
   * Modularity
   * Extensibility
3. User Experience and Creativity

The final grade of the project is determined by the following rules.

Each of these 3 criteria are awarded points from 0-5, where 0 is the lowest score and 5 is the highest score:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| Missing | Poor | Fair | Good | Very Good | Excellent |

Then the final project grade is determined by the following formula:

**[Feature Completeness] \* 3 + [Code Architecture Quality] \* 2 + [User Experience and Creativity] \* 1.**

Maximum number of points is 5\*3 + 5\*2 + 5 = 30.

This final score should be awarded by partner representative on **Dec 18**