# ASP.NET MVC Sample Exam (June 2015) – Bookmarks

Design and implement a **Web based bookmark management system**. Use C#, ASP.NET MVC, Entity Framework and SQL Server. Implement the front-end with HTML, CSS and Bootstrap.

* The system holds **users** that have **bookmarks**, organized in **categories**.
  + **Users** have **email** address (used also as username) and **password**.
  + **Categories** have **name** (non-empty, up to 100 characters). Categories are shared between all users.
  + **Bookmarks** have **title**, **URL**, **category** and optional **description**. Bookmarks are personal. Users can access all bookmarks and can update only their bookmarks. URL and title should be non-empty, up to 200 characters. Bookmarks have **votes**. A user can vote for certain bookmark several times.
  + Bookmarks can have list of comments. **Comments** have **author** (existing user), **text** and **bookmark**.
* **Anonymous** users can **register**, **login** and view all bookmarks.
* **Logged-in users** can **view** bookmark details page.
* **Logged-in users** can **create** and **edit** their own **bookmarks**.
* **Logged-in users** can **add comments** and **vote** for bookmarks.
* **Logged-in users** can **logout**.
* **Administrator users** can create / edit / delete **categories**. Non-empty categories cannot be deleted.
* **Administrator users** can delete bookmarks.
* All forms should implement **data validation**. Use client-side validation with AJAX and server-side validation.
* Display a **notification message** after each successful operation that changes the database.

## Design the Database

Design **entity classes** and create a **database** to hold the **users**, **categories**, **bookmarks, comments** and **votes** with Entity Framework **code first**. Put your data layer in separate Visual Studio project (class library).

The data layer should be implemented with some abstraction:

* You can use repository pattern and unit of work.
* You can use repositories and data service layer with database queries.
* You can use only data services layer on the top of the EF context.

10 score

Use **automatic migrations**. At application start-up, **fill sample data** in the database if it is empty: a few **users**, a few **categories** and a few **bookmarks** with a few **votes** and a few **comments**.

10 score

## Implement Home Page, Users, Login and Logout

Implement user **registration**, **login** and **logout**.

* **User registration** should create a new user account and display "**User registered**" **notification message**. In case of error, display an error message. **Validate** the registration form. **Allow simple passwords** (e.g. "123").
* **User registration form** should have **username**, **email** and **password** fields (all mandatory).

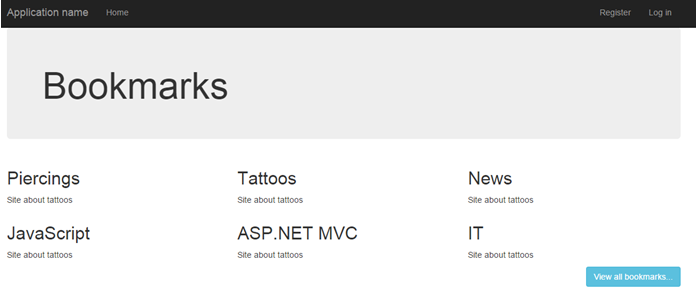
8 score

* **User login** should login an existing user by **username** and **password**. After login, display user's home page holding **top 6 bookmarks** by votes count. In case of problem, display an error message. **Validate** the login form. **Allow simple passwords** (e.g. "123").

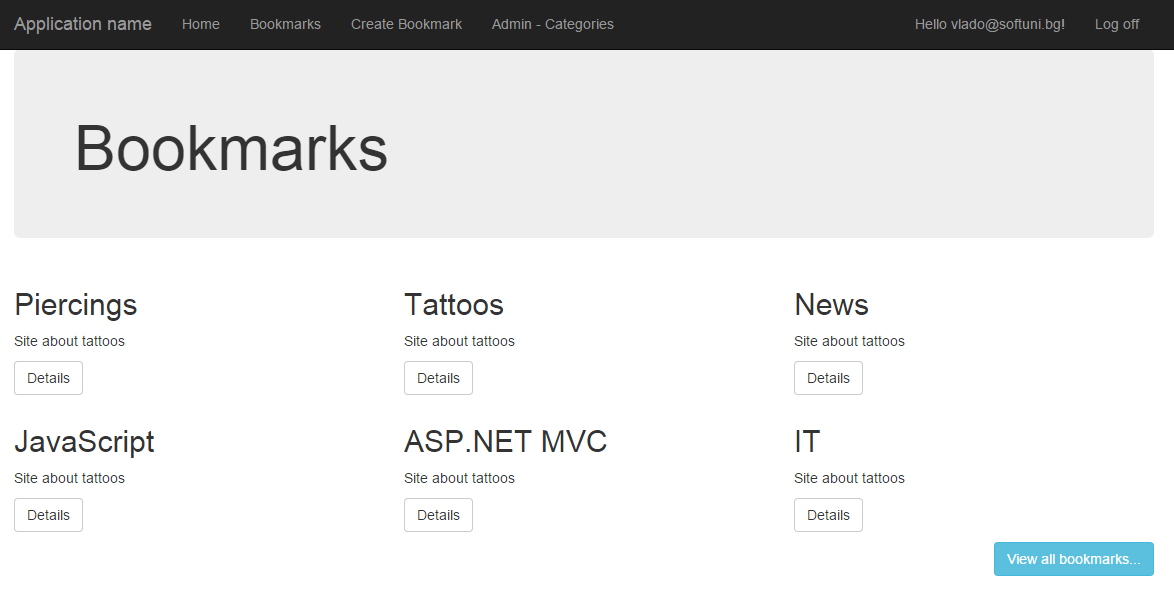
4 score

The user's home page should display **top 6 bookmarks** ordered by votes and should have **button [View all bookmarks]**. Display only **title** and **description** for the bookmarks. Logged-in users can see the [Details] button for every bookmark.

* Anonymous users can **view the top 6 bookmarks** at the home page and can view the "**All bookmarks**" page:



* After registration and login, change the **navigation menu** to hold links to **view the bookmarks**, **create bookmark** and a **logout link**. In case the logged-in user has "**Administrator**" role, display the "**Admin Categories**" link as well.



3 score

* **Prevent anonymous access** to the pages that require login (e.g. view bookmarks details page or create bookmark page). Prevent the non-administrator users from accessing the administrator page.

3 score

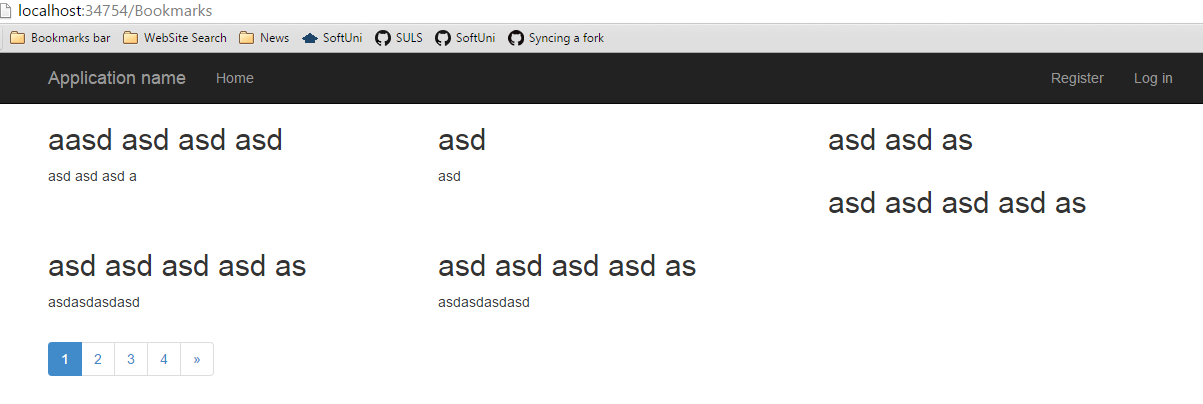
* **User logout** should invalidate user's session and redirect to the application home page.

2 score

## Implement Bookmark Pages

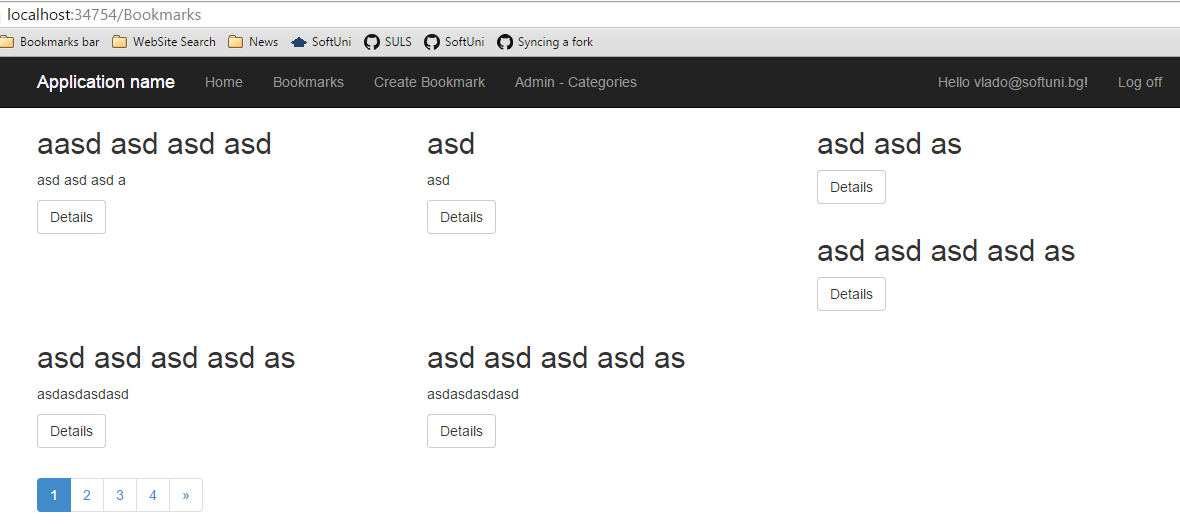
Implement a page to **view all bookmarks**. The page should display all bookmarks (eventually with paging).

* The **"All bookmarks" page for anonymous users** may look like this:



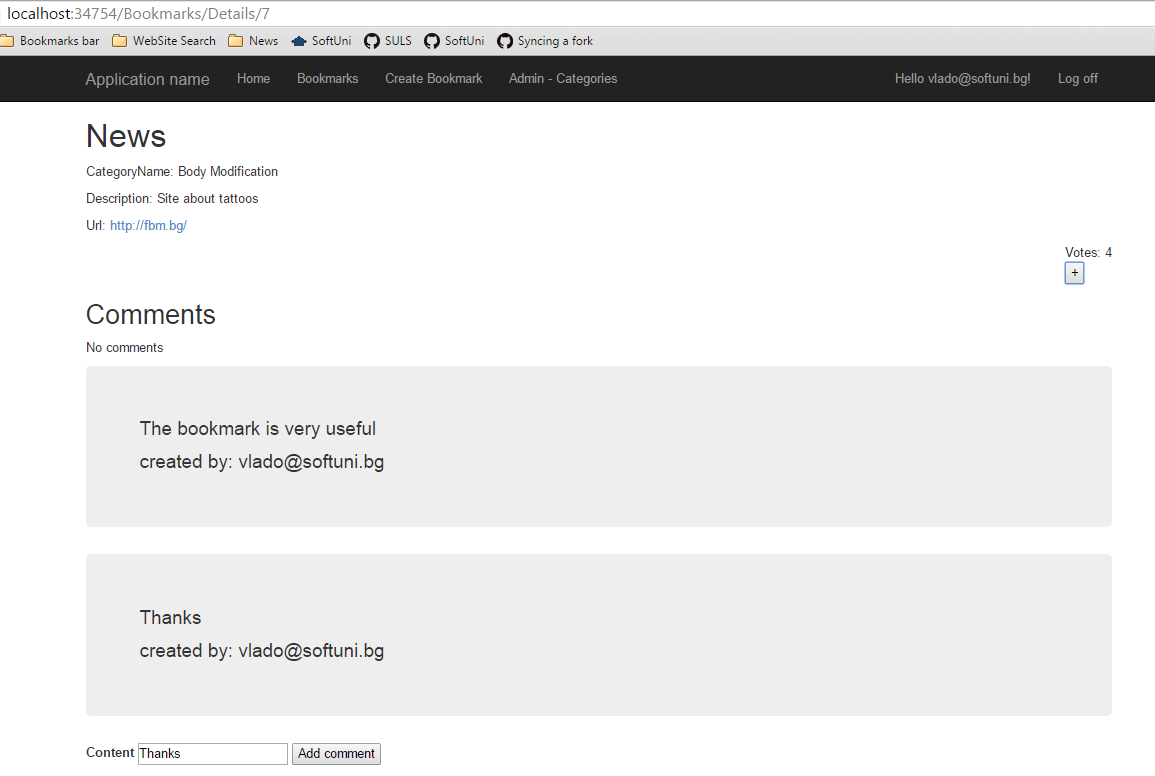
5 score

* The **"All bookmarks" page for logged-in users** may look like this (each bookmark has **[Details] button**):



2 score

* Implement the **"Bookmark details" page**. It should be available for logged-in users only. The bookmarks details page should display **all information about the bookmark**, its **comments** and **votes**:



10 score

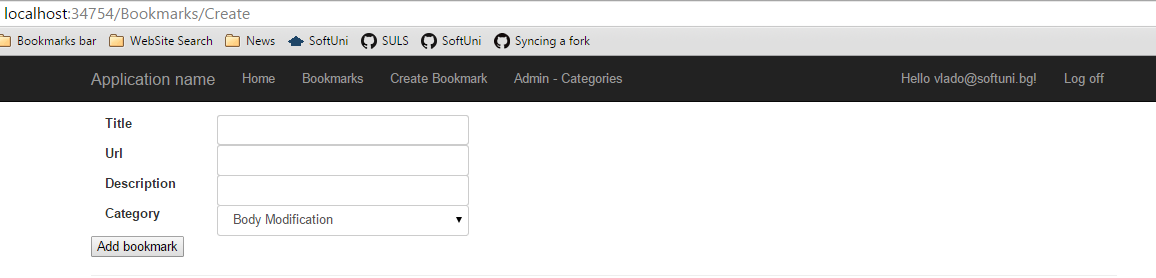
* Implement **adding comment** with AJAX. The newly added comments should be displayed last, just after all previous comments. In case of empty comment form, display a validation error message.

10 score

* Implement **voting** for the bookmark with AJAX. The voting button **[+]** can be clicked many times, and each time, it adds a vote for the selected bookmark by the currently logged-in user.

8 score

* Implement **"Create bookmark" page**:



10 score

* **Validate** the "**create bookmark**" form and display an error message just after each invalid form control.

5 score

## Implement Categories Administration

Implement **CRUD operations for categories**. The page should be accessible **only for administrators**.

* **Register an administrator** user when seeding the database manually for testing purposes.
* Display an **error message** if the user tries to **delete an empty category**.
* Display **notification message** after successful database change.
* You can use controls / wrappers like KendoUI, IgniteUI, MvcControlsToolkit, DevExpress or create categories administration manually.

20 score

## Implement Paging for the Bookmarks

Implement **paging** when listing the **bookmarks**. Use page size 3 to simplify testing. Display the current page, the total number of pages + [First] + [Previous] + [Next] + [Last] buttons.

10 score

## Project Infrastructure Bonus

Bonus points code quality / good application structure / additional effort.

* Bonus points for implementing **separate data layer**.
* Bonus points for using **dependency inversion**.
* Bonus points for using **AutoMapper**.
* Bonus points for using **services** for different jobs.

5 score