|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Пояснительная записка** | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| к курсовой работе | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| на тему | проектирование и программная реализация системы, | | | | | | | | | | | | | | | | |
| моделирующей процессы создания и взаимодействия группы объектов | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| по дисциплине | | **Объектно-ориентированное программирование** | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Выполнил: студент | | | | | II | | курса, группы: | | | | | **ИС/б-22-о** | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Направления подготовки (специальности) | | | | | | | | | | | | 09.03.02 | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Информационные системы и технологии | | | | | | | | | | | | | | | | | |
| (код и наименование направления подготовки (специальности)) | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| профиль (специализация) | | | | | | | |  | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| **Горбенко Кирилла Николаевича** | | | | | | | | | | | | | | | | | |
| (**фамилия, имя, отчество студента**) | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Дата допуска к защите « | | | | | |  | | | » |  | | | | | 20 | 19 | г. |
|  | | | | | |  | | |  |  | | | | |  |  |  |
|  | | | | | | | | | | | | | | | | | |
| Руководитель | | | |  | | | | | | | | |  |  | | | |
|  | | | (подпись) | | | | | | | | | |  | (инициалы, фамилия) | | | |
|  | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | 19 | г. | | | | | |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ **АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** УЧРЕЖДЕНИЕ **ВЫСШЕГО ОБРАЗОВАНИЯ**

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт Информационных технологий и управления в технических системах

Кафедра «Информационные системы»

Направление подготовки/специальность 09.03.02 – Информационные системы и технологии (код и название)

Профиль/специализация

Курс 2 Группа ИС/б-22о Семестр 4

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ СТУДЕНТА**

Горбенко Кирилл Николаевич

(фамилия, имя, отчество)

**1. Тема проекта (работы)** Проектирование и программная реализация системы, моделирующей процессы создания и взаимодействия группы объектов

**2. Сроки сдачи** студентом законченного проекта (работы) 29.05.2019г.

**3. Исходные данные** к проекту (работе) Исходными данными к проекту является техническое задание, предоставляющее спецификацию требований к проектированию программного продукта

**4. Содержание пояснительной записки** (перечень вопросов, подлежащих разработке)

1. Титульный лист.
2. Техническое задание.
3. Аннотация.
4. Содержание.
5. Перечень сокращенных и условных обозначений (при необходимости).
6. Введение.
7. Список исполнителей (при необходимости).
8. Постановка задачи.
9. Проектное решение.
10. Программная реализация.
11. Выводы.
12. Перечень ссылок.

**5. Перечень графического материала** (с точным указанием обязательных чертежей)

1. Диаграмма переходов состояний.
2. Диаграмма потоков классов и действий.

**6. Дата выдачи задания** 06.02.2019г.

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |
| --- | --- |
| Номер недели | Содержание работы |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Студент Горбенко Кирилл Николаевич

(подпись)

Руководитель проекта (работы) Тимофеев И.С.

(фамилия, инициалы, должность) (подпись)

« 06 » февраля 2019г.

1. АННОТАЦИЯ

Пояснительная записка к курсовому проектированию по дисциплине «Объектно-ориентированное программирование» для студентов направления 09.03.02. – «Информационные системы и технологии».

Пояснительная записка содержит описание процесса проектирования и кодирования курсового проекта, состоящего в разработке программного продукта в пределах объектной парадигмы. Разрабатываемый программный продукт является веб-приложением, предназначенным для отслеживания времени, уделяемого пользователем на его цели и задачи для достижения большей эффективности в работе.

**Содержание**

[1 ПОСТАНОВКА ЗАДАЧИ 8](#_Toc9984205)

[1.1 Цель разработки 8](#_Toc9984206)

[1.2 Техническое задание на разработку программного продукта 8](#_Toc9984207)

[1.2.1 Введение 8](#_Toc9984208)

[1.2.2 Требования к программе 8](#_Toc9984209)

[1.2.3 Технологические требования 8](#_Toc9984210)

[1.2.4 Структура и описание сайта приложения 9](#_Toc9984211)

[1.2.5 Требования к надежности 9](#_Toc9984212)

[1.2.6 Условия эксплуатации 10](#_Toc9984213)

[2 ПРОЕКТНОЕ РЕШЕНИЕ 12](#_Toc9984214)

[2.1 Абстрагирование и выделение объектов 12](#_Toc9984215)

[2.2 Представим архитектуру приложения. Паттерн MVC и модель Клиент – Сервер 13](#_Toc9984216)

[2.2.1 Что такое паттерн MVC 14](#_Toc9984217)

[2.2.2 Жизненный цикл программы 16](#_Toc9984218)

[2.2.3 Диаграмма состояний 16](#_Toc9984219)

[2.2.4 Диаграммы потоков данных 17](#_Toc9984220)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 19](#_Toc9984221)

[3.1 Выбор языка программирования 19](#_Toc9984222)

[3.1.1 Структурная схема программы 19](#_Toc9984223)

[3.2 Реализация основных классов и их методов 20](#_Toc9984224)

[3.2.1 Пользователь 20](#_Toc9984225)

[3.2.2 Задачи 21](#_Toc9984226)

[3.2.3 Цели пользователя 22](#_Toc9984227)

[3.2.4 Хранилище целей пользователя 22](#_Toc9984228)

[3.2.5 Таймер 23](#_Toc9984229)

[3.2.6 Хранилище таймеров 23](#_Toc9984230)

[3.2.7 Контроллеры 24](#_Toc9984231)

[3.2.8 Запуск таймеров на клиенте 26](#_Toc9984232)

[3.3 Дополнительно использованные функции 26](#_Toc9984233)

[3.3.1 Система контроля версий 26](#_Toc9984234)

[3.3.2 Внедрение зависимостей (Dependency injection) 27](#_Toc9984235)

[3.3.3 ORM (Object-Relational mapping) 28](#_Toc9984236)

[3.3.4 No-SQL база данных 29](#_Toc9984237)

[3.4 Интерфейс пользователя 29](#_Toc9984238)

[3.5 Критерий качества программной системы 34](#_Toc9984239)

ВВЕДЕНИЕ

В ходе курсового проектирования ведется разработка программы под названием «Objective Time Tracker» на основании технического задания. Техническое задание также разрабатывается в пределах курсового проектирования и согласовывается с преподавателем. Разработка ведется для организации «Севастопольский государственный университет». Дата получения задания на разработку технического задания – 06.02.2019 г. Дата передачи готового программного продукта организации-заказчику – 29.05.2019 г.

Разработка данной программы является учебным проектом, направленным на практическое применение знаний и умений, полученных в ходе курса «Объектно-ориентированное программирование». Актуальность данного курсового проектирования обусловлена непрерывно изменяющейся ситуацией на рынке информационных технологий: постоянное обновление имеющихся способов разработки ПО, создание новых фреймворков, появление принципиально новых технологий. Данный курсовой проект предоставляет возможность разобраться в одной или нескольких актуальных технологиях.

**Целью** курсового проектирования является программа, выполняющая функции создания таймеров для целей конкретного пользователя и средства манипуляции этими таймерами через веб-интерфейс.

Для достижения данной цели необходимо: разработать проектное решение архитектуры приложения, настроить серверную часть приложения и реализовать средства коммуникации серверной части приложения с клиентской частью.

* 1. ПОСТАНОВКА ЗАДАЧИ
     1. Цель разработки

Целью разработки является программный продукт, удовлетворяющий требованиям технического задания, представленного в подразделе 1.2.

* + 1. Техническое задание на разработку программного продукта
       1. Введение

Наименование программы: «Objective Time Tracker».

Назначение и область применения: система представляет собой программный комплекс и предназначена для использования в личных целях для увеличения личной производительности работы и увеличения эффективности использования времени. Приложение доступно через веб-сервер.

* + - 1. Требования к программе

К функциональным характеристикам приложения предъявляются следующие требования:

приложение должно обеспечивать функционал аутентификации: возможность зарегистрироваться, войти в систему под определенным аккаунтом, выйти из системы;

приложение должно обеспечивать возможность создания целей для конкретного пользователя и сохранение их состояния;

приложение должно обеспечивать возможность создания привязанных к определенным целям таймеров и сохранение их состояния;

выводить время доступных таймеров на основной экран приложения.

* + - 1. Технологические требования

К технологической стороне приложения предъявляются следующие требования:

сайт приложения должен корректно отображаться на любых типах устройств: настольные компьютеры, смартфоны, планшеты и т.д.;

сайт должен корректно отображаться браузерами Google Chrome, Safari, Mizilla Firefox, Opera, Edge.

обязательна динамическая поддержка действий пользователя (запуск таймеров).

* + - 1. Структура и описание сайта приложения

Сайт приложения должен состоять из двух страниц: главной страницы и страницы аутентификации, состоящей из следующих элементов:

форма ввода данных пользователя, имеющая поля «логин» и «пароль» и кнопку «Sign in»;

кнопка регистрации, открывающая окно регистрации;

кнопка восстановления пароля по адресу электронной почты.

* + - * 1. Содержание главной страницы сайта

При успешном входе пользователя в систему должна открываться главная страница сайта, имеющая следующие элементы:

имя пользователя, который вошел в систему и кнопка выхода из системы;

список целей данного пользователя;

список таймеров данного пользователя;

кнопки запуска и остановки таймеров;

кнопка удаления таймеров.

* + - 1. Требования к надежности

Надежное функционирование программы должно быть обеспечено выполнением двух факторов:

использованием пользователем распространенных веб-браузеров (таких как Google Chrome, Mozilla Firefox, Safari и т.д.)

регулярным выполнением требований ГОСТ 51188-98 «Защита информации».

Время восстановления после отказа не регламентируется поскольку любые отказы программы, которые произойдут на веб-сервере, будут по вине подрядной организации, специализирующейся на хостинге. Время восстановления программы будет зависеть от данной организации и не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

Вследствие некорректных действий оператора отказы программы невозможны.

* + - 1. Условия эксплуатации

Климатических условий эксплуатации к пользователю не предъявляется.

Использование программы операторами в количестве, большем одного пользователя, не предполагается. Для каждого конкретного нового пользователя должен быть создан новый аккаунт внутри системы.

К составу и параметрам технических средств предъявляются требования, предъявляемые современными веб-браузерами:

операционная система Windows не ниже XP с пакетом обновления 2+, или Windows Vista или более поздние; MacOS X 10.6 или более поздние; Ubuntu 10.04+ или Debian 6+;

процессор Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2;

свободное место на диске: 350 Мб;

оперативная память 512 Мб.

Требования к информационным структурам не предъявляются.

Требования к исходным кодам и языкам программирования: возможность исполнение кода на языке JavaScript в браузере.

Специальные требования к данной программе не предъявляются.

Ориентировочная экономическая эффективность не рассчитывается т.к. разрабатываемое ПО не предназначено для коммерческой деятельности, а только для личного использования.

* 1. ПРОЕКТНОЕ РЕШЕНИЕ

Для представления проектного решения выделим сущности, входящие в доменную модель. Доменная модель предметной области – модель, использующаяся в объектно-ориентированной парадигме создания ПО, которая представляет из себя набор абстракций сущностей этой предметной области в виде классов и связи между ними.

* + 1. Абстрагирование и выделение объектов

После оценки предметной области были выделены следующие сущности: пользователь, цель, таймер, репозиторий пользователей, репозиторий таймеров.

Сущность «пользователь» представляет из себя человека, который будет пользоваться программой. Класс пользователя должен содержать следующие атрибуты: логин, адрес электронной почты, пароль, идентификатор пользователя.

Цель – сущность для представления задачи, которую пользователь хочет решать с помощью программы. При этом программе не обязательно знать, в чем состоит эта цель и что требуется для ее решения. Поэтому, для идентификации цели достаточно ее имени. Сущность будет содержать следующие атрибуты: имя, время, выделяемое под эту цель в неделю, количество недель на выполнение цели. Кроме того, цель должна рассчитывать время, выделяемое на нее сегодня, общее оставшееся время и иметь интерфейс для изменения состояния таймера.

Сущность таймер будет использоваться для отслеживания времени, потраченного на цель. Таймер должен ассоциироваться с целью, хранить состояние времени, иметь возможность запускать и останавливать счетчик, сбрасывать его.

Введем сущность, представляющую цели пользователя. Она будет содержать атрибут Id для сохранения в базу данных, Id соответствующего пользователя и коллекцию его целей.

Поскольку для одной цели в системе может существовать лишь один таймер, для обеспечения единственности каждого таймера необходимо ввести Singleton-репозиторий таймеров. Он будет иметь следующие атрибуты: словарь (Dictionary) для связи идентификатора пользователя с его таймерами и метод получения нужных таймеров.

* + 1. Представим архитектуру приложения. Паттерн MVC и модель Клиент – Сервер

Модель клиент-сервер предполагает наличие двух обособленных компонентов – Клиента и Сервера и инфраструктуру связи между ними. Сервер представляет из себя поставщика неких услуг, которые требуются пользователю, представляемого клиентом. Коммуникация между сервером и клиентами осуществляется посредством выбранных заранее протоколов. В веб-разработке будем использовать сетевой протокол http. Модель Клиент-Сервер представлена на рисунке 2.1.

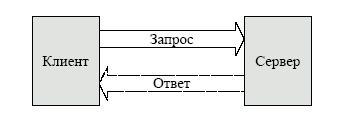


Рисунок 2.1 – Модель Клиент-Сервер

В качестве архитектуры сервера выберем паттерн MVC («Model-View-Controller», «Модель-Представление-Контроллер»), повсеместно использующийся в разработке клиент-серверных приложений. Он позволяет разделять данные приложения, управляющую логику и логику отображения данных пользователю, обеспечивая принцип разделения обязанностей (англ. Separation of concerns).

* + - 1. Что такое паттерн MVC

Model-View-Controller. MVC — это фундаментальный паттерн, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам. Впервые паттерн появился в языке SmallTalk. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес логики, а бизнес логику от данных. Таким образом, в классическом варианте, MVC состоит из трех частей, которые могут изменяться независимо. Рассмотрим их.

**Модель (Model).** Под моделью обычно понимается часть, содержащая в себе функциональную бизнес логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель. Модель обладает следующими характеристиками:

Модель – это бизнес-логика приложения;

Модель не обладает знаниями ни о контроллерах, ни о представлениях;

Для некоторых проектов Модель – это просто слой данных (например, база данных или набор файлов).

**Представление (View).** В обязанности Представления входит отображение данных полученных от Модели, следовательно, представление должно знать о структуре организации данных в Модели. Представление обладает следующими признаками:

в представлении реализуется отображение данных, полученных от модели;

Представление может иметь код (логику), если она относится к изменению данных для отображения.

Представление может обращаться к Модели

**Контроллер (Controller).** Контроллер используется для осуществления связи между Моделью и Представлением как некоторое промежуточное звено. Его обязанность – получать запросы от пользователя на получение данных из модели или их изменение и, осуществив необходимую операцию, вернуть ответ из модели к пользователю, используя для этого Представление. Особенности Контроллера:

Контроллер обладает информацией как о Модели, так и о Представлении;

контроллер отвечает за то, что Представление получит данные в том формате, в котором оно их ожидает.

На рисунке 2.2 изображена схема представления паттерна MVC.

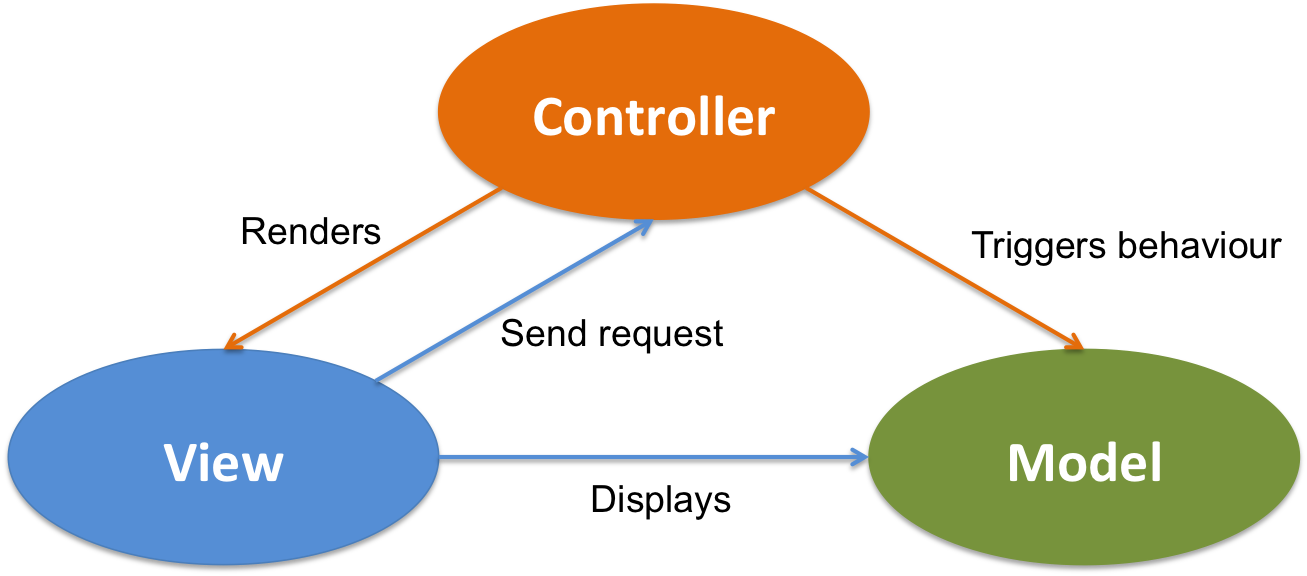


Рисунок 2.2 – Схема взаимодействий элементов в паттерне MVC

Поскольку в MVC часть Model не знает ничего ни о контроллере, ни о представлении, обеспечивается отличная ее переносимость. Таким образом, используя написанную один раз Model-часть, ее можно многократно использовать для различных типов приложений. Также преимуществом является и то, что в таком случае изменения, произведенные над моделью, повлияют на все зависящие от нее проекты.

MVC – архитектура отлично подходит для модульного тестирования приложения.

* + - 1. Жизненный цикл программы

Жизненный цикл программы, соответствуя области применения, в идеальном случае, должен быть бесконечным. То есть программа, развернутая на веб-сервере, будет принимать адресованные ей запросы до тех пор, пока сервер не будет отключен.

* + - 1. Диаграмма состояний

Объекты, передаваемые клиентом на сервер (UserCreateModel, UserLoginModel, ObjectiveCreateModel) будут иметь общий набор состояний. Эти состояния представлены на диаграмме состояний, изображенной на рисунке 2.3.

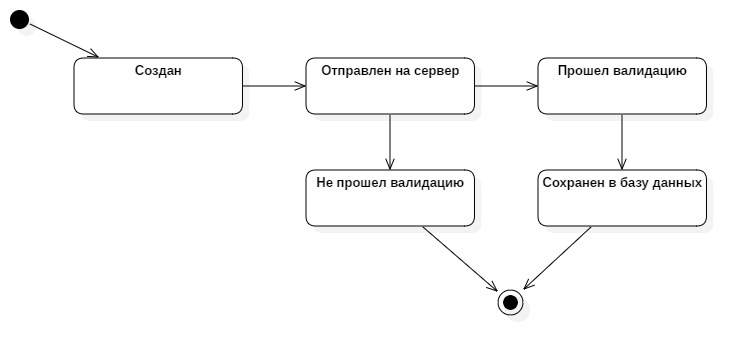


Рисунок 2.3 – Диаграмма состояний объектов UserCreateModel, UserLoginModel, ObjectiveCreateModel

На диаграмме состояний обозначены не внутренние состояния вышесказанных объектов, а их состояние в системе, поскольку их внутреннее состояние не интересно, оно содержит лишь данные. При этом, переходы между состояниями происходят не по вызову функций этих объектов, а по включению или выключению данного объекта хранилища данных приложения.

* + - 1. Диаграммы потоков данных

Для представления потоков данных в приложении опишем потоки данных при запуске и остановке таймера. Диаграммы потоков данных представлены на рисунках 2.4 и 2.5.

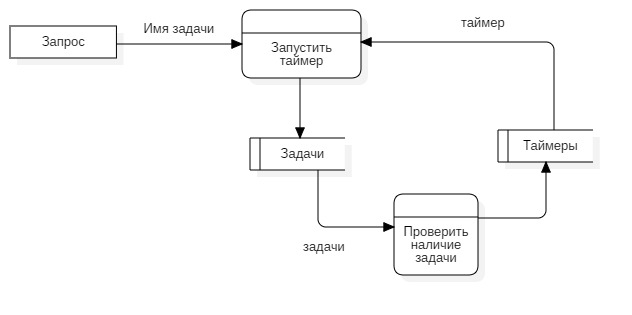


Рисунок 2.4 – Диаграмма потоков данных операции запуска таймера

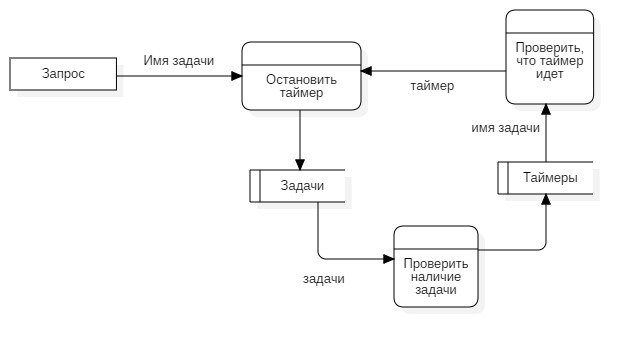


Рисунок 2.5 – Диаграмма потоков данных операции остановки таймера

* 1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ
     1. Выбор языка программирования

При выборе языка программирования основной упор делался на то, в каком языке исполнитель разбирается наиболее хорошо. Для написания курсового проекта была выбрана платформа ASP.NET Core – Open Source фреймворк для создания и развертки веб-приложений. В качестве языка программирования используется C# - мультипарадигменный язык программирования со статической типизацией и встроенным сборщиком мусора. На C# можно разрабатывать клиентские приложения на Windows, приложений клиент-сервер, приложений для платформ IOS и Android, а, с недавнего времени и приложений для Linux и MacOS.

* + - 1. Структурная схема программы

Составим диаграмму классов, представляющую архитектуру приложения на основе паттерна MVC. Она представлена на рисунке 3.1.

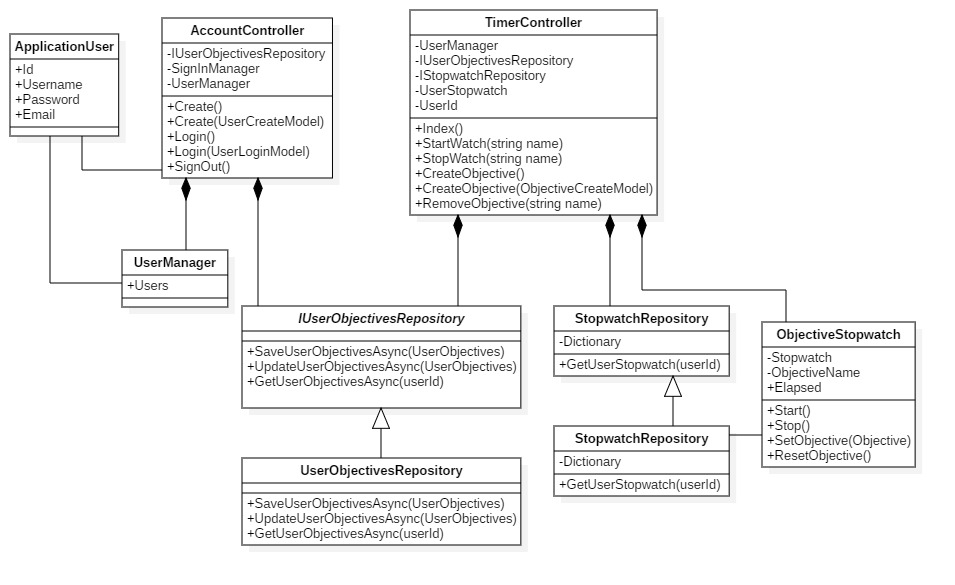


Рисунок 3.1 – Диаграмма классов

На диаграмме изображена структура классов приложения. При этом, в соответствии с паттерном MVC, контроллерами являются классы AccountController и LoginController. Все остальные классы относятся к модели. Представления на диаграмме классов не обозначены, так как в веб-приложении ими являются HTML-документы.

* + 1. Реализация основных классов и их методов

Реализовывать основные классы с учетом включения нескольких фреймворков или библиотек, предоставляющих соответствующую функциональность. Такими являются: Entity Framework для реализации ORM взаимодействия с реляционной базой данных (Object-Relational Mapping), MongoDB для реализации базы данных, не имеющей строгой структуры организации объектов (No-SQL), Identity для реализации аутентификации в приложении пользователей.

* + - 1. Пользователь

Для реализации пользователя так, чтобы использовать его класс для аутентификации, необходимо унаследовать его от встроенного в Identity Framework класс User. Данный класс предоставляет следующие атрибуты:

Id – идентификатор пользователя;

UserName – имя пользователя;

Email – электронная почта;

PasswordHash – захэшированное представление пароля;

PhoneNumber – номер телефона и др.

Поскольку данной функциональности более чем достаточно для осуществления заявленной функциональности, вносить новые атрибуты нет необходимости.

Для осуществления хранения данных пользователей в реляционной базе данных, предоставляемой Entity Framework, создадим класс UserIdentityDbContext, который должен наследовать от класса IdentityDbContext<ApplicationUser> (обобщенный) для организации связи класса с фреймворком. Данный класс будет состоять лишь из конструктора, который принимает настройки базы данных. После реализации данного класса он будет использоваться в качестве параметра-типа для классов UserManager и SignInManager.

Программный код данных классов приведен в приложении.

* + - 1. Задачи

Поскольку задача в приложении создается исключительно для реализации для нее таймера, не требуется множества описательных данных, а лишь имя задачи чтобы пользователь смог ассоциировать с ним свои реальные цели. Создадим класс Objective со следующими атрибутами:

Name – имя задачи;

WeeklyTimeGoal – выделяемое на неделю время на задачу;

TotalWeeks – количество недель для реализации задачи;

StartDate – дата создания задачи;

TimeSpent – общее время, проведенное над задачей;

TimeSpentToday – время, потраченное на задачу сегодня;

LastDate – дата последней работы над задачей;

ProjectedForToday – время, которое необходимо потратить на задачу сегодня.

Также есть метод Spend, который принимает временной промежуток (уже использованное на задачу время) и обновляет соответствующие свойства класса.

Внутри данного класса необходимо реализовать get-only свойство, которое рассчитывает время, которое пользователю необходимо использовать на данную задачу сегодня. Вычислим его как частное от деления времени, оставшегося до завершения данной задачи с момента начала текущих суток на количество дней до завершения проекта.

Для предотвращения ситуации, когда формат времени в классе Objective и времени, возвращаемого при инициализации объекта DateTime, отличается, при сравнении всегда будем приводить время к одному формату следующим образом:

LastDate == DateTime.Now.Date.ToUniversalTime();

* + - 1. Цели пользователя

Цели пользователя – класс, содержащий коллекцию задач данного пользователя, его имя и идентификатор. Он предназначен для единовременного получения сразу всех свойств задач для одного пользователя при обращении к базе данных. Его артибутами являются:

Id – уникальный идентификатор объекта в базе данных;

Username – Имя пользователя;

UserId – идентификатор пользователя;

Objectives – коллекция целей пользователя. Реализуется как объект, реализующий обобщенный интерфейс IEnumerable<T>, который представляет коллекцию C#, а также множество операций, доступных для него.

* + - 1. Хранилище целей пользователя

Хранилище целей пользователя реализуем с использованием паттерна «Репозиторий». Но, кроме этого, обеспечим полиморфное поведение классов, зависящих от этого хранилища, объявив интерфейс этого хранилища IUserObjectivesRepository. Интерфейс объявляет следующие методы (синтаксис С#):

Task SaveUserObjectivesAsync(UserObjectives userObjectives);

Task UpdateUserObjectivesAsync(UserObjectives newObjectives);

Task<UserObjectives> GetUserObjectivesAsync(string userId);

Данные методы возвращают Task, поскольку являются асинхонными. Чтобы получить результат этих методов, необходимо использовать ключевое слово await. В таком случае, если асинхронный метод еще не завершил выполнения, программа остановится в точке вызова await и дождется выполнения этого метода.

Теперь можно реализовать конкретный метод UserObjectivesRepository, реализующий созданный ранее интерфейс. Данный класс использует для хранения данных удаленную No-SQL базу данных, предоставляемую MongoDB.

* + - 1. Таймер

Тип таймера, обеспечивающий необходимую функциональность – секундомер. Класс секундомера представлен в библиотеке System.Diagnostics фреймворка .NET. под названием Stopwatch. В нем представлены методы запуска, остановки, сброса таймера. Этот класс использует механизм подсчета количества тактов тактового генератора компьютера для обеспечения точности. Помимо средств манипулирования секундомером, классу ObjectiveStopwatch необходимы дополнительные атрибуты. В данном случае мы могли наследовать наш класс от встроенного и добавить необходимые атрибуты. Но такой подход, хоть и возможен, но неверен, поскольку композиция обеспечивает большую гибкость. Поэтому создадим класс ObjectiveStopwatch, содержащий следующие атрибуты:

\_stopwatch – экземпляр встроенного секундомера;

ObjectiveName – имя задачи;

Elapsed – возвращает накопленное время;

Методы, использующиеся в классе ObjectiveStopwatch делегируют выполнение действий над таймером экземпляру таймера, хранящемуся в атрибутах класса.

* + - 1. Хранилище таймеров

Поскольку в пределах приложения для одного пользователя не должно существовать более чем одного таймера, создадим singleton-репозиторий таймеров. Такой репозиторий должен однозначно связывать пользователя с таймером и возвращать единственный нужный таймер. Объявим интерфейс IObjectiveStopwatch для этого репозитория. Он объявляет одну операцию: ObjectiveStopwatch GetUserStopwatch(string userId). Реализуем конкретный класс для данного интерфейса. Для связывания пользователя с таймером используем обобщенный словарь из библиотеки System.Collections.Generic – Dictionary. Он позволяет по ключу – идентификатору пользователя получить таймер.

Код класса приведен в приложении.

* + - 1. Контроллеры

На диаграмме 3.1 изображены классы AccountController и TimerController. Предполагается, что эти классы будут являться связующим звеном между Model и View частями приложения. Для этого, эти классы будут единственными, кто способен принимать Http-запросы от пользователя. Оба класса являются примерами инкапсуляции поскольку внутреннее состояние скрыто, а открыты лишь сервисы, которые эти классы предоставляют.

* + - * 1. Контроллер аутентификации

AccountController должен предоставлять функциональность аутентификации в приложении. Классы контроллеров в asp.net требуется наследовать от базового класса Controller. В нем присутствуют следующие методы:

Login();

Login(UserLoginModel) - перегружен;

Create();

Create(UserCreateModel) - перегружен.

SignOut().

Методы Login, Create представлены в двух экземплярах с разными параметрами (перегружены). Методы без параметров возвращают страницу с формой для заполнения данных. Форма же, в свою очередь, посылает запрос с соответствующим параметром к методу, перегруженному для приема этого параметра. Метод SignOut реализован без параметров, так как предполагается, что текущий пользователь существует в инфраструктуре фреймворка, на базе которого создается приложение.

Атрибутами класса TimerController будут экземпляры классов SignInManager и UserManager, предоставляющие интерфейсы для манипуляции списком пользователей приложения и аутентификации. Эти классы предоставляются инфраструктурой фреймворка. Также атрибутом AccountController является объект, реализующий интерфейс IUserObjectivesRepository. Он нужен для того, чтобы при регистрации пользователя в хранилище целей пользователей создавалась запись с новым пользователем.

Методы Login и Create принимают параметрами экземпляры классов UserCreateModel и UserLoginModel. Эти классы являются так называемыми View Model, - отображением класса в модели на представление. Отличие заключается в том, что от оригинального класса берутся только те атрибуты, которые необходимы для осуществления операции. Кроме того, на эти классы уже можно применять правила валидации, в отличие от классов, представленных в модели. Код классов представлен в приложении.

* + - * 1. Контроллер таймеров

Рассмотрим класс TimerController. Он предоставляет логику для работы с таймерами. Класс содержит следующие атрибуты:

UserManager для получения идентификатора текущего пользователя;

IUserObjectivesRepository для получения задач данного пользователя;

IStopwatchRepository для получения таймера для данного пользователя;

UserStopwatch – сам таймер;

UserId – идентификатор текущего пользователя.

TimerController содержит следующие методы:

Index() – метод, возвращающий домашнюю страницу сайта;

StartWatch(ObjectiveName) – запускает таймер для конкретной цели;

StopWatch(ObjectiveName) – останавливает таймер;

CreateObjective() – возвращает страницу с формой создания задачи;

CreateObjective(ObjectiveCreateModel) – создает задачу пользователя;

RemoveObjective(Objective) – удаляет задачу.

Метод CreateObjective(ObjectiveCreateModel) принимает параметром экземпляр класса ObjectiveCreateModel – View Model. Класс содержит три атрибута: имя, цель времени на неделю и количество недель. Для класса применена валидация. Код классов приведен в приложении.

* + - 1. Запуск таймеров на клиенте

Для запуска таймера на клиенте используется документ javascript, привязанный к HTML документу. Он обеспечивает динамическое изменение состояния таймера на клиенте без перезагрузки приложения.

* + 1. Дополнительно использованные функции
       1. Система контроля версий

Для разработки проекта активно использовалась система контроля версий Git. Репозиторий с проектом публично доступен по адресу: <https://github.com/astro6703/EffectiveTimeUsageTracker>.

Системы контроля версий незаменимы при разработке больших проектов. Они позволяют сохранять текущее состояние проекта и откатываться к ранее сохраненным состояниям.

* + - 1. Внедрение зависимостей (Dependency injection)

Внедрение зависимостей — это стиль настройки объекта, при котором управление разрешением зависимостей объектов предоставляется внешним сущностям в полном соответствии с принципом единственной ответственности. Зависимость – объект, экземпляр которого требуется другому объекту. Другими словами, объекты настраиваются внешними объектами. DI — это альтернатива самонастройке объектов. Является одной из форм инверсии управления («Inversion of control»).

В платформе ASP.NET Core внедрение зависимостей осуществляется через классы-сервисы. Чтобы создать сервис для внедрения, необходимо в метод ConfigureServices() класса Startup добавить вызов метода, осуществляющего определенный сервис. Сервисов для осуществления внедрения зависимостей три:

Transient – запрашиваемый экземпляр создается заново каждый раз, когда необходим;

Singleton – создается один экземпляр запрашиваемого объекта на все время работы программы;

Scoped – экземпляр создается для каждого запроса, но удаляется только при выходе из соответствующей запросу зоны видимости.

Методы добавления сервиса на внедрение зависимостей:

services.AddSingleton<Dependency>();

services.AddTransient<Dependency>();

services.AddScoped<Dependency>();

В данном проекте внедрение зависимостей используется для разрешения следующих зависимостей:

IStopwatchRepository - singleton;

IMongoClient - singleton;

IUserObjectivesRepository – transient.

* + - 1. ORM (Object-Relational mapping)

ORM – технология, позволяющая привязать объекты объектно-ориентированного языка к отношениям реляционной базы данных. Используется в тех случаях, когда с данными необходимо работать в терминах классов, а не таблиц. Это осуществляется за счет того, что ORM-фреймворк скрывает все операции непосредственно с базой данных за программным интерфейсом.

В данном проекте использован ORM-фреймворк Entity Framework Core. Для работы в нем необходимо создать класс контекста данных, унаследованный от встроенного класса DbContext. В нем необходимо поместить get-set свойство с коллекцией типа DbSet<T>, где T – тип данных, которые вы хотите хранить в базе данных. Чтобы настроить базу данных существуют конструкторы параметров, которые передаются в конструктор контекста данных.

Для изменения структуры базы данных используется концепция миграций – изменений структуры, описанных в виде C# классов. Система работает следующим образом: при изменении используемой модели данных возникает необходимость изменить структуру этой модели, следовательно, необходимо изменить и структуру базы данных. Для этого используется команда командной строки dotnet ef migrations add. После выполнения команды в каталоге проекта появятся классы миграций, которые можно просматривать и редактировать. Однако, при этом структура не изменяется. Она изменится только после успешного выполнения команды dotnet ef database update, которая применяет все миграции к базе данных.

В качестве поставщика реляционной базы используется LocalDB – расширение для Visual Studio, позволяющее использовать MS SQL Server без создания реального сервера для целей разработки и отладки.

* + - 1. No-SQL база данных

В ходе работы возникла необходимость хранить в базе данные, которые не могут иметь постоянной одинаковой структуры. В этом случае мы используем No-SQL базы данных, которые менее эффективны, но могут хранить данные любой структуры в пределах одной коллекции. В данном проекте использовалась база, поставляемая Mongo.

В базе MongoDb данные представляются в виде документов BSON. Документы хранятся в коллекциях. Достоинством такой модели является быстрое развертывание базы данных и возможность использовать данные разнородной структуры.

Для использования необходимо всего лишь проинициализировать объект MongoClient со строкой-connection string. После этого операции для работы с базой данных становятся доступны из интерфейса этого объекта.

В данном проекте экземпляр, реализующий IMongoClient был с помощью внедрения зависимостей включен в класс UserObjectivesRepository.

* + 1. Интерфейс пользователя

В разработанном сайте существует домашняя страница. Однако, чтобы попасть на нее, сначала необходимо войти в свой аккаунт пользователя. Если аккаунта пользователя нет, придется зарегистрироваться. Страница регистрации выглядит следующим образом:

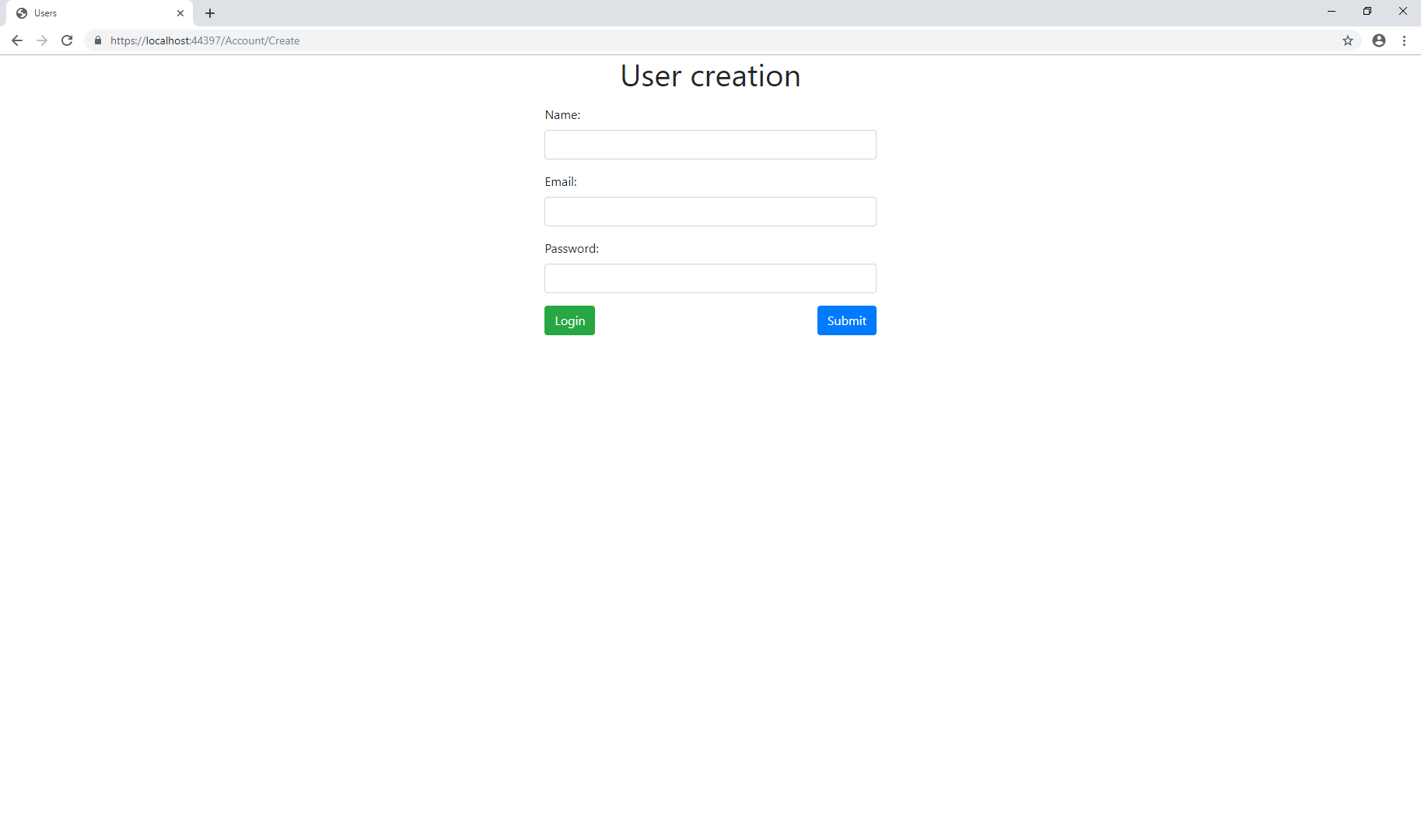


Рисунок 3.1 – Вид страницы регистрации пользователя

На странице присутствует форма ввода регистрационных данных, кнопка подтверждения и кнопка перехода к странице логина.

Если на данной странице допустить ошибки валидации, на экране покажется сообщение о том, какие именно данные введены неправильно:

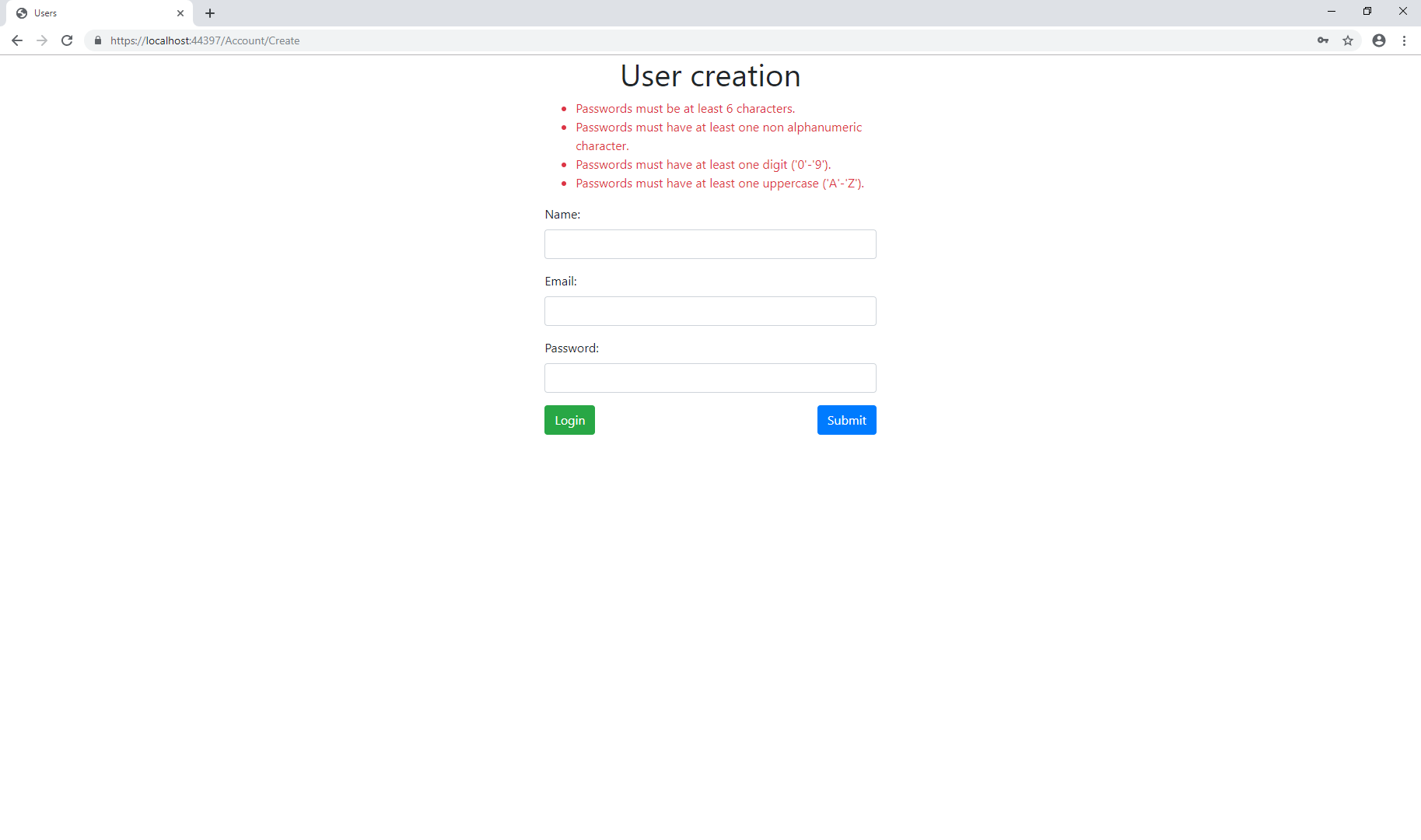


Рисунок 3.2 – Вывод ошибок валидации на странице регистрации пользователя

После завершения регистрации вы попадете на главную страницу. Однако, сначала рассмотрим страницу логина:

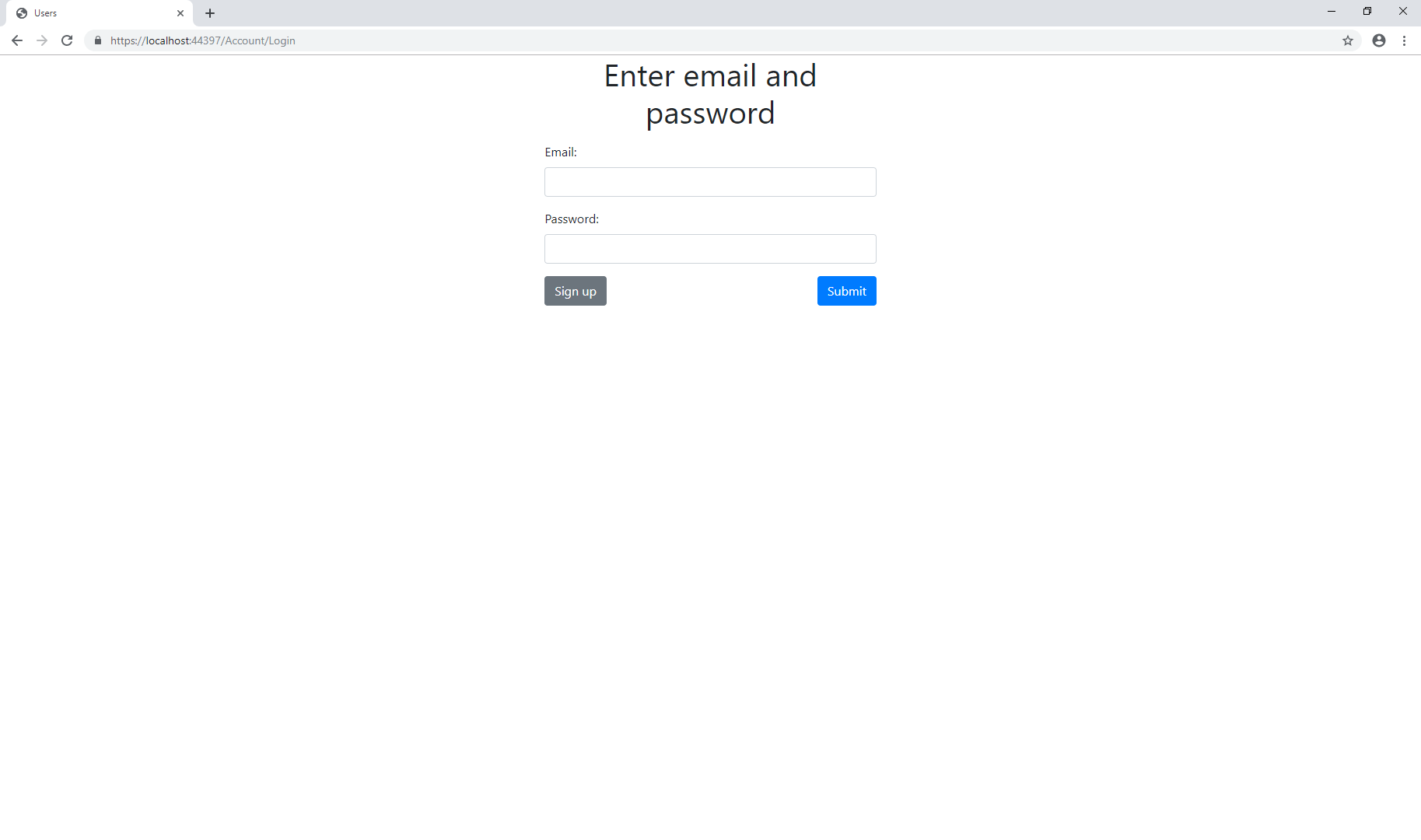


Рисунок 3.3 – Страница логина

Она содержит форму для ввода данных и кнопки перехода на страницу регистрации и подтверждения формы.

На данной странице также присутствует валидация:

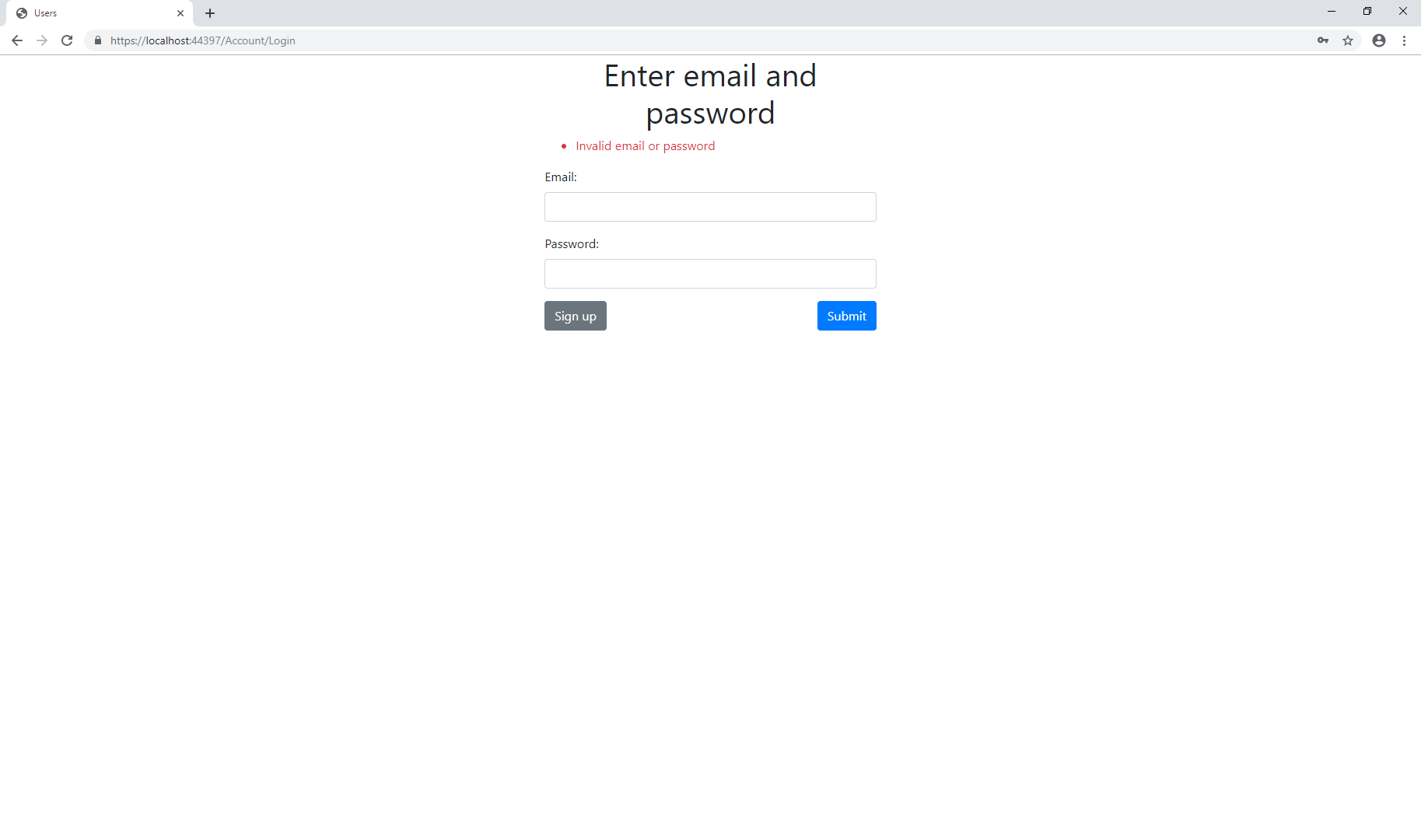


Рисунок 3.4 – Ошибки валидации на странице логина

Совершив успешный вход, перейдем на главную страницу:

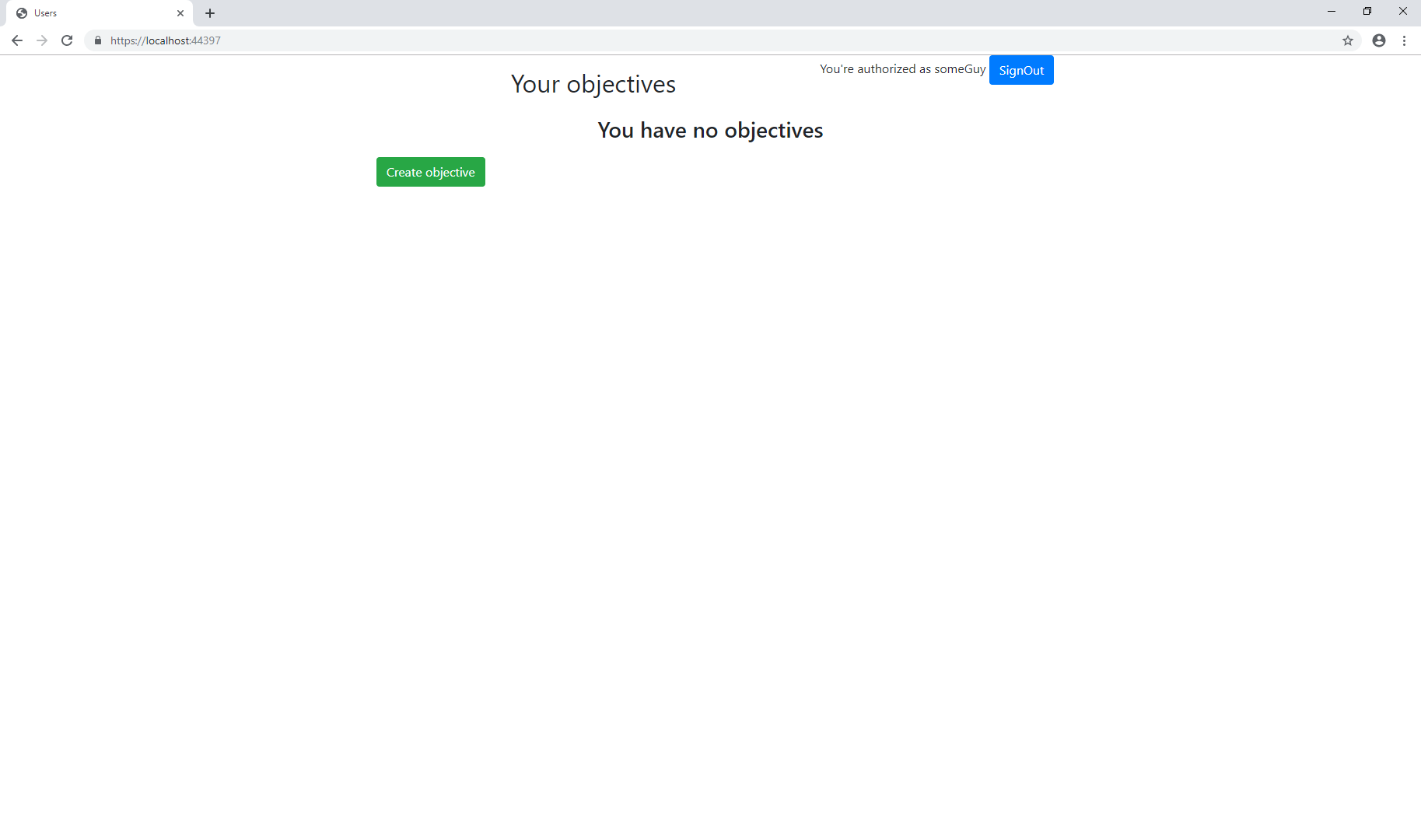


Рисунок 3.5 – Главная страница только что зарегистрированного пользователя

Данная страница содержит секцию для задач пользователя, но сейчас она пуста. Также здесь есть кнопка выхода из системы текущего пользователя и кнопка создания задачи. Создадим задачу, перейдя по кнопке к странице создания задачи:

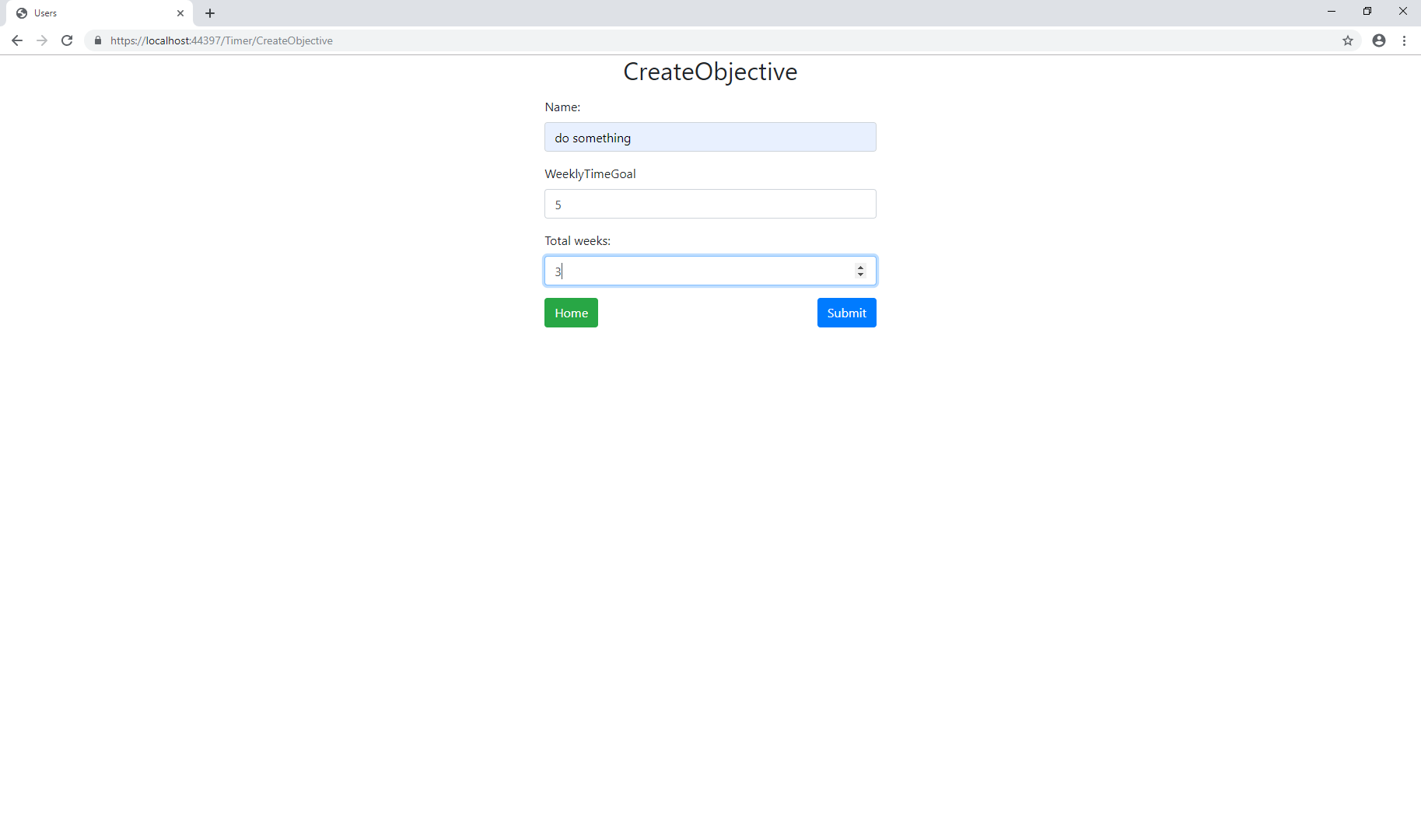


Рисунок 3.6 – Вид страницы создания задачи

На данной странице содержится форма для заполнения данных, кнопки перехода на домашнюю страницу и подтверждения формы. Подтвердим введенные данные:

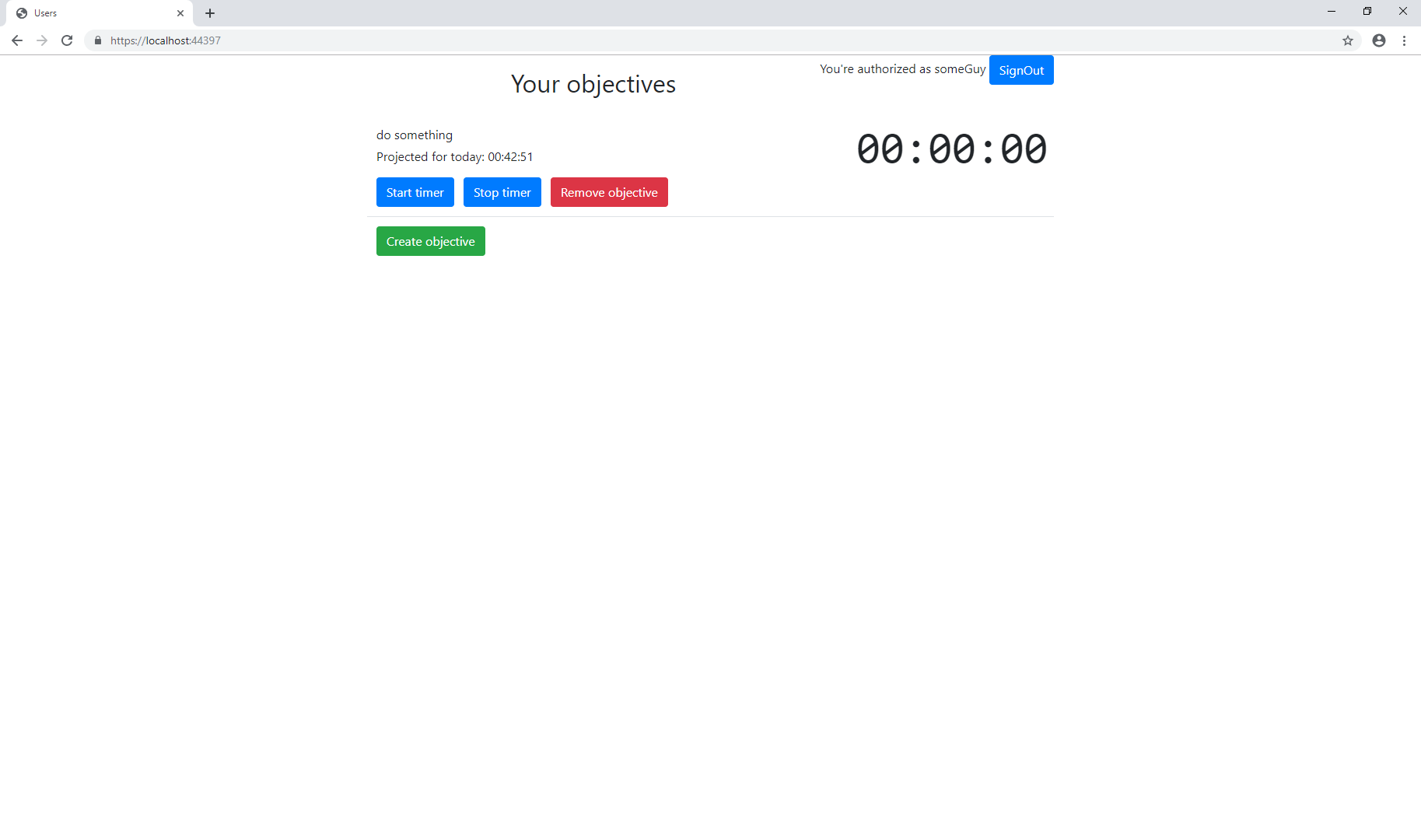


Рисунок 3.7 – Вид главной страницы с таймерами

Как видно из рисунка, на странице появился таймер, соответствующий задаче, которую мы только что создали. В контейнере данной задачи также есть кнопки запуска таймера, остановки таймера и удаления задачи. Кроме того, в контейнере находятся данные о запланированном на сегодня времени, которое нужно потратить на эту задачу. Запустим таймер:

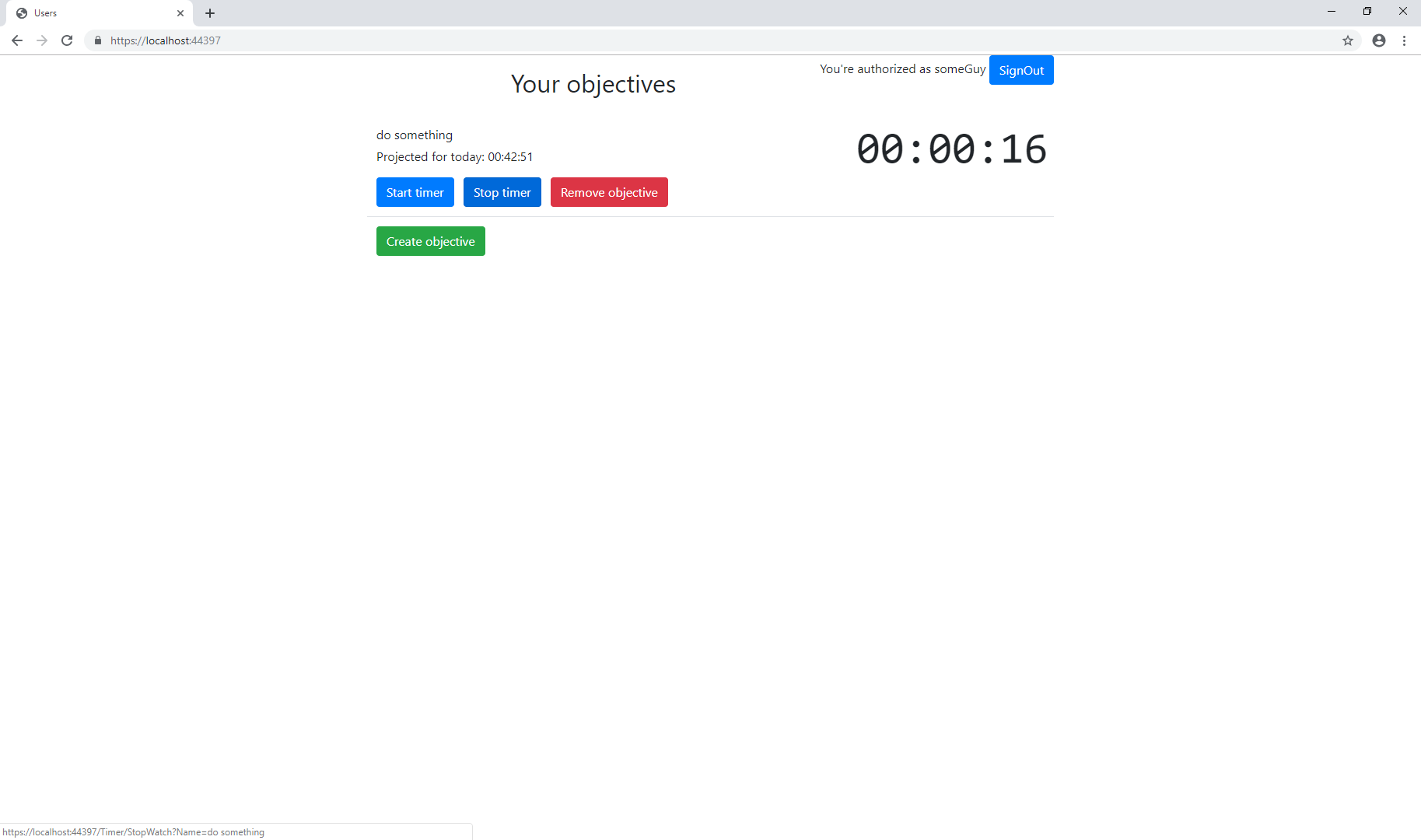


Рисунок 3.8 – Состояние таймера после его кратковременного включения и выключения

Состояние таймера изменилось как на клиенте в браузере, так и на сервере. Обратите внимание, что количество времени, запланированное на сегодня не изменилось. Добавим еще несколько задач:

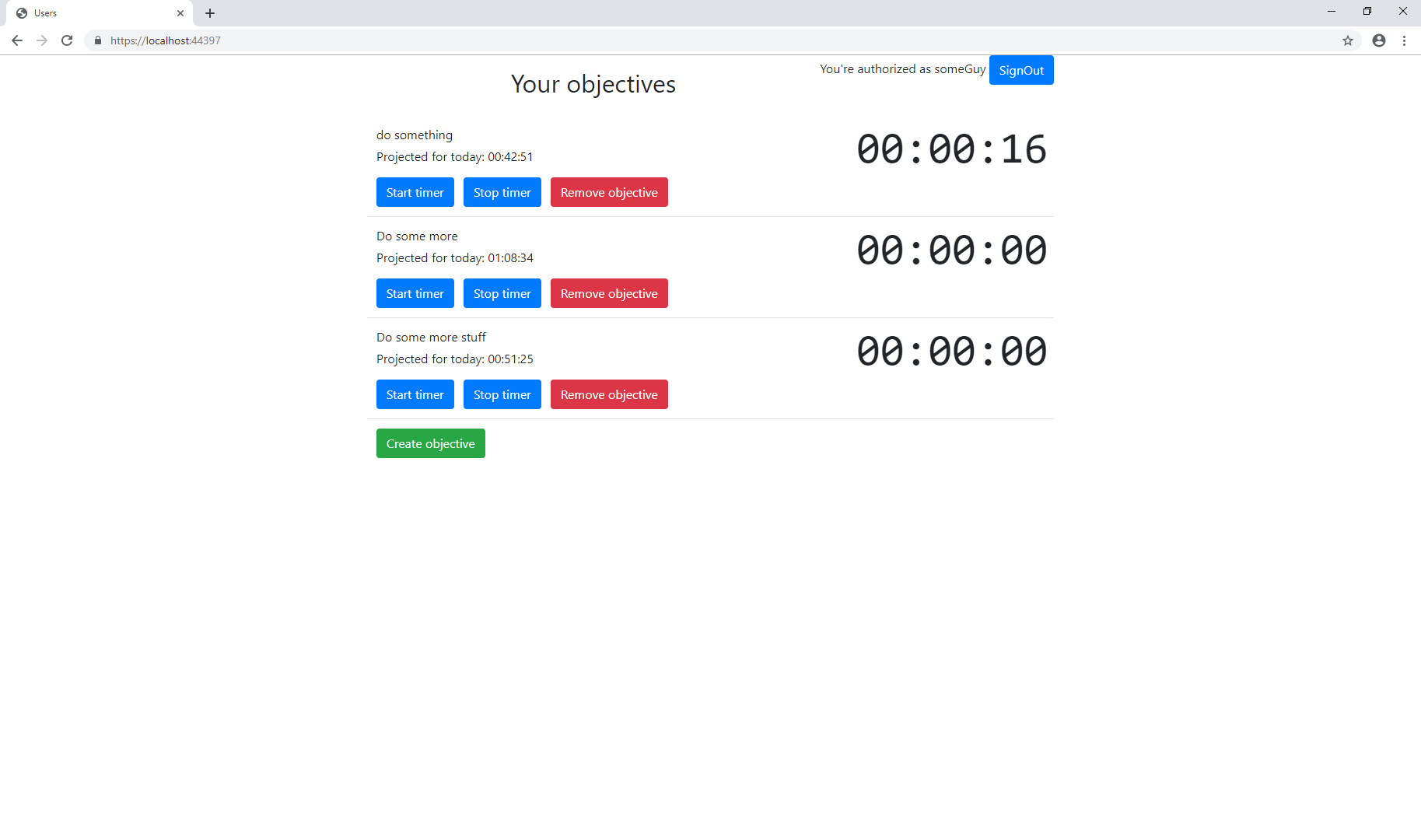


Рисунок 3.9 – Вид главной страницы при наличии нескольких задач

Когда на странице находится несколько задач, одновременно можно включить только одну из них. При попытке включить второй таймер, первый остановится.

* + 1. Критерий качества программной системы

Для спроектированной системы применимы как стандартные оценки качества программных продуктов, так и оценки качества веб-сайтов. Согласно стандарта ISO 9126, основными критериями качества программного продукта являются:

функциональность – способность ПО выполнять набор функций (действий), удовлетворяющих заданным или подразумеваемым потребностям пользователей;

 надежность – это его способность с достаточно большой вероятностью безотказно выполнять определенные функции при заданных условиях и в течение заданного периода времени;

эффективность – соотношение уровня услуг, предоставляемых ПО пользователю при заданных условиях, и объема используемых для этого ресурсов. К числу таких ресурсов могут относиться требуемые аппаратные средства, время выполнения программ, затраты на подготовку данных и интерпретацию результатов;

эргономичность – характеристики ПО, которые позволяют минимизировать усилия пользователя по подготовке исходных данных, применению ПО и оценке полученных результатов, а также вызывать положительные эмоции определенного или подразумеваемого пользователя;

модифицируемость – характеристики ПО, которые позволяют минимизировать усилия по внесению изменений для устранения ошибок и по его модификации в соответствии с изменяющимися потребностями пользователей. Модифицируемость ПО существенно зависит от степени и качества его документированности;

мобильность – способность ПО быть перенесенным из одной среды (окружения) в другую, в частности, с одной аппаратной платформы на другую.

Кроме данных критериев, обозначим критерии качества для вебсайтов:

видимость сайта поисковыми системами;

удобство использования сайта;

дизайн сайта;

функциональность.

1. ВЫВОДЫ

В ходе выполнения курсового проекта была выполнена разработка приложения в соответствии с требованиями технического задания по курсовому проектированию на тему «проектирование и программная реализация системы, моделирующей процессы создания и взаимодействия группы объектов». Для реализации заявленной функциональности была разработана доменная модель, содержащая классы для работы с таймерами и их хранения. Затем были продуманы способы взаимодействия объектов доменной модели с объектами, являющимися частью Controller архитектурного паттерна MVC, которые затем были реализованы. Последним этапом создание приложения стала разработка представлений на базе HTML. Разработанная программа уже сейчас может быть использована в личных целях.

При разработке возник ряд проблем, связанных с архитектурой приложения и его визуальной частью, которые в итоге были успешно решены, вследствие чего были повышены навыки отладки объектно-ориентированных приложений.

В ходе работы использовались дополнительные средства, такие как Git, Dependency Injection, ORM, No-SQL, были улучшены навыки работы с ними.

В будущем планируется осуществить рефакторинг программного кода и модульное тестирование для повышения качества продукта. После выполнения модульного тестирования возможно создание мобильной версии сайта и мобильного приложения, осуществляющих аналогичную функциональность.

Объектно-ориентированный подход улучшает структурированность кода, позволяет гибко выстраивать архитектуру приложений.

1. ПЕРЕЧЕНЬ ССЫЛОК

Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений.: Пер с англ. / Г. Буч // М. : ООО "И.Д. Вильямс", 2008 г.

Adam Freeman, “Pro ASP.NET Core 2 MVC”, 2017, 1017 p.

Adam Freeman, “Pro Entity Framework Core 2 for ASP.NET Core MVC”, 2018, 650 p.

Albahari J., Albahari B., “С# 7.0 in a nutshell”, 2018, 1071 p.

Фримен, Эрик Паттерны проектирования / Эрик Фримен и др. - М.: Питер, 2015. - 656 c.

Rauschmayer A., “Speaking JavaScript”, 2018, 460 p.

1. ПРИЛОЖЕНИЕ

Программный код разработанной программы:

public class ApplicationUser : IdentityUser { }

public class UsersIdentityDbContext : IdentityDbContext<ApplicationUser>

{

public UsersIdentityDbContext(DbContextOptions<UsersIdentityDbContext> options) : base(options) { }

}

public class Objective

{

public string Name { get; set; }

public int WeeklyTimeGoal { get; set; }

public int TotalWeeks { get; set; }

public DateTime StartDate { get; set; }

public TimeSpan TimeSpent { get; set; }

public TimeSpan TimeSpentToday { get; set; }

public DateTime LastDate { get; set; }

public TimeSpan ProjectedForToday

{

get

{

var projectedTimeLeftFromTodaysNight = new TimeSpan(WeeklyTimeGoal \* TotalWeeks, 0, 0) - TimeSpent + TimeSpentToday;

var daysLeft = Math.Floor((StartDate.AddDays(TotalWeeks \* 7) - StartDate).TotalDays);

return projectedTimeLeftFromTodaysNight / daysLeft;

}

}

public void Spend(TimeSpan time)

{

if (LastDate == DateTime.Now.Date.ToUniversalTime())

{

TimeSpentToday += time;

TimeSpent += time;

}

else

{

LastDate = DateTime.Now.Date.ToUniversalTime();

TimeSpentToday = time;

TimeSpent += time;

}

}

}

public class UserObjectives

{

[BsonId]

public BsonObjectId Id { get; set; }

[BsonRequired]

public string Username { get; set; }

public string UserId { get; set; }

public IEnumerable<Objective> Objectives { get; set; }

}

public interface IUserObjectivesRepository

{

Task SaveUserObjectivesAsync(UserObjectives userObjectives);

Task UpdateUserObjectivesAsync(UserObjectives newObjectives);

Task<UserObjectives> GetUserObjectivesAsync(string userId);

}

public class UserObjectivesRepository : IUserObjectivesRepository

{

private readonly IMongoClient \_client;

public UserObjectivesRepository(IMongoClient client)

=> \_client = client ?? throw new ArgumentNullException("Mongo client was null");

public async Task<UserObjectives> GetUserObjectivesAsync(string userId)

{

if (userId == null)

throw new ArgumentNullException($"{nameof(userId)} string instance was null");

if (string.IsNullOrWhiteSpace(userId))

throw new ArgumentException($"{nameof(userId)} string was empty or whitespace");

var database = \_client.GetDatabase("TomatoTimeTracker");

var collection = database.GetCollection<UserObjectives>("UserObjectives");

return await collection

.AsQueryable()

.Where(x => x.UserId == userId)

.FirstOrDefaultAsync();

}

public async Task SaveUserObjectivesAsync(UserObjectives userObjectives)

{

if (userObjectives == null)

throw new ArgumentNullException($"{nameof(userObjectives)} instance was null");

if (userObjectives.UserId == null)

throw new ArgumentException($"{nameof(userObjectives)} {nameof(userObjectives.UserId)} property was null");

if (string.IsNullOrWhiteSpace(userObjectives.UserId))

throw new ArgumentException($"{nameof(userObjectives)} {nameof(userObjectives.UserId)} property was empty or whitespace");

var database = \_client.GetDatabase("TomatoTimeTracker");

var collection = database.GetCollection<UserObjectives>("UserObjectives");

if (await GetUserObjectivesAsync(userObjectives.UserId) != null)

throw new InvalidOperationException("Instance of UserObjectives for this user already exists");

await collection.InsertOneAsync(userObjectives);

}

public async Task UpdateUserObjectivesAsync(UserObjectives newObjectives)

{

if (newObjectives == null)

throw new ArgumentNullException($"{nameof(newObjectives)} instance was null");

if (string.IsNullOrWhiteSpace(newObjectives.UserId))

throw new ArgumentException($"{nameof(newObjectives)} {nameof(newObjectives.UserId)} property was empty or whitespace");

var database = \_client.GetDatabase("TomatoTimeTracker");

var collection = database.GetCollection<UserObjectives>("UserObjectives");

var oldObjective = await GetUserObjectivesAsync(newObjectives.UserId);

if (oldObjective == null)

throw new InvalidOperationException($"{nameof(newObjectives)} instance to update is missing");

newObjectives.Id = oldObjective.Id;

var filter = new BsonDocument("\_id", newObjectives.Id);

await collection.ReplaceOneAsync(filter, newObjectives);

}

}

public class StopwatchRepository : IStopwatchRepository

{

private IDictionary<string, ObjectiveStopwatch> \_stopwatchesMap;

public StopwatchRepository()

{

\_stopwatchesMap = new Dictionary<string, ObjectiveStopwatch>();

}

public ObjectiveStopwatch GetUserStopwatch(string userId)

{

if (userId == null) throw new ArgumentNullException($"{nameof(userId)} was null");

if (string.IsNullOrWhiteSpace(userId)) throw new ArgumentNullException($"{nameof(userId)} was empty or whitespace");

if (\_stopwatchesMap.TryGetValue(userId, out var userStopwatch))

return userStopwatch;

var stopwatch = new ObjectiveStopwatch();

\_stopwatchesMap.Add(userId, stopwatch);

return stopwatch;

}

}

public class ObjectiveStopwatch

{

private readonly Stopwatch \_stopwatch = new Stopwatch();

public string ObjectiveName { get; private set; }

public TimeSpan Elapsed => \_stopwatch.Elapsed;

public void Start() => \_stopwatch.Start();

public void Stop() => \_stopwatch.Stop();

public void SetObjective(string objectiveName)

{

ObjectiveName = objectiveName ?? throw new ArgumentNullException("Objective name was null");

\_stopwatch.Reset();

}

public void ResetObjective() => ObjectiveName = null;

}

public interface IStopwatchRepository

{

ObjectiveStopwatch GetUserStopwatch(string userId);

}

public class AccountController : Controller

{

private readonly UserManager<ApplicationUser> \_userManager;

private readonly SignInManager<ApplicationUser> \_signInManager;

private readonly IUserObjectivesRepository \_objectivesRepository;

public AccountController(UserManager<ApplicationUser> userManager, SignInManager<ApplicationUser> signInManager, IUserObjectivesRepository objectivesRepository)

{

\_userManager = userManager ?? throw new ArgumentNullException($"{nameof(userManager)} was null");

\_signInManager = signInManager ?? throw new ArgumentNullException($"{nameof(signInManager)} was null");

\_objectivesRepository = objectivesRepository ?? throw new ArgumentNullException($"{nameof(signInManager)} was null");

}

[AllowAnonymous]

public IActionResult Login()

{

return View();

}

[AllowAnonymous]

public IActionResult Create()

{

return View();

}

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> Login(UserLoginModel loginModel)

{

if (loginModel == null) throw new ArgumentNullException($"{nameof(loginModel)} was null");

if (ModelState.IsValid)

{

var user = await \_userManager.FindByEmailAsync(loginModel.Email);

if (user != null)

{

await \_signInManager.SignOutAsync();

var signInResult = await \_signInManager.PasswordSignInAsync(user, loginModel.Password, true, false);

if (signInResult.Succeeded)

return RedirectToAction("Index", "Timer");

}

ModelState.AddModelError(nameof(loginModel.Email), "Invalid email or password");

}

return View(loginModel);

}

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> Create(UserCreateModel createModel)

{

if (createModel == null) throw new ArgumentNullException($"{nameof(createModel)} was null");

if (ModelState.IsValid)

{

var user = new ApplicationUser

{

UserName = createModel.Name,

Email = createModel.Email

};

var identityResult = await \_userManager.CreateAsync(user, createModel.Password);

if (identityResult.Succeeded)

{

var objectives = new UserObjectives()

{

Username = createModel.Name,

UserId = await \_userManager.GetUserIdAsync(user),

Objectives = new List<Objective>()

};

await \_objectivesRepository.SaveUserObjectivesAsync(objectives);

await \_signInManager.PasswordSignInAsync(user, createModel.Password, true, false);

return RedirectToAction("Index", "Timer");

}

foreach (var error in identityResult.Errors)

ModelState.AddModelError("", error.Description);

}

return View(createModel);

}

[Authorize]

public async Task<IActionResult> SignOut()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Timer");

}

}

public class TimerController : Controller

{

private readonly IUserObjectivesRepository \_userObjectivesRepository;

private readonly UserManager<ApplicationUser> \_userManager;

private readonly IStopwatchRepository \_stopwatchRepository;

private string UserID;

private ObjectiveStopwatch UserStopwatch;

public TimerController(

IUserObjectivesRepository repository,

UserManager<ApplicationUser> userManager,

IStopwatchRepository stopwatchRepository)

{

\_userObjectivesRepository = repository ?? throw new ArgumentNullException($"{nameof(repository)} was null");

\_userManager = userManager ?? throw new ArgumentNullException($"{nameof(userManager)}");

\_stopwatchRepository = stopwatchRepository ?? throw new ArgumentNullException($"{nameof(stopwatchRepository)}");

}

public override void OnActionExecuting(ActionExecutingContext context)

{

UserID = \_userManager.GetUserId(User);

UserStopwatch = \_stopwatchRepository.GetUserStopwatch(UserID);

}

[Authorize]

public async Task<IActionResult> Index()

{

var objectives = await \_userObjectivesRepository.GetUserObjectivesAsync(\_userManager.GetUserId(User));

return View(objectives);

}

[Authorize]

public async Task<IActionResult> StartWatch(string name)

{

if (name == null) throw new ArgumentNullException(nameof(name));

UserStopwatch.Stop();

var objectives = await \_userObjectivesRepository.GetUserObjectivesAsync(\_userManager.GetUserId(User));

if (!objectives.Objectives.Where(x => x.Name == name).Any())

throw new InvalidOperationException("No objective with such name exists");

var currentObjectiveName = UserStopwatch.ObjectiveName;

if (currentObjectiveName != null)

{

var currentObjective = objectives.Objectives.Where(x => x.Name == currentObjectiveName).FirstOrDefault();

currentObjective.Spend(UserStopwatch.Elapsed);

objectives.Objectives = objectives.Objectives.UpdateObjective(currentObjective).ToArray();

await \_userObjectivesRepository.UpdateUserObjectivesAsync(objectives);

}

UserStopwatch.SetObjective(name);

UserStopwatch.Start();

return NoContent();

}

[Authorize]

public async Task<IActionResult> StopWatch(string name)

{

UserStopwatch.Stop();

var objectives = await \_userObjectivesRepository.GetUserObjectivesAsync(\_userManager.GetUserId(User));

if (!objectives.Objectives.Where(x => x.Name == name).Any())

return BadRequest("No objective with such name exists");

var currentObjectiveName = UserStopwatch.ObjectiveName;

if (currentObjectiveName == null)

return BadRequest("Timer is not set up");

var currentObjective = objectives.Objectives.Where(x => x.Name == currentObjectiveName).FirstOrDefault();

currentObjective.Spend(UserStopwatch.Elapsed);

UserStopwatch.ResetObjective();

objectives.Objectives = objectives.Objectives.UpdateObjective(currentObjective).ToArray();

await \_userObjectivesRepository.UpdateUserObjectivesAsync(objectives);

return NoContent();

}

[Authorize]

public IActionResult CreateObjective()

=> View();

[Authorize]

[HttpPost]

public async Task<IActionResult> CreateObjective(ObjectiveCreateModel objectiveCreateModel)

{

if (objectiveCreateModel == null) throw new ArgumentNullException($"{nameof(objectiveCreateModel)} instance was null");

if (ModelState.IsValid)

{

var currentUserObjectives = await \_userObjectivesRepository.GetUserObjectivesAsync(\_userManager.GetUserId(User));

if (currentUserObjectives == null)

ModelState.AddModelError("", "Cannot find userObjectives for this user");

else if (currentUserObjectives.Objectives.Where(x => x.Name == objectiveCreateModel.Name).Any())

ModelState.AddModelError("", "Objective with this name already exists");

else

{

var objective = new Objective

{

Name = objectiveCreateModel.Name,

TotalWeeks = objectiveCreateModel.TotalWeeks,

WeeklyTimeGoal = objectiveCreateModel.WeeklyTimeGoal,

StartDate = DateTime.Now.Date.ToUniversalTime(),

LastDate = DateTime.Now.Date.ToUniversalTime()

};

currentUserObjectives.Objectives = currentUserObjectives.Objectives.AddObjective(objective).ToArray();

await \_userObjectivesRepository.UpdateUserObjectivesAsync(currentUserObjectives);

return RedirectToAction("Index", "Timer");

}

}

return View(objectiveCreateModel);

}

public async Task<IActionResult> RemoveObjective(string name)

{

if (name == null) throw new ArgumentNullException($"{nameof(name)} was null");

if (string.IsNullOrWhiteSpace(name)) throw new ArgumentException($"{nameof(name)} was empty or whitespace");

var userObjectives = await \_userObjectivesRepository.GetUserObjectivesAsync(\_userManager.GetUserId(User));

var objective = userObjectives.Objectives.Where(x => x.Name == name).FirstOrDefault();

userObjectives.Objectives = userObjectives.Objectives.RemoveObjective(objective);

await \_userObjectivesRepository.UpdateUserObjectivesAsync(userObjectives);

UserStopwatch.ResetObjective();

return RedirectToAction("Index", "Timer");

}

}

public class ObjectiveCreateModel

{

[Required]

public string Name { get; set; }

[Required]

public int WeeklyTimeGoal { get; set; }

[Required]

public int TotalWeeks { get; set; }

}

public class UserCreateModel

{

[Required]

public string Name { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string Password { get; set; }

}

public class UserLoginModel

{

[Required]

public string Email { get; set; }

[Required]

public string Password { get; set; }

}

public static class MyExtensionMethods

{

public static IEnumerable<Objective> UpdateObjective(this IEnumerable<Objective> enumerable, Objective objective)

{

foreach (var item in enumerable)

{

if (item.Name != objective.Name)

yield return item;

else yield return objective;

}

}

public static IEnumerable<Objective> AddObjective(this IEnumerable<Objective> enumerable, Objective objective)

{

foreach (var item in enumerable)

yield return item;

yield return objective;

}

public static IEnumerable<Objective> RemoveObjective(this IEnumerable<Objective> enumerable, Objective objective)

{

foreach (var item in enumerable)

if (item.Name != objective.Name)

yield return item;

}

}

var isRunning = false;

var clocktimer;

var currentTimerNumber;

function start() {

if (event.target.id !== currentTimerNumber) {

if (typeof currentTimerNumber !== "undefined") {

let oldStopButton = document.getElementById("stop-timer-" + currentTimerNumber);

let oldTimerName = document.getElementById("name-" + currentTimerNumber).innerHTML;

oldStopButton.click();

oldStopButton.removeEventListener("click", stop);

}

isRunning = false;

clearInterval(clocktimer);

let eventId = event.target.id.toString();

currentTimerNumber = eventId[eventId.length - 1];

document.getElementById("stop-timer-" + currentTimerNumber).addEventListener("click", stop);

time = getCurrentTime(currentTimerNumber);

seconds = time[2];

minutes = time[1];

hours = time[0];

}

if (!isRunning) {

clocktimer = setInterval("update()", 1000);

isRunning = true;

}

}

function update() {

seconds++;

if (seconds >= 60) {

seconds = 0;

minutes++;

if (minutes >= 60) {

minutes = 0;

hours++;

}

}

document.getElementById("time-spent-" + currentTimerNumber).innerHTML = hours.toString() + ":" + minutes.toString() + ":" + String(seconds).padStart(2, '0');

}

function stop() {

clearInterval(clocktimer);

isRunning = false;

}

function getCurrentTime(currentTimerNumber) {

let id = "time-spent-" + currentTimerNumber;

let time = document.getElementById(id).innerHTML;

let timeSplit = time.split(":");

return timeSplit;

}