**01.11.2024**

: **Тема** Создание проекта на основе шаблона *MVC.*

**Ход работы**:

***ASP.NET*** – это фреймворк от Microsoft для разработки веб-приложений и веб-сервисов. Он позволяет создавать динамичные сайты, поддерживает различные языки программирования, предлагает встроенные функции безопасности и оптимизирован для высокой производительности. Включает технологии, такие как *ASP.NET MVC и ASP.NET Web API.*

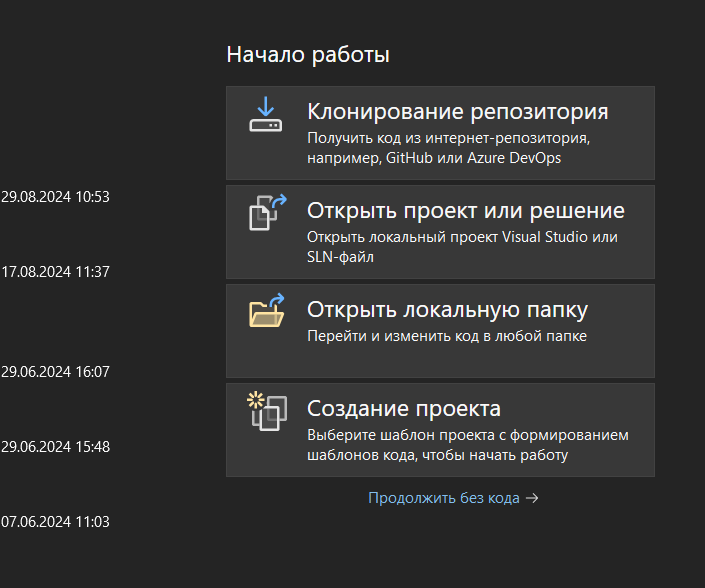


Рисунок 1 – Создание проекта

Создание веб-приложения на *ASP.NET Core MVC* начинается с установки *Visual Studio* и .*NET SDK*. Затем создайте новый проект, выбрав шаблон "*Web Application (Model-View-Controller*)". В структуре проекта будут папки для контроллеров, моделей и представлений. Создайте модель, например, *User,* и контроллер *UsersController* с методом *Index,* который возвращает представление. Добавьте представление *Index.cshtml* в папку *Views/Users*. Наконец, запустите приложение, нажав "Запуск" в *Visual Studio*. Это основные шаги для начала работы!

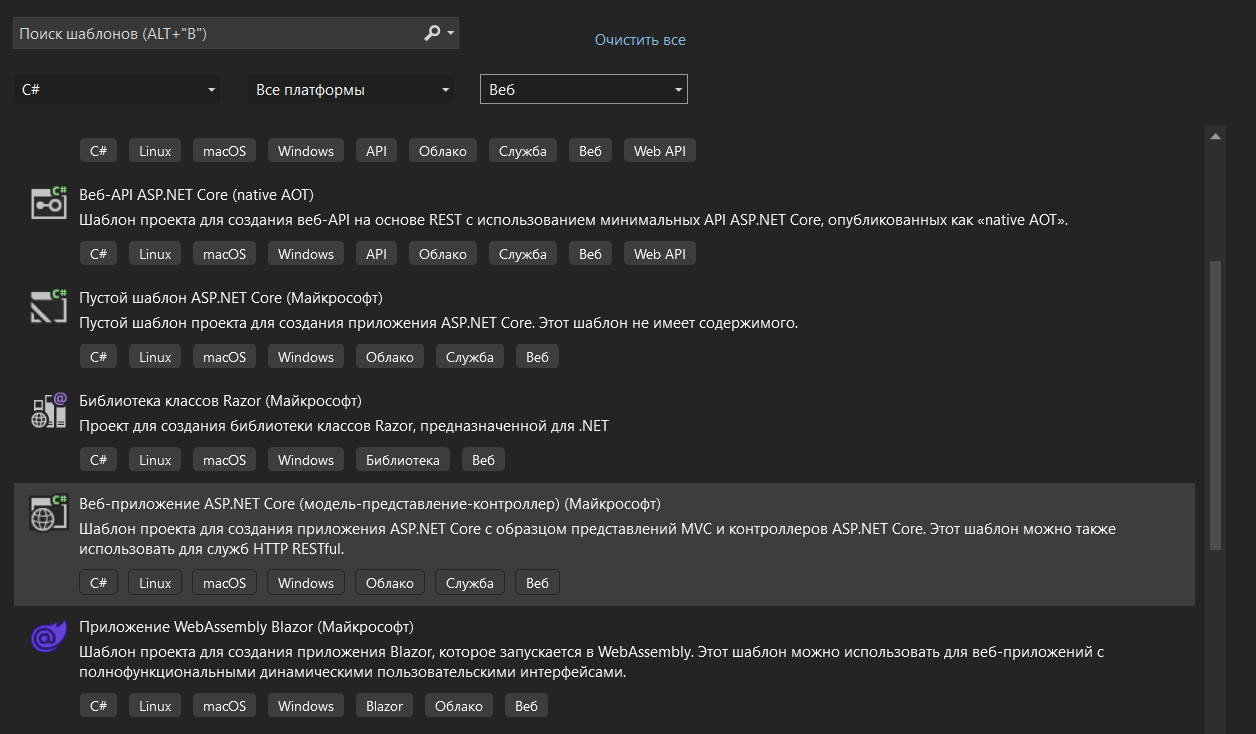


Рисунок 2 – Шаблоны

*MVC* – это архитектурный шаблон, который делит приложение на три основных компонента. Модель отвечает за данные и бизнес-логику, управляя взаимодействием с базой данных. Представление отвечает за отображение информации пользователю, формируя интерфейс. Контроллер обрабатывает входящие запросы, взаимодействует с моделью и выбирает, какое представление отобразить. Эта структура позволяет улучшить организацию кода, облегчает тестирование и масштабирование приложения, а также повышает его гибкость и поддержку.

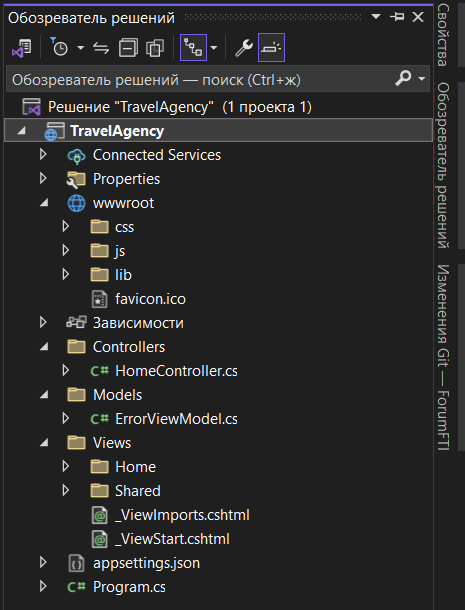


Рисунок 3 – Обозреватель решений

В проекте есть несколько ключевых папок для работы. Папка ***wwwroot***включает в себя директорию для стилей, папку для скриптов, обеспечивающих динамическое взаимодействие, и раздел для дополнительных файлов, которые могут упростить разработку при правильном использовании. В данном проекте будет применяться чистый *JavaScript.* Папка ***Controllers***отвечает за обработку входящих запросов. Здесь определяются конкретные URL-адреса, по которым пользователи попадают на страницы приложения при переходе по ссылкам или нажатии кнопок. Папка ***Models***предназначена для управления данными, которые хранятся и обрабатываются на сервере.

Папка ***Views*** содержит *HTML*-шаблоны, которые сервер возвращает после обработки запроса. В обозревателе решений можно выбрать файл ***Views/Shared/\_Layout.cshtml***. Файлы макетов, или мастер-страницы, позволяют задать единый шаблон для страниц *Razor*, создавая унифицированный вид приложения. Они используют код *Razor* и *HTML*, как и обычные страницы, но упрощают процесс разработки. С помощью мастер-страниц можно определить общие элементы, такие как меню и другие компоненты, а также подключить общие стили и скрипты. Это избавляет от необходимости указывать путь к файлам стилей на каждой странице и упрощает их изменение в будущем. Тег <*head*>, как показано на рисунке, предназначен для хранения служебной информации о странице и располагается первым внутри тега <*html*>, сразу перед <*body*>. Внутри <*head*> обычно находятся заголовок, ключевые слова, описание страницы и другие служебные данные.

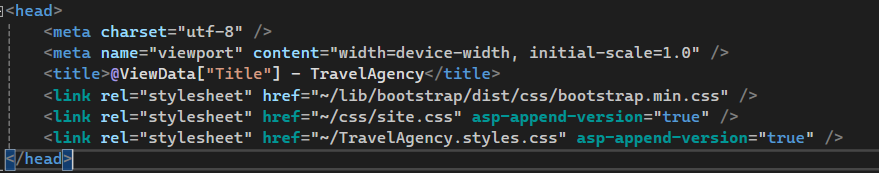


Рисунок 4 – тег <*head*>

Элемент <*header*> является семантическим и служит для отделения вводного содержимого или навигации от основного контента страницы или секционного блока. Например, он может использоваться для создания "шапки" с навигацией для всего сайта, а также для отображения имени автора и даты публикации в превью статьи или новости.

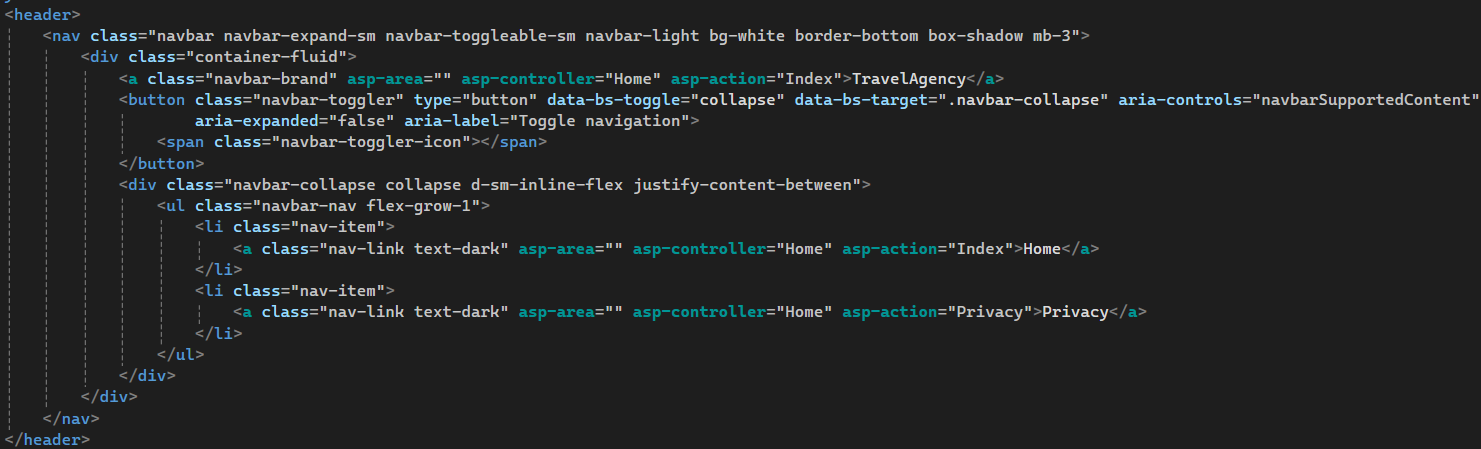


Рисунок 5 – тег <*header*>

Основной особенностью файлов макетов является использование метода @*RenderBody*(). Этот метод вставляет содержимое страниц *Razor*, которые используют данную мастер-страницу. Благодаря этому можно легко установить единообразный стиль оформления для всех страниц приложения.

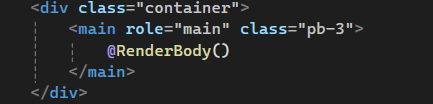


Рисунок 6 – функция @*RenderBody()*

Тег <*footer*> служит для создания футера сайта, располагающегося в нижней части страницы. В этом разделе обычно размещаются такие элементы, как контактная информация, нижнее меню и полезные ссылки. Скрипты чаще всего подключаются из внешних файлов с расширением .*js*. Для этого используется тег <*script*> с атрибутом *src*, в котором указывается путь к файлу. Рекомендуется размещать подключение внешних скриптов перед закрывающим тегом </*body*>, чтобы обеспечить правильную загрузку и работу страниц.

В папке ***Views/Shared*** также находятся файлы \_*ViewImports.cshtml* и \_*ViewStart.cshtml*. Файл \_*ViewImports.cshtml* содержит две строки кода, которые предназначены для глобального подключения пространств имен и дополнительных функций, применяемых ко всем представлениям проекта *ASP.NET Core MVC.* Это упрощает разработку и обеспечивает согласованность.

Файл \_*ViewStart.cshtml*, как правило, тоже содержит лишь пару строк. Он используется для указания файла макета (*layout)*, который будет применяться к текущему представлению. Это позволяет установить общий стиль для всех страниц, значительно упрощая поддержку и изменения в будущем.

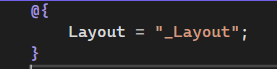


Рисунок 7 – содержимое файла \_*ViewStart.cshtml*

В разделе представлений есть еще одна важная папка — ***Home***. В ней хранятся представления, связанные с действиями контроллера ***HomeController***. Этот контроллер обычно отвечает за основные страницы веб-приложения, такие как главная страница, страница контактов и другие.

По умолчанию при запуске проекта вызывается именно этот контроллер, и действие ***Index***. Если необходимо изменить поведение по умолчанию, это можно сделать в файле ***Program.cs***, в соответствующих строках, как показано на рисунке.

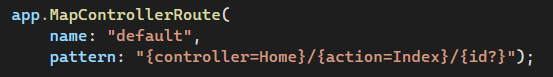


Рисунок 8 – фрагмент кода из файла *Program cs*

**Вывод:**

03.11.2024

Тема: Использование *Git* для контроля версий.

Цель работы: Научиться использовать *Git* для контроля версий.

Ход работы:

*Git* – это удобный инструмент для отслеживания изменений в проектах, командной работы и сохранения истории изменений, чтобы можно было вернуться к предыдущим версиям. В этой практике мы будем использовать *GitHub Desktop* для работы с проектами на локальном компьютере. Для этого понадобится аккаунт на *GitHub*. Чтобы начать работу с *GitHub*, нужно создать учетную запись и подтвердить свой адрес электронной почты. Каждому пользователю на *GitHub* присваивается личная учетная запись с уникальным именем пользователя и профилем.

**Как зарегистрировать аккаунт:**

1. Зайдите на [*github.com*](https://github.com/).
2. Нажмите «*Sign up*» (Регистрация).
3. Следуйте инструкциям для создания учетной записи.

При регистрации *GitHub* попросит подтвердить ваш адрес электронной почты. Без этого некоторые функции, такие как создание репозиториев, будут недоступны.

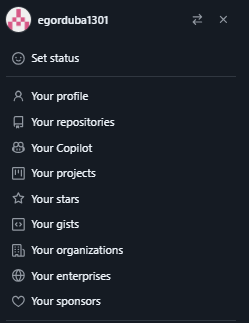


Рисунок 12 – ваш профиль.

Чтобы создать репозиторий, сначала откройте меню пользователя в правом верхнем углу и выберите пункт Your repositories. На открывшейся странице с репозиториями нажмите кнопку New. Появится страница для создания репозитория. Введите название репозитория — TravelAgency, как показано на рисунке 13. Остальные поля можно оставить без изменений.

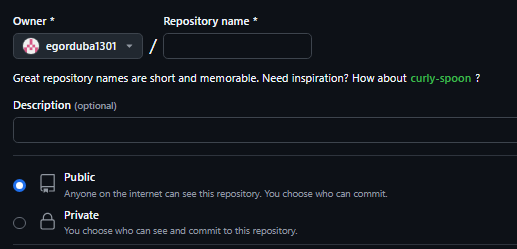


Рисунок 13 – создание удаленного репозитория

После создания удалённого репозитория нужно создать локальный репозиторий. Чтобы понять, как работать с GitHub Desktop, можно перейти на сайт [htmlacademy.ru](https://htmlacademy.ru/blog/git/github-desktop) или посмотреть информацию ниже, которая взята оттуда. После первого входа в GitHub Desktop программа попросит ввести ваш логин и пароль от GitHub.com. После этого у вас будет доступ ко всем репозиториям, которые есть в вашем профиле. **Создание репозитория**: если вы впервые используете GitHub, необходимо создать репозиторий для работы над проектом. На главном экране GitHub Desktop выберите пункт Create a New Repository on your hard drive.

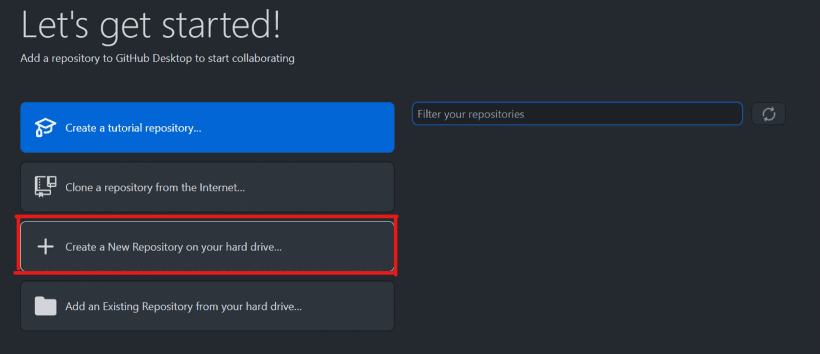


Рисунок 14 – начало работы

Нужно будет ввести название репозитория, его описание и выбрать папку на компьютере, куда будут сохраняться файлы.

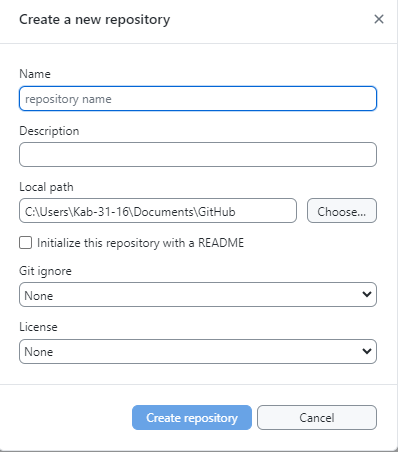


Рисунок 15 – создание локального репозитория

После этого нажмите на Create repository, подождите несколько секунд, и всё готово — на вашем компьютере появится папка, которую можно использовать для разработки проекта. **Клонирование репозитория**: если у вас уже есть репозиторий на GitHub, его можно клонировать. Клонирование означает, что вы скачиваете все файлы на свой компьютер, чтобы можно было их изменять и затем загружать обратно.

Для этого выберите Add -> Clone Repository....

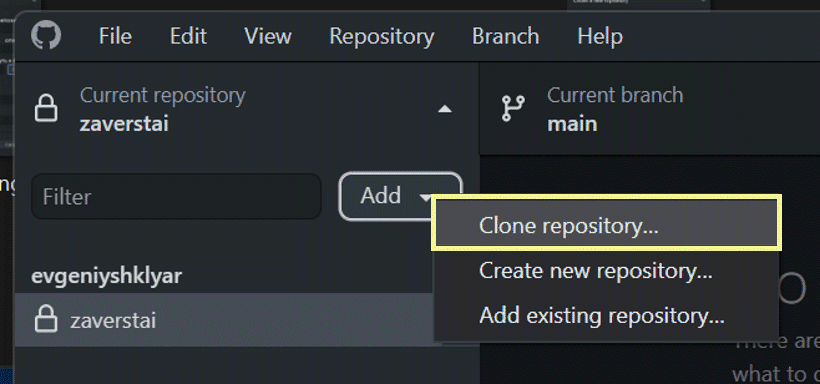
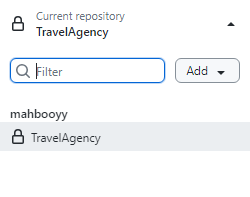


Рисунок 16 – клонирование репозитория

В открывшемся окне выбираем один из имеющихся репозиториев. В данном случае он называется *zaverstai,* но у вас может быть любой другой.

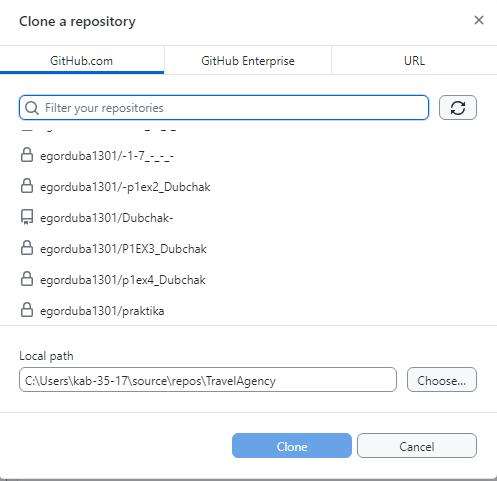


Рисунок17 – выбор репозитория

После этого файлы репозитория начнут скачиваться – если их много, то это займет некоторое время.

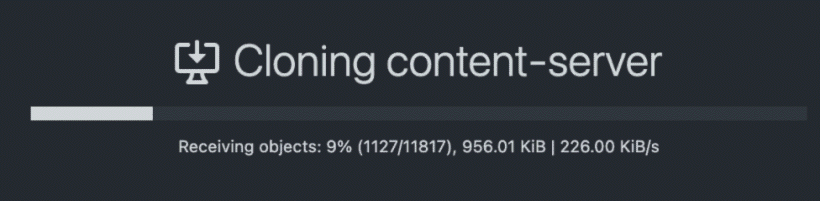


Рисунок 18 – скачивание файлов

Работа с репозиторием. Изменение файлов и их сохранение обратно: независимо от того, создали ли вы репозиторий или клонировали его, так выглядит GitHub Desktop с открытым репозиторием, в котором пока ничего не изменено.

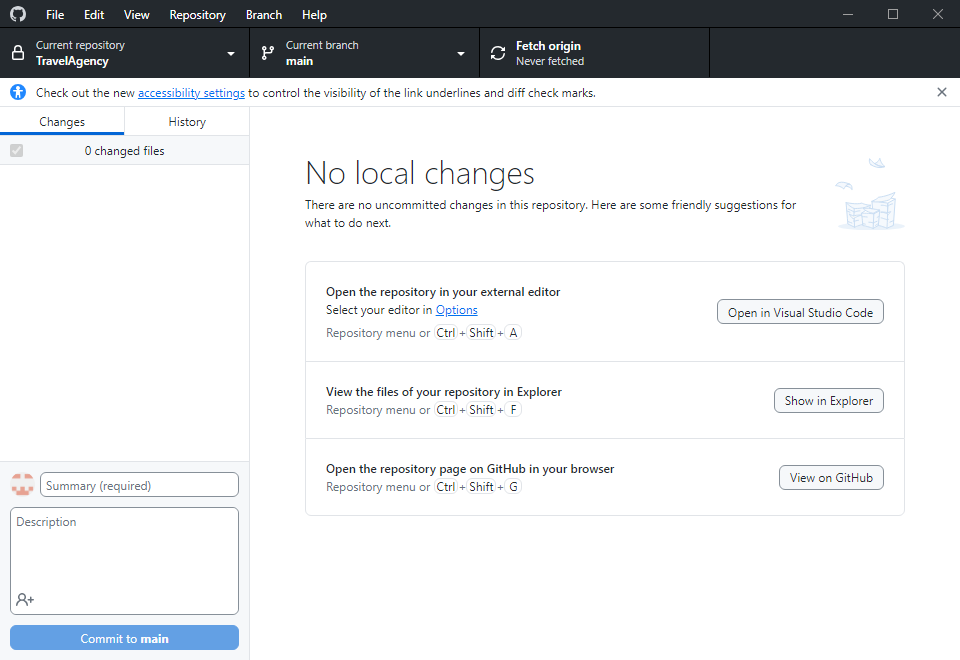


Рисунок 19 – вид приложения с открытым репозиторием

Слева находится поле для изменённых файлов, справа - служебная информация. В левом нижнем углу - поле для коммитов. Давайте добавим какой-нибудь файл. Например, я добавил в локальный репозиторий (скопировал в папку) файл index.html. Вы можете добавить текстовый файл. После добавления или изменения файла в окне GitHub Desktop сразу будет видно, что изменилось. Если мы добавили новый файл целиком, все строки будут отмечены плюсиками и будут зелёными. Это означает, что строки были добавлены в файл, и GitHub Desktop их раньше не видел.

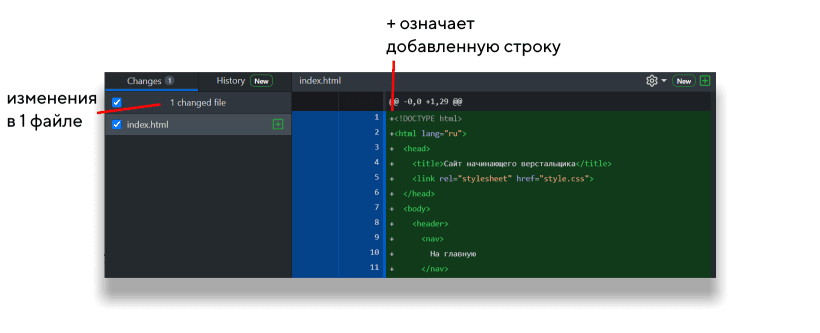


Рисунок 20 – вид при добавлении файла

Загружаем новый репозиторий на GitHub: после того как мы добавили код в новый репозиторий, нужно сделать коммит, то есть зафиксировать все сохранённые изменения и дать им название. Текст должен быть лаконичным и информативным, чтобы отражать суть изменений. Например, «добавляет имя наставника в Readme», «вводит функцию сортировки изображений», «правит ошибку в поиске городов на карте». Вводим название и нажимаем большую синюю кнопку Commit to main.

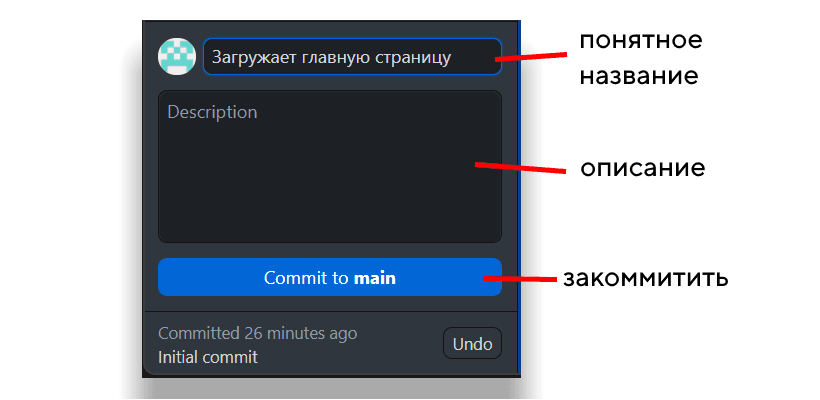


Рисунок 21 – создание коммита

Изменения, которые мы внесли и сохранили, пока локальны. Их нужно послать на *GitHub.* Чтобы опубликовать свежесозданный репозиторий на *GitHub*, нажмите *Publish repository*.

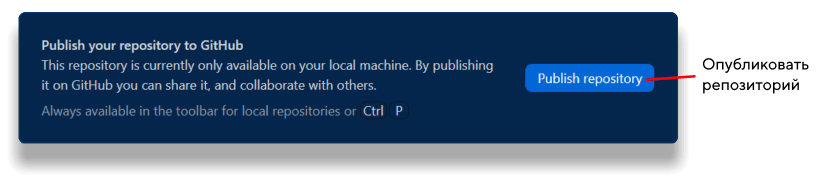


Рисунок 22 – опубликовать репозиторий

Появится подтверждение о публикации репозитория – проверяем название и описание, если нужно, ставим галочку о том, что код приватный, и публикуем.

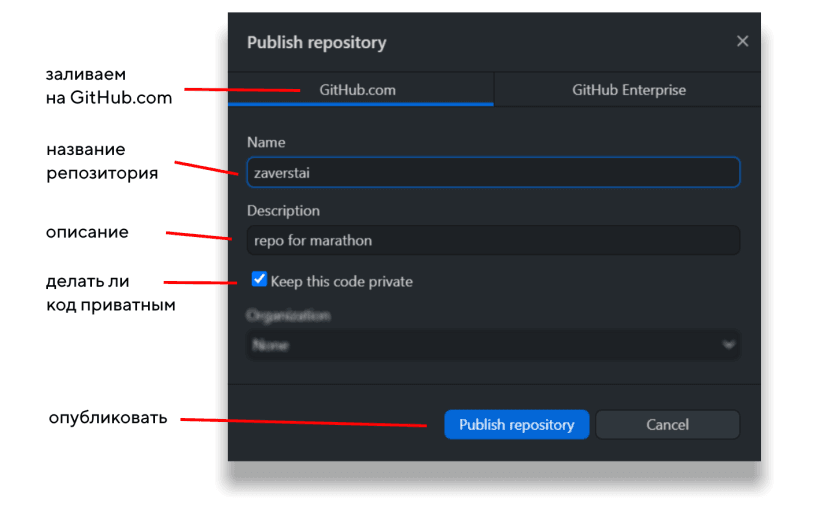


Рисунок 23 – подтверждение о публикации репозитория

Готово - после этого репозиторий появится в вашем профиле на GitHub.com. Добавляем код и коммитим изменения: репозиторий создан и загружен на GitHub, теперь нужно добавить немного кода. Когда вы допишете код в файлы, которые находятся в репозитории, вы сможете просмотреть все их изменения в окне GitHub Desktop. Например, если мы изменили «второй» на «третий» в тексте страницы, изменения сразу будут видны, и можно проверить, что всё исправленное будет загружено.

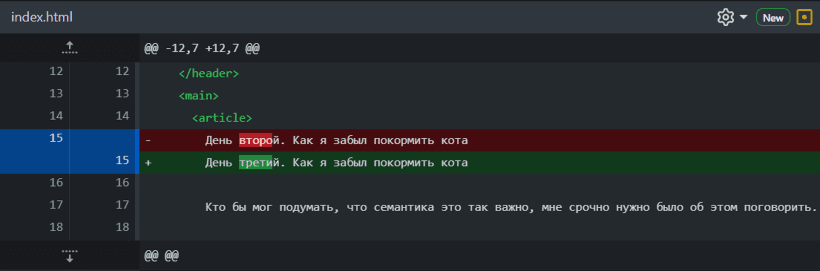


Рисунок 24 – просмотр исправлений

Дальше действуем по проверенной схеме – коммитим изменения.

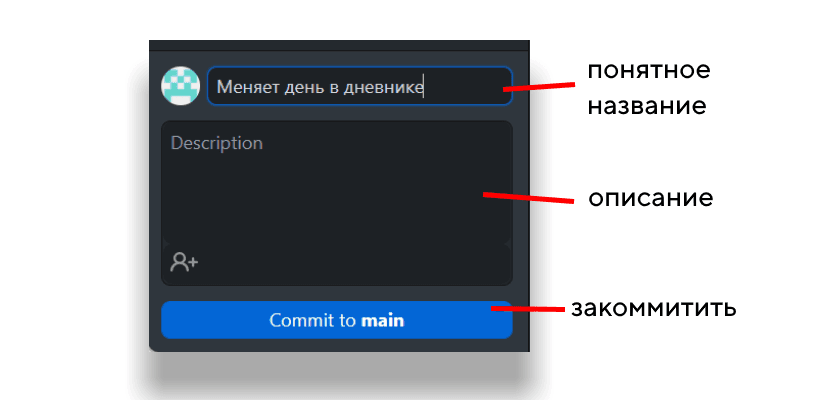


Рисунок 25 – коммит изменений

В центре главного экрана появится предложение запушить коммит в удалённый репозиторий. Соглашаемся и жмём *Push origin*.

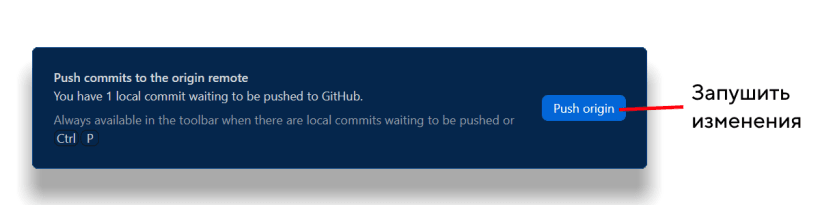


Рисунок 26 – пуш изменений

Готово. Теперь, если зайти на [*GitHub.com*](http://github.com/), в наш репозиторий, увидим изменённый файл, который мы только что отправили.

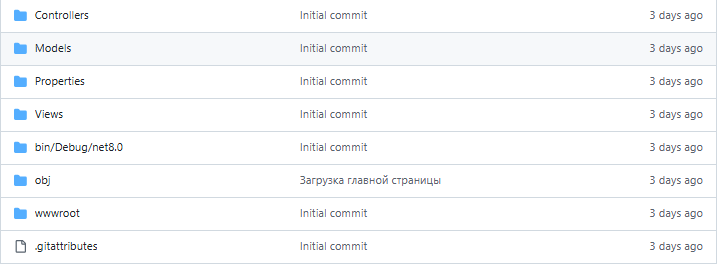


Рисунок 27 – проверка отправки изменений

Всё получилось – теперь вы можете создать или склонировать репозиторий, добавить туда файлы, опубликовать всё это на [*GitHub.com*](http://github.com/), не прикасаясь к консоли.

**04.11.2024**

**Тема:** Определение структуры MVC-приложения, взаимодействие моделей, контроллеров и представлений. Оформление отчетной документации.

**Ход работы:**

1. В созданном нами проекте на основе шаблона MVC будет использоваться чистая архитектура. Чистая архитектура — это рекомендации по организации системной архитектуры. Они были предложены Робертом С. Мартином (известным также как Дядя Боб) и основаны на ряде прежних архитектурных построений, таких как гексагональная архитектура, луковая архитектура и т. д. Это одно из основных правил для создания адаптируемого программного обеспечения (ПО), удобного в тестировании и поддержке. Каждая окружность соответствует различным составляющим ПО. Внешний уровень ПО является самым низким. К центру уровень повышается. В целом, чем ближе слой к центру, тем он менее подвержен изменениям. Вот иллюстрация чистой архитектуры, созданная Робертом Мартином:

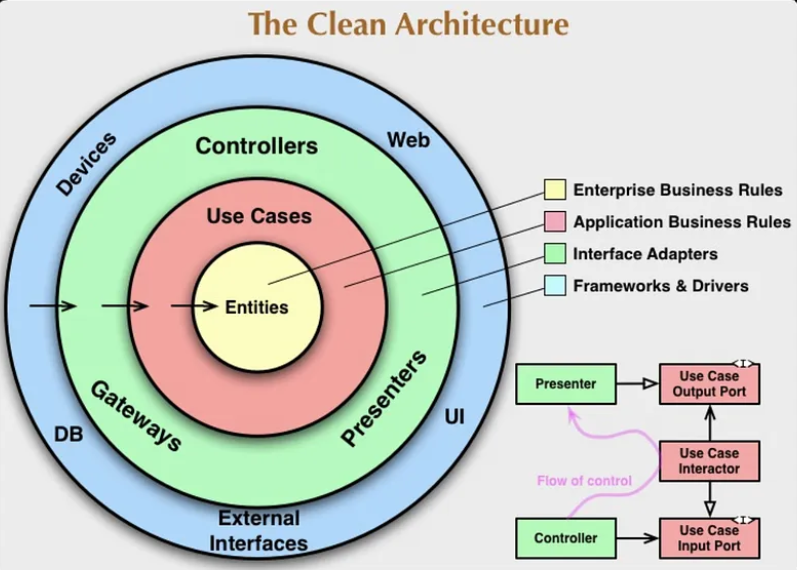


Рисунок 16. Схема чистой архитектуры.

1. В отношении чистой архитектуры существует правило зависимостей. Согласно этому правилу, зависимости исходного кода могут быть направлены только внутрь круговой схемы. Это означает, что компоненты из внутренней окружности могут вообще ничего не знать о компонентах из внешней, то есть внутренняя окружность никак не зависит от внешней. Направление черных стрелок на схеме соответствует этому принципу.

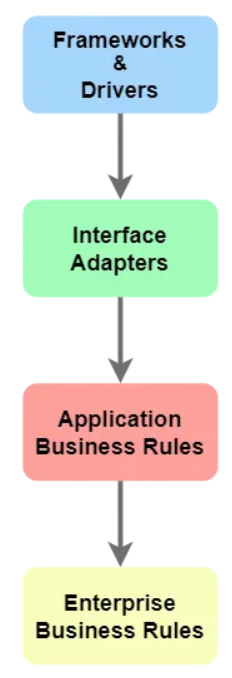


Рисунок 17. Правило зависимостей.

1. Основные компоненты чистой архитектуры - это сущности, сценарии, интерфейс адаптеры и фреймворки. Надо учитывать тот факт, что слои находящиеся ближе к центру становятся все менее и менее изменяемыми.
2. Для реализации данной архитектуры в нашем проекте сперва необходимо удалить папку *Models,* так как наши модели будут храниться в других местах и добавить 3 папки внутри которых будут находиться библиотеки классов: папку *DAL*, папку *Domain* и папку *Service*. Ниже результат проделанной работы:

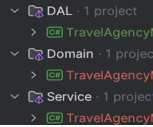


Рисунок 18. Созданные папки

**Вывод:** изучил понятие чистой архитектуры и определил структуру проекта на основе шаблона *MVC.*