Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Курсовая работа по теме «Веб-сервис “Track Everything”»

Выполнил: Научный руководитель

студент группы 753502 Рогов Максим Геннадьевич

Галкин Илья Викторович

Минск, 2019

**Содержание**

[Введение 4](#_Toc11465491)

[1 Сравнение с аналогами 5](#_Toc11465492)

[**1.1 Jira** 5](#_Toc11465493)

[**1.2 Trello** 6](#_Toc11465494)

[**1.3 Redbooth** 7](#_Toc11465495)

[2 Анализ используемых технологий 9](#_Toc11465496)

[**2.1 Объектно-ориентированное программирование в разработке веб-приложений** 9](#_Toc11465497)

[**2.2 Использование паттернов проектирования в разработке веб-приложений** 10](#_Toc11465498)

[2.2.1 Singleton 10](#_Toc11465499)

[2.2.2 Repository 11](#_Toc11465500)

[2.2.3 Factory method 11](#_Toc11465501)

[**2.3 Использование многоуровневой архитектуры в разработке веб-приложений** 13](#_Toc11465502)

[2.3.1 Понятие многоуровневой архитектуры 13](#_Toc11465503)

[2.3.3 Использование многоуровневой архитектуры в проекте 14](#_Toc11465504)

[3 Разработка программного средства 17](#_Toc11465505)

[**3.1 Описание структуры проекта**. 17](#_Toc11465506)

[**3.2 Описание классов и интерфейсов проекта**. 18](#_Toc11465507)

[**3.3 Описание работы программы** 21](#_Toc11465508)

[3.3.1 Запуск приложения 21](#_Toc11465509)

[3.3.2 Страница со списком проектов 22](#_Toc11465510)

[3.3.3 Страница добавления проекта 23](#_Toc11465511)

[3.3.4 Детальная страница проекта 24](#_Toc11465512)

[3.3.5 Страница изменения проекта 25](#_Toc11465513)

[3.4.6 Добавление связей между сущностями 25](#_Toc11465514)

[3.3.6 Удаление связанных сущностей 27](#_Toc11465515)

[4 Тестирование программного средства 29](#_Toc11465516)

[**4.1 Описание концепций тестирования программного средства** 29](#_Toc11465517)

[**4.2 Тестирование различных ситуаций программного взаимодействия** 30](#_Toc11465518)

[4.2.1 Тестирование настройки сервера 30](#_Toc11465519)

[4.2.2 Тестирование операций над записями 31](#_Toc11465520)

[4.2.3 Тестирование защиты от некорректных данных 33](#_Toc11465521)

[4.2.4 Тестирование связей между сущностями 34](#_Toc11465522)

[5 Руководство пользователя 36](#_Toc11465523)

[**5.1 Требования для разворачивания проекта** 36](#_Toc11465524)

[**5.1 Разворачивание проекта** 36](#_Toc11465525)

[**5.2 Дополнительные возможности** 36](#_Toc11465526)

[Заключение 38](#_Toc11465527)

[Список использованных источников 39](#_Toc11465528)

# **Введение**

В качестве темы курсового проекта был выбран веб-сервис для отслеживания задач, проектов и пользователей в системе. Смысл данного сервиса заключается в обеспечении удобства распределения задач между пользователями. Для пользователей доступны функции добавления, изменения и удаления проектов и задач, а также просматривания информации о пользователях.

При запуске приложения, пользователь оказывается на главной странице, где он может перейти на вкладку для работы с одной из трех основных сущностей: проект, задача, исполнитель. По своей функциональности все вкладки будут схожи между собой. На каждой странице будет доступны функции добавления, изменения, удаления и просматривания объектов. Основные различия будут в связях между этими сущностями. Например, проект может иметь несколько задач, когда в то же время, у задачи может быть только один проект. Эти связи можно будет увидеть на страницах с таблицами добавленных сущностей. Так, если брать предыдущий пример, мы сможет увидеть название проекта на странице со списком задач. Для того, чтобы задать эти связи, в приложении будет использоваться окно редактирования, где можно будет выбрать связи с уже существующими объектами. Приложение будет обладать простым и понятным интерфейсом с низким порогом вхождения. В плане использования системных ресурсов, при развертывании, приложение будет требовать малую производительность компьютера.

Целью курсовой работы является углубленное изучение языка C#, веб-фреймворка ASP.NET Core, методов проектирования и архитектурных решений при разработке веб-сервисов.

# **1 Сравнение с аналогами**

Подобный вид сервисов имеет огромное количество уникальных аналогов со своими особенностями. Рассмотрим некоторые из них.

## **1.1 Jira**

Рисунок 1.1 Интерфейс Jira

Jira – одна из самых известных систем для управления проектами (рисунок 1.1). В качестве основного элемента учета выступает задача. Задача создается в контексте проекта и может быть назначена на любого пользователя, который имеет доступ к данному проекту.

У задачи может быть один из трех статусов: «Будет выполнено», «В процессе выполнения», «Выполнено». При создании задачи по умолчанию ставится статус «Будет выполнено». Как только исполнитель принимается за выполнение, ставится статус «В процессе выполнения». Это избавляет от ситуаций, когда пользователь увидел невыполненную задачу и принимается выполнять ее, не зная, что кто-то другой уже работает над ней. После выполнения задача ставится в статус «Выполнено» и переводится на человека, проверяющего реализацию. В случае, если в задаче находится дефект, ее можно открыть повторно для исправления недочета.

При создании новой задачи пользователь может назначить теги, установить приоритет, добавить описание и указать пользователей для отслеживания выполнения.

Из дополнительных возможностей можно выделить удобный интерфейс для составления ежедневных, еженедельных и ежемесячных отчетов по работе, управление «спринтами», а также возможность создания собственных фильтров поиска.

Все это помогает организовать и облегчить работу над проектом командам, использующим данный сервис [1].

## **1.2 Trello**

Trello – известная система для управления проектами (рисунок 1.2). В отличии от Jira, данный сервис опирается на работу с задачами, поэтому не имеет такого же широкого спектра возможностей для работы с проектами.

Одна из основных особенностей данного сервиса это работа через визуальные карточки, которые можно упорядочивать, расставлять по приоритету и назначать на конкретного пользователя. Также карточки поддерживают комментарии, вложения, сроки выполнения, контрольные списки, цветные ярлыки и изображения на заднем плане. Структура Trello состоит из досок, на которых могут находится разные карточки. Примером использования этой возможности может быть организация проекта, когда каждая доска может отвечать за конкретный процесс, либо какую-либо общую задачу [2].

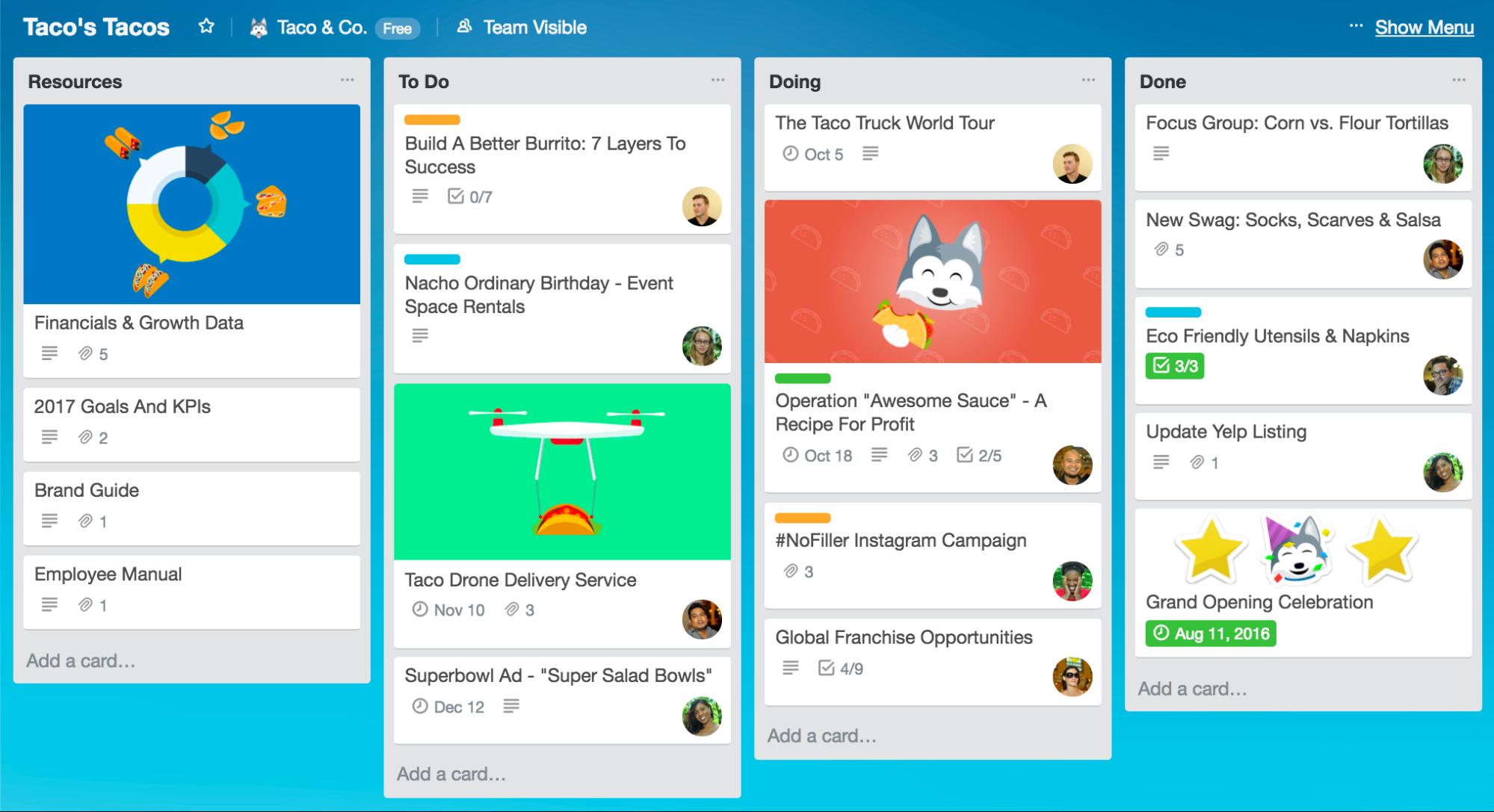


Рисунок 1.2 Интерфейс Trello

## **1.3 Redbooth**

Redbooth – очередной сервис для работы с проектами (рисунок 1.3). Имеет удобный интерфейс для отслеживания ошибок, планирования проектов и активности. Позволяет устанавливать крайние сроки и вехи проекта, следить за расходами проекта.

Из особенностей можно выделить поддержку чата между участниками проекта, либо по отдельному приглашению, удаленного общего доступа к экранам и синхронизацию с известными облачными хранилищами и сервисами.

Из заметных минусов: отсутствие тегов, подзадач, неудобная система оповещений [3].

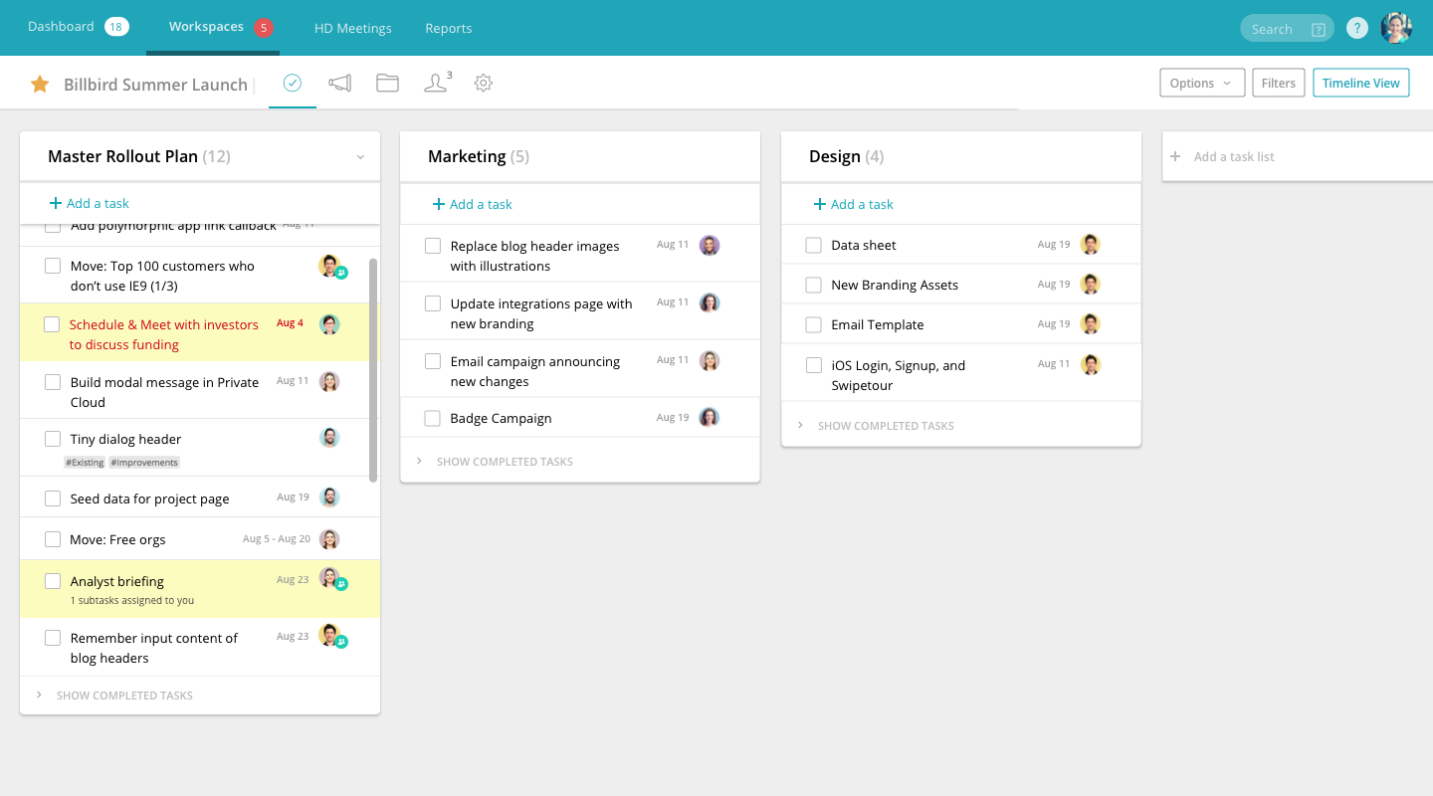


Рисунок 1.3 Интерфейс Redbooth

# **2 Анализ используемых технологий**

## **2.1 Объектно-ориентированное программирование в разработке веб-приложений**

Современные веб-разработчики используют различные технологии и подходы для создания продукта, который будет максимально актуален и легок в поддерживании на протяжении всего своего использования. Для того, чтобы добиться подобного результата разработчикам следует тщательно исследовать и изучить решаемую ими проблему и определится с выбором используемых технологий с принятием важных архитектурных решений. В большинстве случаев, заранее определенные подходы и решения, определенные для разработки, гарантируют, что продукт получится качественным и не потребует дополнительных затрат на доработку.

Объектно-ориентированное программирование является одним из подходов разработки программ. Данный подход позволяет организовывать и создавать различные модели информационных объектов, что помогает разработчикам легче понимать логику разрабатываемого продукта. А при работе с крупным проектом это является особенно важным фактором.

В веб-разработке объектно-ориентированное программирование используется повсеместно. Благодаря простой организации объектов, модели легко ассоциируются с реальными объектами тем самым помогая общаться разработчикам с теми, кто не имеют отношения к коду. Такой подход предоставляет возможность создавать множество однотипных объектов избегая повторения кода, как и создание уникального объекта со своими уникальными свойствами.

Три парадигмы ООП определяют разработку выпускаемой программы. Механизм наследования позволяет создавать новые типы объектов, основываясь на уже созданных моделях. Механизм инкапсуляции позволяет защитить важные данные от несанкционированного доступа. Механизм полиморфизма дает программам возможность не утруждать себя изучением вида объектов. Все это широко используется при разработке веб-сервисов и во многом это удобнее функционального подхода программирования.

Среди минусов ООП можно выделить не самую быструю скорость выполнения кода и неэффективное распределение памяти по сравнению с функциональным подходом программирования. Хотя в современной разработке, зачастую, это не столь важно.

## **2.2 Использование паттернов проектирования в разработке веб-приложений**

Паттерн проектирования – это проверенный подход к решению определенной задачи, связанной с объектно-ориентированным программированием. Существует множество паттернов для разных ситуаций и многие из них не привязаны к конкретным языкам программирования. Но также существует паттерны для отдельных языков, которые дополняют эти языки новыми возможностями. В неумелых руках паттерн может принести немало проблем, а при правильном использовании такие подходы помогают быстро и правильно решать задачи реализации. Рассмотрим несколько из них.

### 2.2.1 Singleton

Основное назначение паттерна Singleton является гарантия того, что будет создан только лишь один экземпляр класса и к нему будет предоставлен глобальный доступ из любого места в программе. Различие синглтона и глобальной переменной лишь в том, что синглтон создает объект только при необходимости, когда глобальная переменная создается при первом же упоминании.

Основной смысл реализации синглтона заключается в создании в классе статической переменной, указывающей на единственный экземпляр этого класса. Этот объект можно получить, либо инициализировать только из специального метода, в котором прописана логика единственности объекта.

Хоть данный шаблон проектирования и является одним из самых простых, но это не исключает того факта, что он является одним из самых используемых паттернов. Потребность в хранении единственного объекта на класс не обходит и веб-разработку. Наиболее подходящий пример — это объект веб-хоста. Нет никакого смысла в том, чтобы их было несколько и в то же время этот объект может потребоваться в любой части кода, что идеально вписывается в рамки данного паттерна.

### 2.2.2 Repository

Является одним из самых популярных шаблонов проектирования при работе с данными. Основная идея заключается в том, чтобы изолировать логику обращений к базе данных от бизнес логики проекта. Это дает гибкость при работе с различными типами подключений, позволяя заменять реализации обращений в базу данных, не влияя на работу других частей приложения. Хорошей практикой считается использование отдельного репозитория для каждой бизнес-сущности, что помогает не загружать классы лишней логикой.

Подобный шаблон повсеместно используется в веб-разработке. Классы-репозитории, содержат в себе логику работы с базой данных и ничего более. Таким образом, не создаются никакие зависимости от других частей приложения, что позволяет легко заменять классы между собой.

Зачастую, используется в паре с паттерном Unit of Work, который объединяет все репозитории в одном классе. Этот помогает быть уверенным в том, что используется один контекст данных и позволяет упростить работу с базой данных, предоставляя одну входную точку.

### 2.2.3 Factory method

Относится к порождающему типу паттернов. Данный шаблон проектирования предоставляет интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс инстанцировать. Иными словами, Фабрика делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Паттерн используется, когда:

* Класс не имеет информации о том, какой тип объекта он должен создать
* Класс передает ответственность по созданию объектов наследникам.
* Необходимо создать объект в зависимости от входящих данных.

В веб-разработке зачастую используется механизм инверсии управления (Inversion of Control), который работает через данный шаблон проектирования [4].

## **2.3 Использование многоуровневой архитектуры в разработке веб-приложений**

### 2.3.1 Понятие многоуровневой архитектуры

Многоуровневая архитектура сосредоточена на иерархическом распределении отдельных частей системы при помощи эффективного разделения отношений. Каждая часть соотносится с определенным уровнем, для каждого уровня заданы выполняемые им функции, уровни выстроены в стековую структуру (то есть находятся один поверх другого). Наиболее распространенная реализация – использование трех уровней. Трехуровневая архитектура включает уровень представления, уровень бизнес-логики и уровень доступа к данным. При этом уровень представления считается высшим, за ним идет уровень бизнес-логики, а за уровнем бизнес-логики – уровень доступа к данным [5].

2.3.2 Смысл многоуровневой архитектуры

Зачастую при разработке программного средства разработчику требуется, чтобы логика отдельных компонентов программы была независима между собой. Это помогает изменять отдельные слои программы, не влияя на работоспособность остальных частей приложения, тем самым упрощая дальнейшую поддержку продукта. В случае одноуровневых приложений любое изменение в структуре программы повлечет за собой изменение всего проекта.

В случае трехуровневой архитектуры, приложение делится на три независимых слоя, которые выполняют определенную роль. Слой доступа к данным отвечает за обращения к базе данных и только. Слой бизнес-логики содержит в себе логику работы с данными. Например, это могут быть какие-то математические расчеты, либо определение какие данные должны быть извлечены из базы данных. Слой представления отвечает за взаимодействие с пользователем, ввод и вывод информации.

Благодаря тому, что слои выстроены в стековую структуру, слои нижнего уровня не могут иметь доступ к слоям верхнего уровня. Например, слой доступа к базе данным не имеет доступа к слою бизнес-логики, однако, слой бизнес-логики использует функциональность слоя доступа к базе данным.

### 2.3.3 Использование многоуровневой архитектуры в проекте

Данный проект реализован на основе трехуровневой архитектуры с использованием модулей, к которым имеют доступ все слои приложения. Подобные модули удобны тем, что не нарушают целостность уровней и позволяют использовать одну и ту же функциональность, не внедряя никаких зависимостей. В данном приложении таким модулем является класс логгера, который необходим в использовании на каждом слое приложения.

Для того, чтобы передавать данные между слоями используются специальные конверторы, которые позволяют использовать в контексте слоя только те типы, которые описаны на этом самом уровне. Ниже приведен листинг кода программы, в котором реализован процесс подобной конвертации.

public ProjectViewModel Convert(ProjectBO projectBo)

{

if (projectBo != null)

{

var project = new ProjectViewModel

{

Id = projectBo.Id,

Name = projectBo.Name,

Shortname = projectBo.Shortname,

Description=ProjectBo.Description,

Status = projectBo.Status,

CreationDate=projectBo.CreationDate,

Tasks = null

};

if (projectBo.Tasks != null)

{

var taskBoToViewConverter = new TaskBOConverter();

var tempTasks = new List<TaskViewModel>();

foreach (var taskBo in projectBo.Tasks) {

var task = taskBoToViewConverter.Convert(taskBo);

task.Project = null;

task.Executors = null;

tempTasks.Add(task);

}

project.Tasks = tempTasks;

}

return project;

}

return null;

}

Метод принимает тип ProjectBO, который находится на слое бизнес-логики и возвращает тип ProjectViewModel, который уже находится на слое представления. Зачастую приходится писать такие методы для каждой сущности приложения, однако, это не всегда практично. В данном проекте используется библиотека AutoMapper, которая берет на себя ответственность за конвертирование данных из одного типа в другой. По умолчанию она соотносит поля классов по названиям, однако, это не всегда применимо. Для того, чтобы задать свои правила конвертирования, создаются объекты-профили, которые загружаются в класс конвертора.

Ниже приведен листинг кода программы, в котором описываются правила конвертации для AutoMapper.

public TaskProfile()

{

CreateMap<Task, EFTask>()

.ForMember(p => p.TaskWorkers, opt =>

opt.MapFrom(u => u.Executors))

.AfterMap((model, entity) =>

foreach (var entityTaskWorker in entity.TaskWorkers) entityTaskWorker.Task = entity;

});

CreateMap<Worker, EFTaskWorker>()

.ForMember(entity => entity.Worker, opt => opt.MapFrom(model => model))

.ForMember(entity => entity.WorkerId, opt => opt.MapFrom(model => model.Id));

CreateMap<TaskBO, Task>()

.ForMember(entity => entity.Executors, opt => opt.MapFrom(model => model.Workers))

.ForMember(p => p.Estimation, opt => opt.MapFrom(t => t.Estimation.Ticks));

CreateMap<Task, TaskBO>()

.ForMember(entity => entity.Workers, opt => opt.MapFrom(model => model.Executors))

.ForMember(p => p.Estimation, opt => opt.MapFrom(t => TimeSpan.FromTicks(t.Estimation)));

CreateMap<EFTask, Task>()

.ForMember(v => v.Executors, opt => opt.MapFrom(u => u.TaskWorkers.Select(y => y.Worker).ToList()));

}

# **3 Разработка программного средства**

## **3.1 Описание структуры проекта**.

**Automation** – директория, содержащая файлы автоматизации.

**Documents** – директория, в которой хранится документация по проекту.

**.gitattributes** – git-аттрибуты.

**.gitignore** – файл, содержащий указания для игнорирования файлов в git.

**README.md** – руководство пользователя в удобном формате.

**version.txt** – версия программы.

**Sources** – директория, содержащая весь код программы.

* **Sources/TrackEverything.BusinessLogic** – содержит код бизнес-логики проекта.
* **Sources/TrackEverything.Database** – содержит скрипты для базы данных.
* **Sources/TrackEverything.EFStorage** – содержит реализацию паттерна репозиторий через Entity Framework.
* **Sources/TrackEverything.Storage** – содержит реализацию паттерна репозиторий через ADO.NET.
* **Sources/TrackEverything.Storage.Core** – содержит общие данные для EFStorage и Storage.
* **Sources/TrackEverything.Tools** – содержит дополнительную функциональность использующуюся во всем проекте.
* **Sources/TrackEverything.View** – содержит точки доступа к сервису.

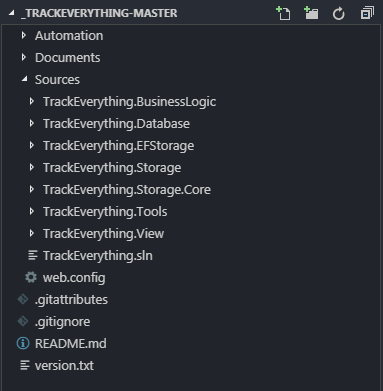


Рисунок 3.1 Структура проекта

## **3.2 Описание классов и интерфейсов проекта**.

**IRepository** – классы, реализующие данный интерфейс, гарантируют, что будут содержать методы добавления, редактирования, чтения и удаления записей из базы данных.

**IProjectRepository, ITaskRepository, IWorkerRepostiory** – интерфейсы, унаследованные от IRepository, дополняющие логику базового интерфейса в зависимости от сущности.

**IUnitOfWork** – интерфейс, содержащий в себе интерфейсы репозиториев.

**ProjectRepository, TaskRepository, WorkerRepository** – классы репозиториев, реализующие соответствующие интерфейсы через ADO.NET провайдер.

**EFProjectRepository, EFTaskRepository, EFWorkerRepository** – классы репозиториев, реализующие соответствующие интерфейсы через Entity Framework.

**Project, Task, Worker** – классы основных сущностей приложения.

**DbContext** – класс контекста базы данных. Используется Entity Framework для работы с базой данных. Содержит в себе описание связей между сущностями.

**IConverter** – интерфейс, содержащий метод конвертации моделей между слоями приложения. Реализуется всеми конверторами, используемыми в приложении.

**IService** – интерфейс, описание методов бизнес-логики приложения.

**ProjectService, TaskService, WorkerService** – классы, реализующие интерфейс IService. Хранят в себе реализацию бизнес логики приложения.

**ProjectController, TaskController, WorkerController** – классы входных точек приложения. Содержат методы принимающие обращения к серверу через HTTP запросы.

**CustomLogger** – класс логгера, используемого во всем приложении.

**BLServiceModule, ViewServiceModule** – классы реализующие интерфейс NinjectModule. Переопределяют метод Load используемый Ninject для регистрации абстракций в IoC-контейнере.

**Startup** – является входной точкой в приложение ASP.NET Core. Этот класс производит конфигурацию приложения, настраивает сервисы, которые приложение будет использовать и устанавливает компоненты для обработки запроса [6].

**Program** – класс с которого начинается выполнение проекта.

Код класса ViewServiceModule:

public class ViewServiceModule : NinjectModule

{

public override void Load()

{

Bind(typeof(IService<WorkerBO>)).To(typeof(WorkerService));

Bind(typeof(IService<TaskBO>)).To(typeof(TaskService));

Bind(typeof(IService<ProjectBO>)).To(typeof(ProjectService));

Bind<IMapper>().ToMethod(AutoMapper).InSingletonScope();

Bind<ILogger>().To<CustomLogger>()

.WithConstructorArgument("name", x => x.Request.ParentContext.Request.Service.FullName);

Bind(typeof(IConverter<WorkerBO, WorkerViewModel>)).To(typeof(WorkerBOConverter));

Bind(typeof(IConverter<WorkerViewModel, WorkerBO>)).To(typeof(WorkerViewConverter));

Bind(typeof(IConverter<TaskBO, TaskViewModel>)).To(typeof(TaskBOConverter));

Bind(typeof(IConverter<TaskViewModel, TaskBO>)).To(typeof(TaskViewConverter));

Bind(typeof(IConverter<ProjectBO, ProjectViewModel>)).To(typeof(ProjectBOConverter));

Bind(typeof(IConverter<ProjectViewModel, ProjectBO>)).To(typeof(ProjectViewConverter));

}

private IMapper AutoMapper(IContext context)

{

Mapper.Initialize(config =>

{

config.ConstructServicesUsing(type => context.Kernel.Get(type));

config.AddProfile(new WorkerProfile());

config.AddProfile(new TaskProfile());

config.AddProfile(new ProjectProfile());

});

return Mapper.Instance;

}

}

Как видно из кода, данный класс привязывает конкретные классы к абстракциям и добавляет их в IoC-контейнер. Особое внимание заслуживает интерфейс IMapper, который привязывается к методу возвращающий экземпляр типа Mapper. В этом методе реализована логика инициализации объекта, в которой явно прописывается конфигурация конвертора.

## **3.3 Описание работы программы**

### 3.3.1 Запуск приложения

После запуска приложения перед пользователем открывается главная страница приложения. Оттуда ему предоставляется возможность перейти на вкладку со списком проектов, задач и сотрудников (рисунок 3.2).

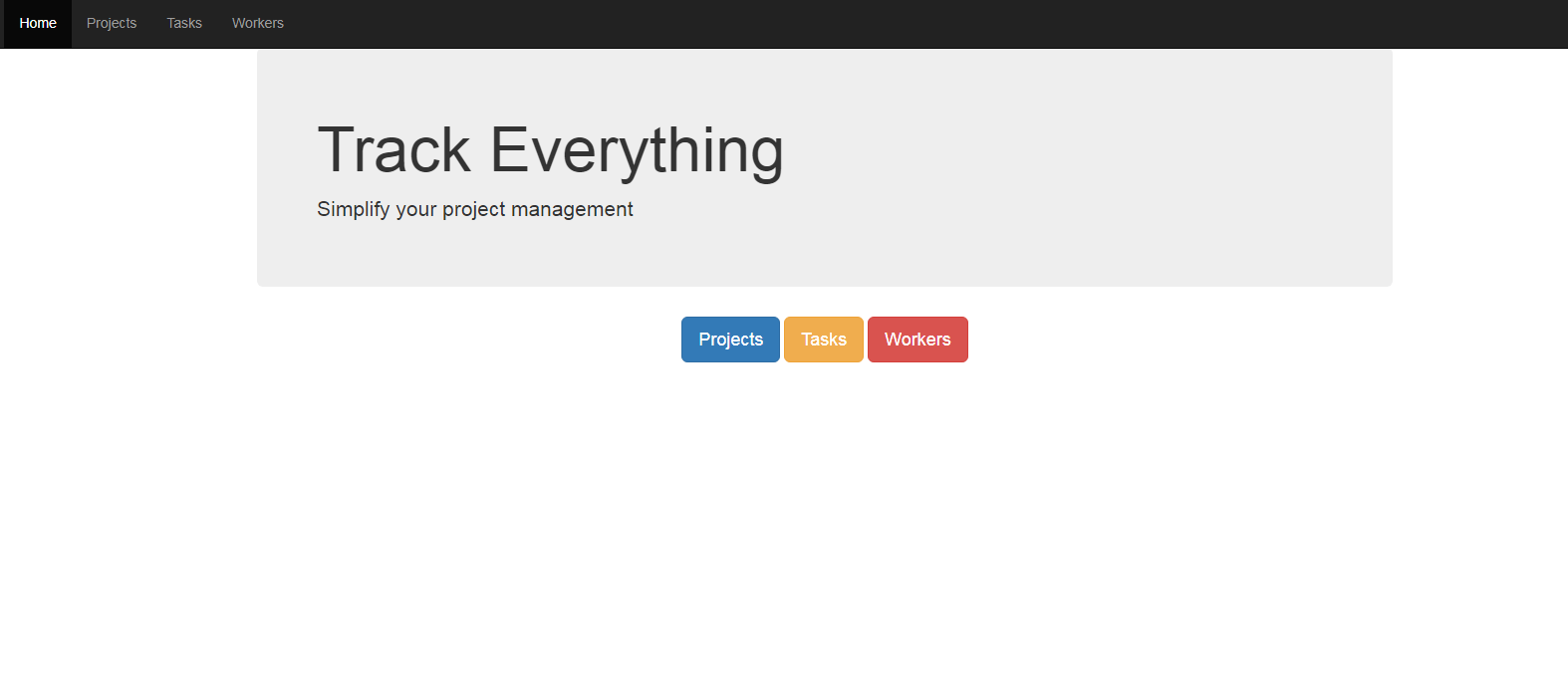


Рисунок 3.2 Главная страница приложения

При первом запуске программы, в корне проекта появится лог-файл, в котором будут записываться все операции, проводимые на сервере. Например, при переходе на вкладку со списком проектов, в лог-файле появится запись «DATABASE: Uploaded all projects».

### 3.3.2 Страница со списком проектов

При нажатии кнопки «Projects» открывается окно со списком проектов, если те уже есть в базе данных (рисунок 3.3).

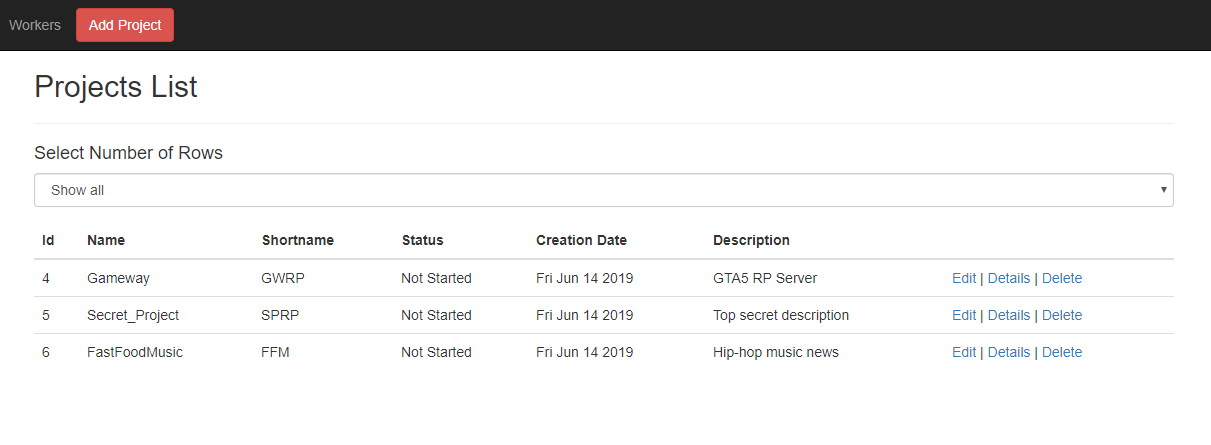


Рисунок 3.3 Страница со списком проектов в системе

Перед пользователем появилась таблица с данными о проектах, которые были добавлены в систему. Данные в таблице всегда будут отсортированы от первых добавленных к последним. Каждую запись о проекте можно отредактировать, нажав кнопку «Edit» на той же строке. Для того, чтобы открыть окно с полной информацией о проекте, необходимо нажать на кнопку «Details». Если же пользователю необходимо удалить проект, ему следует нажать кнопку «Delete».

Сверху от таблицы, пользователь может выбрать количество элементов, отображаемых на странице. Приложение рассчитано на страницы по 5, 10, 15, 20 и 50 объектов.

В самом верху страницы, на навигационной панели, можно заметить кнопку «Add project», которая перенаправит пользователя на страницу добавления нового проекта.

### 3.3.3 Страница добавления проекта

При нажатии кнопки «Add project» пользователя перенаправит на страницу с формой добавления нового проекта (рисунок 3.4).



Рисунок 3.4 Форма добавления нового проекта

Теперь пользователь может ввести данные для добавления проекта. Те поля, которые необходимо ввести помечены знаком «\*». Причем приложение рассчитывает, что пользователь будет вводить корректные данные. Так, сервис не примет, если пользователь введет в поле «Name», либо в после «Shortname» строку, содержащую цифры, либо специальные символы (рисунок 3.5) В случае с полем «Description» разрешено использование любых символов.

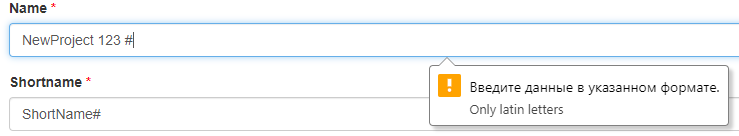


Рисунок 3.5 Валидация поля «Name»

Стоит отметить, что валидация полей используется на всех слоях приложения. На данном рисунке показана валидация на клиентском уровне, однако, в то же время, такая запись не пройдет и на серверном уровне.

После успешного создания проекта пользователя перенаправит на детальную страницу созданного проекта.

### 3.3.4 Детальная страница проекта

После создания проекта, либо при переходе через кнопку на странице со списком, перед пользователем появляется страница с детальной информацией о проекте (рисунок 3.6).

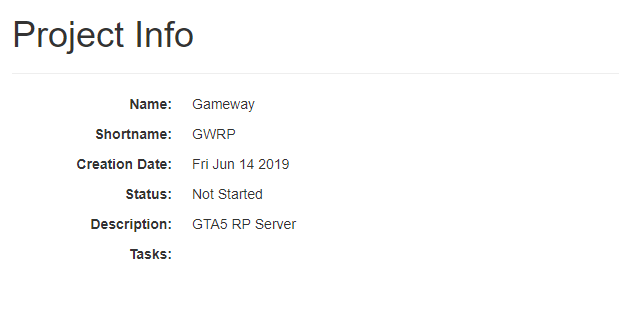


Рисунок 3.6 Детальная информация о созданном проекте

На странице отображается вся введенная пользователем информация. Стоит обратить внимание на поле «Status». При создании проекта, данное поле всегда будет выставлено в значение «Not Started». Для того, чтобы изменить статус проекта, нужно перейти на страницу изменения проекта и выставить другое значение.

Поле «Tasks» показывает, какие задачи связанны с проектом. Для того, чтобы в нем появились запись, необходимо создать новую задачу и выбрать для нее текущий проект.

### 3.3.5 Страница изменения проекта

Для того, чтобы изменить данные о проекте, необходимо нажать на кнопку «Edit» в таблице всех проектов (рисунок 3.7).



Рисунок 3.7 Форма изменения проекта

Форма изменения проекта отличается от формы добавления наличием поля «Status». При загрузке страницы, с сервера запрашиваются данные о выбранном проекте и все поля заполняются автоматически. На данной форме присутствует вся та валидация полей, которая есть на форме добавления. При успешном выполнении запроса на изменение данных, пользователя также перенаправляет на детальную страницу о проекте.

### 3.4.6 Добавление связей между сущностями

В целом, страницы просмотра, добавления и изменения для задач и сотрудников ничем не отличаются между собой по функциональности. Однако, на них можно отчетливо увидеть связь между сущностями.

При добавлении сотрудника, пользователь может выбрать те задачи, которые он хочет присвоить сотруднику и в то же время, на странице добавления задачи, можно указать, какие сотрудники будут назначены на создаваемую задачу (рисунок 3.8).

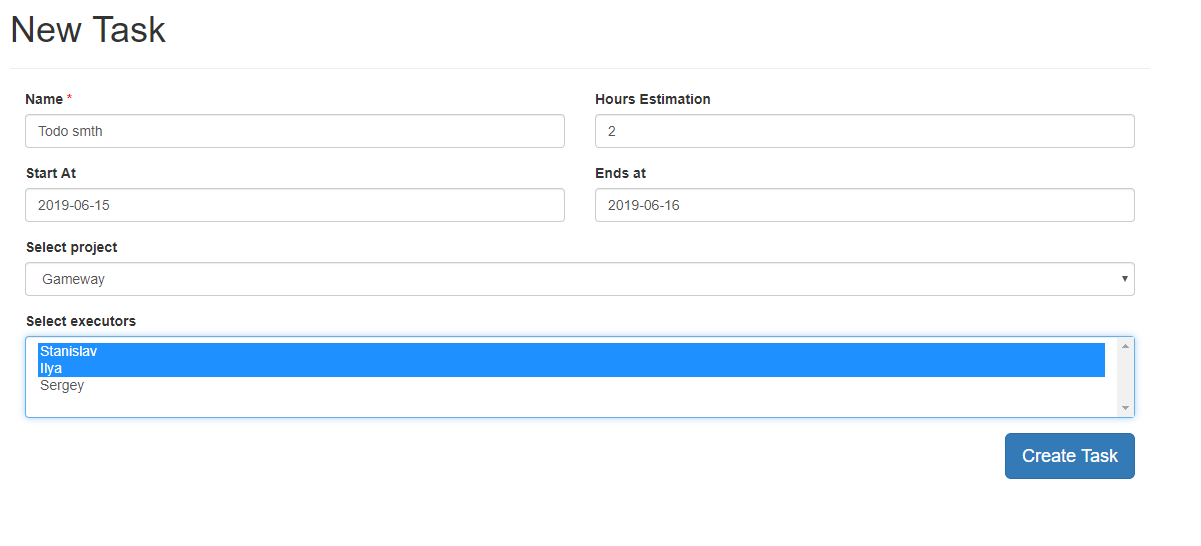


Рисунок 3.8 Форма добавления задачи

Как видно по рисунку, на странице автоматически подгружается список сотрудников и проектов. После этого пользователь может выбрать несколько исполнителей и проект, к которому относится данная задача. Такая же логика и для страницы добавления сотрудника: при загрузке страницы подгружаются все доступные задачи, которые можно назначить исполнителю.

После добавления задачи пользователя перенаправляет на детальную страницу созданной задачи, где в поле «Executors» можно увидеть имена сотрудников, которые были указаны при создании. Также и для поля «Project» (рисунок 3.9).

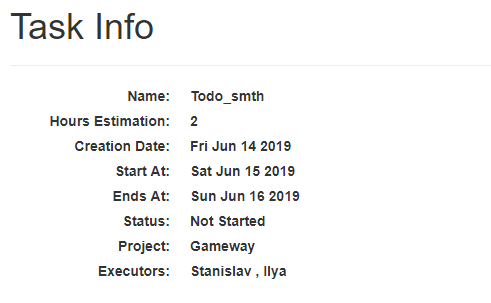


Рисунок 3.9 Детальная информация о задаче, после сохранения.

Теперь на детальной странице указанного проекта в поле «Tasks» можно увидеть название созданной задачи (рисунок 3.10).

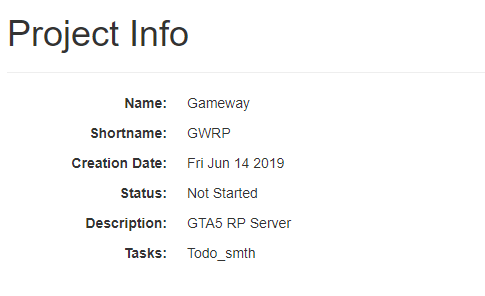


Рисунок 3.10 Детальная информация о проекте, после добавление связанной задачи

Соответственно, на детальных страницах указанных пользователей также будет отображена информация о созданной задаче.

### 3.3.6 Удаление связанных сущностей

В том случае, если пользователь захочет удалить проект, на который уже ссылаются задачи, то при нажатии на кнопку удаления произойдет механизм каскадного удаления проекта вместе с зависимыми от него задачами. Причем, задачи, удаленные таким образом, пропадут из назначенных у соответствующих сотрудников.

Во всех других случаях удаления, у всех связанных объектов будет пропадать ссылка на удаленную сущность.

# **4 Тестирование программного средства**

## **4.1 Описание концепций тестирования программного средства**

Одним из критериев успеха любого программного продукта является фактор того, что выпускаемый продукт был максимально протестирован. Необходимо предусмотреть все варианты использования программы и предугадать все возможные ситуации в процессе эксплуатации продукта.

Сетевое программное средство должно полностью соответствовать всем базовым канонам данной сферы. Банально, программа должна обработать все возможные ситуации и ни в коем случае не должна вызывать аварийное завершение. Также программа должна быть защищена от возможности нарушить стандартную логику работы. Такие ситуации отпугивают пользователей и это чревато тем, что программный продукт не будет использоваться в дальнейшем.

Тестирование продукта проходит в несколько этапов:

1. Системное тестирование – тестирование всех программы в целом на работоспособность.
2. Модульное тестирование - тестирование всех модулей программы по-отдельности.
3. Тестирование на выявление всех неполадок и неординарных ситуаций.
4. Тестирование на совместимость с различными браузерами.
5. Нагрузочное тестирование – тестирование с целью выявить пик максимальной нагрузки на программу.
6. Стресс-тестирование – тестирование с целью выявления, что происходит с программой при переходе в нерабочее состояние
7. Тестирование локализации – проверка перевода программного средства.

В процессе разработки программы были протестированы все модули данного проекта. Каждый модуль тестировался как отдельно, так и в совокупности со всей программой. Была обеспечена защита от неправильного использования и аварийных завершений программы.

## **4.2 Тестирование различных ситуаций программного взаимодействия**

### Тестирование настройки сервера

* **Тест 1**

**Тестовая ситуация:**

Пользователь запускает приложение для использования.

**Исходный набор данных:**

Пользователь выполнил все инструкции из руководства пользователя и развернул приложение.

**Ожидаемый результат:**

В браузере открывается главная страница приложения.

**Фактический результат:**

В браузере открывается главная страница приложения.

* **Тест 2**

**Тестовая ситуация:**

Пользователь запускает приложение для использования.

**Исходный набор данных:**

Пользователь не настроил базу данных для ее использования в приложении и развернул проект.

**Ожидаемый результат:**

В браузере открывается главная страница приложения. Все страницы со списками данных возвращают 500 статусный код.

**Фактический результат:**

В браузере открывается главная страница приложения. Все страницы со списками данных возвращают 500 статусный код.

* **Тест 3**

**Тестовая ситуация:**

Пользователь открывает страницу с задачами.

**Исходный набор данных:**

Пользователь настроил приложение для использования и перешел на страницу с задачами.

**Ожидаемый результат:**

Пользователю приходит пустой список с задачами. Возвращается 200 статусный код.

**Фактический результат:**

Пользователю приходит пустой список с задачами. Возвращается 200 статусный код.

* **Тест 4**

**Тестовая ситуация:**

Пользователь открывает страницу со списком задач. В лог-файл добавляется запись о том, что пользователю были загружены все записи по задачам.

**Исходный набор данных:**

Пользователь настроил приложение для использования и перешел на страницу с задачами.

**Ожидаемый результат:**

В лог-файл будет добавлена запись, что пользователю были загружены все записи по задачам.

**Фактический результат:**

В лог-файл была добавлена запись, что пользователю были загружены все записи по задачам.

### Тестирование операций над записями

* **Тест 1**

**Тестовая ситуация:**

Пользователь добавляет новый проект.

**Исходный набор данных:**

Пользователь перешел на страницу добавления проекта и ввел корректные данные.

**Ожидаемый результат:**

После нажатия на кнопку добавления, пользователя перенаправит на детальную страницу о добавленном проекте. В базе данных появится запись о добавленном проекте.

**Фактический результат:**

После нажатия на кнопку добавления, пользователя перенаправило на детальную страницу о добавленном проекте. В базе данных появилась запись о добавленном проекте.

* **Тест 2**

**Тестовая ситуация:**

Пользователь изменяет добавленную задачу.

**Исходный набор данных:**

Пользователь добавил корректную задачу, перешел на страницу изменения и подтвердил измененные данные.

**Ожидаемый результат:**

Пользователя перенаправит на детальную страницу о измененном проекте. Данные о задаче будут изменены.

**Фактический результат:**

Пользователя перенаправило на детальную страницу о измененном проекте. Данные о задаче были изменены.

* **Тест 3**

**Тестовая ситуация:**

Пользователь удаляет задачу.

**Исходный набор данных:**

Пользователь добавил корректную задачу и нажал на кнопку удаления.

**Ожидаемый результат:**

Задача была удалена из базы данных и больше не отображается пользователю.

**Фактический результат:**

Задача была удалена из базы данных и больше не отображается пользователю.

### Тестирование защиты от некорректных данных

* **Тест 1**

**Тестовая ситуация:**

Пользователь хочет добавить задачу с названием, содержащим цифры.

**Исходный набор данных:**

Пользователь ввел некорректные данные и нажал на кнопку сохранения

**Ожидаемый результат:**

Приложение не позволит добавить такую запись. Пользователю выводится сообщение об ошибке.

**Фактический результат:**

Приложение не позволило добавить такую запись. Пользователю выводится сообщение об ошибке.

* **Тест 2**

**Тестовая ситуация:**

Пользователь хочет добавить задачу с датой окончания раньше, чем дата начала.

**Исходный набор данных:**

Пользователь ввел некорректные данные и нажал на кнопку сохранения.

**Ожидаемый результат:**

Приложение не позволило добавить такую запись. Пользователю выводится сообщение об ошибке, указывающее, что дата окончания должна быть позже, чем дата начала.

**Фактический результат:**

Приложение не позволило добавить такую запись. Пользователю выводится сообщение об ошибке, указывающее, что дата окончания должна быть позже, чем дата начала.

* **Тест 3**

**Тестовая ситуация:**

Пользователь создает нового сотрудника.

**Исходный набор данных:**

Пользователь создает нового сотрудника, указывая в должности название, содержащее буквы и цифры.

**Ожидаемый результат:**

Приложение сохраняет в базе данных нового сотрудника.

**Фактический результат:**

Приложение сохраняет в базе данных нового сотрудника.

### Тестирование связей между сущностями

* **Тест 1**

**Тестовая ситуация:**

Пользователь хочет связать задачу с определенным проектом.

**Исходный набор данных:**

В базе данных уже есть записи о проекте и задаче. Пользователь изменяет на странице изменения задачи поле с проектом.

**Ожидаемый результат:**

На детальной странице выбранной задачи, указанно название связанного проекта.

**Фактический результат:**

На детальной странице выбранной задачи, указанно название связанного проекта.

* **Тест 2**

**Тестовая ситуация:**

Пользователь хочет назначить задачу нескольким исполнителям.

**Исходный набор данных:**

В базе данных уже есть записи о задаче и сотрудниках. Пользователь изменяет на странице изменения задачи поле с исполнителями.

**Ожидаемый результат:**

На детальной странице выбранной задачи и на страницах указанных исполнителей отображаются связанные данные.

**Фактический результат:**

На детальной странице выбранной задачи и на страницах указанных исполнителей отображаются связанные данные.

* **Тест 3**

**Тестовая ситуация:**

Пользователь хочет назначить сотруднику несколько задач.

**Исходный набор данных:**

В базе данных уже есть записи о сотруднике и задачах. Пользователь изменяет на странице изменения сотрудника поле с задачами.

**Ожидаемый результат:**

На детальной странице выбранного сотрудника и на страницах указанных задач отображаются связанные данные.

**Фактический результат:**

На детальной странице выбранного сотрудника и на страницах указанных задач отображаются связанные данные.

# **5 Руководство пользователя**

## **5.1 Требования для разворачивания проекта**

Необходимые зависимости для разворачивания проекта:

1. .NET Core SDK 2.1
2. MS SQL Server 2014
3. MSBuild 15.0

## **5.1 Разворачивание проекта**

Для того, чтобы развернуть проект необходимо:

1. Открыть папку «Automation»
2. Запустить «build.bat» для того, чтобы собрать приложение
3. После открытия командной строки, ввести путь к установленному «msbuild.exe»
4. Запустить «create\_database.bat» для того, чтобы сгенерировать базу данных для приложения.
5. Запустить «start.bat», чтобы развернуть сервер.

## **5.2 Дополнительные возможности**

В приложении предусмотрены дополнительные возможности для пользователя:

1. В том случае, если пользователю необходимо очистить базу данных. В папке «Automation» находится файл «drop\_database.bat», после которого необходимо выполнить «create\_database.bat» для дальнейшего использования приложения.
2. В приложении есть возможность переключать провайдера обращений к базе данных. Пользователю предоставляется возможность использования двух провайдеров: ADO.NET и Entity Framework. Для того, чтобы переключить провайдера, необходимо изменить в файле «appsettings.json» в секции «Database» поле «ADO.NET» на «true» или «false» соответственно.

# **Заключение**

По результатам курсового проекта был создан веб-сервис для отслеживания задач, проектов и пользователей в системе. Во время разработки программы были учтены все требования, предъявляемые к современным приложениям такого рода.

Приложение было написано с использованием принципов объектно-ориентированного программирования, что делает код проекта максимально понятным и легко модифицируемым.

В результате разработки проекта были изучены основы разработки веб-приложений, закреплены общие знания по языку программирования C#, были изучены основные методы тестирования программных средств и были изучены шаблоны проектирования и их использование в веб-разработке.

Данный курсовой проект является лишь первой версией разрабатываемого сервиса. В дальнейшем планируется расширение возможностей и добавление новых технологий в проект. Проекты подобного типа предоставляют удобную площадку для изучения и внедрения технологий, используемых в современной разработке.

# **Список использованных источников**

[1] Сайт Jira [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.atlassian.com/software/jira/

[2] Сайт Trello [Электронный ресурс]. – Электронные данные. – Режим доступа: https://trello.com/

[3] Сайт Redbooth [Электронный ресурс]. – Электронные данные. – Режим доступа: https://redbooth.com/

[4] Сайт Metanit [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://metanit.com/sharp/patterns/2.1.php>

[5] Сайт Studopedia [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://studopedia.ru/9_122534_mnogourovnevaya-arhitektura.html>

[6] Сайт Metanit [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://metanit.com/sharp/aspnet5/2.1.php>