СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc418898097)

[1 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc418898098)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 16](#_Toc418898099)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 21](#_Toc418898100)

[3.1 Классы, реализующие непосредственно логику работы веб-приложения 21](#_Toc418898101)

[3.2 Структура таблиц базы данных и классы моделей данных 32](#_Toc418898102)

[3.3 Классы работы с базой данных и доступа к данным 40](#_Toc418898103)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 52](#_Toc418898104)

[4.1 Авторизация 52](#_Toc418898105)

[4.2 Оформление заявления 53](#_Toc418898106)

[4.3 Работа с cookie 60](#_Toc418898107)

[5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 63](#_Toc418898108)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 67](#_Toc418898109)

[7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОГО ПРОЕКТА 73](#_Toc418898110)

[7.1 Описание проекта 73](#_Toc418898111)

[7.2 Расчёт сметы затрат и цены ПО 73](#_Toc418898112)

[7.3 Расчёт экономического эффекта от применения программного средства у заказчика 84](#_Toc418898113)

[8 ПРОФИЛАКТИКА ПЕРЕУТОМЛЕНИЯ РАБОТНИКОВ, ЗАНЯТЫХ РЕШЕНИЕМ ЗАДАЧ ПО СОСТАВЛЕНИЮ ПРОГРАММ 90](#_Toc418898114)

[8.1 Понятие работоспособности и утомления, определяющие их факторы для работников умственного труда 90](#_Toc418898115)

[8.2 Основные пути снижения утомления программистов 92](#_Toc418898116)

[ЗАКЛЮЧЕНИЕ 97](#_Toc418898117)

[СПИСОК ЛИТЕРАТУРЫ 98](#_Toc418898118)

[ПРИЛОЖЕНИЕ А. Исходный текст класса MakeRequestController 100](#_Toc418898119)

[ПРИЛОЖЕНИЕ Б. Исходный текст класса RequestManager 107](#_Toc418898120)

[ПРИЛОЖЕНИЕ В. Спецификация проекта 115](#_Toc418898121)

[ПРИЛОЖЕНИЕ Г. Ведомость документов 116](#_Toc418898122)

ВВЕДЕНИЕ

Данный дипломный проект представляет собой веб-приложение для подачи заявлений на выдачу специальных разрешений на проезд транспортных средств, максимальные весовые и (или) габаритные размеры которых превышают допустимые параметры, установленные для проезда по автомобильным дорогам общего пользования Республики Беларусь.

Данное приложение является частью системы, предназначенной для автоматизации процесса оформления специальных разрешений и предоставления дополнительных возможностей грузоперевозчикам по подаче заявлений на проезд тяжеловесных и крупногабаритных транспортных средств.

Цель разработки приложения – в первую очередь, сокращение сроков оформления специальных разрешений и снижение бумажного документооборота.

Согласно логике работы системы, автоперевозчику при подаче заявления необходимо предоставить следующую информацию: принадлежность транспортного средства, габаритные параметры, осевые нагрузки, межосевые расстояния, типы осей, массу и тип груза, срок перевозки, количество рейсов. Также необходимо задать маршрут, представленный набором точек: начальная, промежуточные и конечная. После чего заявление направляется на рассмотрение в областную группу управления безопасной эксплуатации автомобильных дорог (УБЭАД).

Помимо электронной подачи заявления дополнительно система, частью которой является данный проект, предоставляет следующие возможности:

* автоматизированный процесс оформления специального разрешения по данным электронной заявки, включая проверку соответствия нагрузок и габаритов транспортных средств установленным нормативам, в том числе в период сезонных ограничений;
* ведение электронного реестра поступивших заявлений с текущей информацией о состоянии (рассмотрено, сделан расчет, сделано техническое заключение, не востребовано, оплата расчета);
* проверка поступившей платы по выданным расчетам платы.

По своей структуре данный проект представляет собой веб-приложение, то есть клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер.

Разработка данного приложения производилась по технологии ASP.NET MVC 5, которая в свою очередь является частью платформы Microsoft .NET Framework.

1. ОБЗОР ЛИТЕРАТУРЫ

Во многих странах, в законодательстве которых введены ограничения на проезд крупногабаритных и тяжеловесных транспортных средств, используются приложения, подобные данному проекту.

Один из ярких примеров – приложение от Российского федерального дорожного агентства (Росавтодор) – «Получение специального разрешения на движение по автомобильным дорогам транспортного средства, осуществляющего перевозки тяжеловесных и крупногабаритных грузов» [1]. Данное приложение интегрировано в портал государственных услуг Российской Федерации. Серверная часть приложения реализована с использованием технологии Java Enterprise Edition (Java EE).

На портале электронного лицензирования Республики Казахстан (РК) также используется подобная система – «Получение специального разрешения на проезд тяжеловесных и крупногабаритных транспортных средств по территории РК» [2]. Здесь в качестве технологии создания веб-приложения была использована ASP.NET MVC.

В США для каждого штата существует отдельное приложение для подачи специальных разрешений, учитывающее особенности законодательства каждого штата. Например, приложение для штата Техас «TxPROS Permitting System» [3], реализованное с использованием технологии ASP.NET Web Forms.

Общим моментом всех вышеупомянутых приложений является похожая логика и предоставляемые данные при подаче заявления: наименование автоперевозчика, параметры транспортного средства, маршрут движения. Однако стоит отметить, что информация, которую необходимо предоставить, порядок подачи заявления и получения специального разрешения, допустимые параметры транспортного средства определяются действующим законодательством, и, поэтому различаются в разных государствах.

Необходимо отметить, что с учетом всевозможных достоинств и недостатков известных технологий, сред разработок, были выбраны следующие компоненты и технологии: для написания веб-приложения – технология ASP.NET MVC, являющаяся частью платформы Microsoft .NET Framework, язык программирования – C#.

Microsoft .NET Framework – программная технология, предназначенная для создания как обычных программ, так и веб-приложений. Фактически представляет собой операционную систему внутри операционной системы, высокопроизводительную, основанную на стандартах, многоязыковую среду, которая позволяет интегрировать существующие приложения с приложениями и сервисами следующего поколения, а также решать задачи развертывания и использования интернет-приложений. .NET Framework состоит из двух частей: общеязыковой исполняющей среды Common Language Runtime (CLR) и библиотеки классов Framework Class Library (FCL). Основой платформы является [виртуальная машина](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) [CLR](http://ru.wikipedia.org/wiki/Common_Language_Runtime), способная выполнять как обычные настольные программы, так и веб-приложения. Одной из основных идей Microsoft .NET является совместимость различных служб, написанных на разных языках. Например, служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi; на C# можно написать класс, наследованный от класса, написанного на Visual Basic .NET, а исключение, созданное методом, написанным на C#, может быть перехвачено и обработано в Delphi. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок. Так же, как и технология Java, среда разработки .NET создаёт байт-код, предназначенный для исполнения виртуальной машиной. Входной язык этой машины в .NET называется Microsoft Intermediate Language (MSIL), или Common Intermediate Language (CIL), или просто IL. Применение байт-кода позволяет получить кроссплатформенность на уровне скомпилированного проекта (в терминах .NET – сборка), а не только на уровне исходного текста, как, например, в С. Перед запуском сборки в среде исполнения CLR, байт-код преобразуется встроенным в среду JIT-компилятором в машинные коды целевого процессора. Также существует возможность скомпилировать сборку в родной код для выбранной платформы.

Архитектура .NET Framework описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке FCL. Туда входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL).

Base Class Library (BCL) – стандартная библиотека классов платформы .NET Framework. Программы, написанные на любом из языков, поддерживающих платформу .NET, могут пользоваться классами и методами BCL – создавать объекты классов, вызывать их методы, наследовать необходимые классы BCL и так далее.

C# – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах в компании Microsoft как один из языков разработки приложений для платформы Microsoft .NET Framework. C# относится к семье языков с C‑подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

Описание базовых принципов функционирования платформы .NET, системы типов .NET и различных инструментальных средств разработки, используемых при создании приложений, базовые возможности языка программирования С#, включая новые синтаксические конструкции, появившиеся с выходом .NET 4.5, а также синтаксис и семантика языка CIL подробно описаны в книге «CLR via C#» [4].

В процессе написания любого программного продукта приходится неоднократно сталкиваться с вопросами использования тех или иных функций. В этом случае незаменима электронная документация к .NET Framework [5], предлагаемая разработчиками компании Microsoft и входящая в пакет Visual Studio. Здесь приведены сведения об использовании классов системных библиотек и поведении конкретных методов.

Наравне с документацией [5] использовалась спецификация языка программирования C# [6], которая по праву считается первоисточником для изучения языка С# и платформы .NET Framework.

ASP.NET – технология создания веб-приложений и веб-сервисов от компании Майкрософт. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP. На момент своего появления в 2002 году ASP.NET стала огромным шагом вперед. Стек технологий Microsoft, как он выглядел на то время, показан на рисунке 1.1 [7].

В ASP.NET Web Forms разработчики Microsoft попытались сокрыть как протокол HTTP (с его неизбежным отсутствием состояния), так и язык HTML (который на тот момент был незнаком многим разработчикам), моделируя интерфейс пользователя в виде иерархии серверных объектов, представляющих элементы управления. Каждый такой элемент управления отслеживает собственное состояние между запросами (с помощью средства View State по мере необходимости визуализируя себя в виде HTML – разметки, и автоматически подключая события клиентской стороны (например, щелчки на кнопках) с соответствующим кодом их обработки на стороне сервера). Фактически Web Forms – это гигантский уровень абстракции, разработанный для воссоздания классического, управляемого событиями графического пользовательского интерфейса в веб-среде.



Рисунок 1.1 – Стек технологий ASP.NET Web Forms

Идея состояла в том, чтобы веб-разработка выглядела подобно разработке Windows Forms. Отныне разработчикам не нужно иметь дело с сериями независимых запросов и ответов HTTP; теперь можно мыслить терминами сохраняющего свое состояние интерфейса пользователя. Можно забыть о веб-среде и её не поддерживающей состояние природе, а вместо этого строить пользовательские интерфейсы с помощью визуального конструктора, использующего технологию перетаскивания, и полагать – или, по меньшей мере, делать вид – что все происходит на сервере.

ASP.NET MVC – это платформа для разработки веб-приложений от Microsoft, которая сочетает в себе эффективность и аккуратность архитектуры «модель-представление-контроллер», новейшие идеи и приемы гибкой разработки, а также все лучшее из существующей платформы ASP.NET. Она представляет собой полномасштабную альтернативу традиционной технологии ASP.NET Web Forms, предоставляя существенные преимущества для всех проектов веб-разработки, кроме наиболее тривиальных. Новая платформа ASP.NET MVC обеспечила радикальный сдвиг в разработке веб-приложений на платформе Microsoft. В ней делается упор на ясную архитектуру, шаблоны проектирования и тестируемость, и не предпринимается попыток сокрытия того, как работает веб-среда.

Термин модель-представление-контроллер (model-view-controller) используется с конца 70-х годов прошлого столетия. Эта модель явилась результатом проекта Smalltalk в компании Xerox, где она была задумана как способ организации некоторых из ранних приложений графического пользовательского интерфейса. Некоторые из нюансов первоначальной модели MVC были связаны с концепциями, специфичными для Smalltalk, такими как экраны и инструменты, но более глобальные понятия все еще применимы к приложениям, и особенно хорошо они подходят для веб-приложений [7].

Концепция паттерна (шаблона) MVC (model-view-controller) предполагает разделение приложения на три компонента [8]:

1. Контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.
2. Представление (view) – это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт.
3. Модель (model) представляет класс, описывающий логику используемых данных.

Схема шаблона разработки MVC изображена на рисунке 1.2.

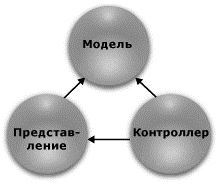


Рисунок 1.2 – Шаблон разработки MVC

В ASP.NET MVC контроллеры являются классами, обычно производными от класса System.Web.Mvc.Controller. Каждый метод public в классе, унаследованном от класса Controller, называется методом действия и посредством системы маршрутизации ASP.NET связан с конфигурируемым URL. Когда запрос отправляется по URL, связанному с методом действия, в классе контроллера выполняются операторы, чтобы провести некоторую операцию по отношению к модели и затем выбрать представление для отображения клиенту. Взаимодействия между контроллером, моделью и представлением показаны на рисунке 1.3 [8].

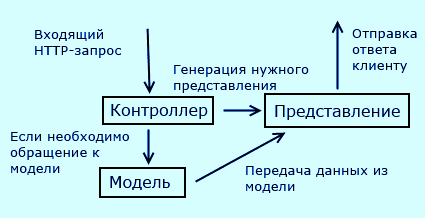


Рисунок 1.3 – Взаимодействие между контроллером, моделью и представлением

В отличие от ASP.NET Web Forms платформа ASP.NET MVC имеет следующие преимущества [9]:

* она облегчает управление сложными структурами путем разделения приложения на модель, представление и контроллер;
* она не использует состояние просмотра и серверные формы, что делает платформу MVC идеальной для разработчиков, которым необходим полный контроль над поведением приложения;
* она использует схему основного контроллера, при которой запросы веб-приложения обрабатываются через один контроллер, что позволяет создавать приложения, поддерживающие расширенную инфраструктуру маршрутизации;
* она обеспечивает расширенную поддержку разработки на основе тестирования;
* она хорошо подходит для веб-приложений, поддерживаемых крупными коллективами разработчиков, а также веб-разработчикам, которым необходим высокий уровень контроля над поведением приложения.

Платформа ASP.NET MVC предоставляет следующие возможности [9]:

1. Разделение задач приложения (логика ввода, бизнес-логика и логика пользовательского интерфейса), широкое возможности тестирования и разработки на основе тестирования. Все основные контракты платформы MVC основаны на интерфейсе и подлежат тестированию с помощью макетов объекта, которые имитируют поведение реальных объектов приложения. Приложение можно подвергать модульному тестированию без запуска контроллеров в процессе ASP.NET, что ускоряет тестирование и делает его более гибким. Для тестирования возможно использование любой платформы модульного тестирования, совместимой с .NET Framework.
2. Расширяемая и дополняемая платформа. Компоненты платформы ASP.NET MVC можно легко заменить или настроить. Разработчик может подключать собственный механизм представлений, политику маршрутизации URL-адресов, сериализацию параметров методов действий и другие компоненты.
3. Расширенная поддержка маршрутизации ASP.NET. Этот мощный компонент сопоставления URL-адресов позволяет создавать приложения с понятными URL-адресами, которые можно использовать в поиске. URL-адреса не должны содержать расширения имен файлов и предназначены для поддержки шаблонов именования URL-адресов, обеспечивающих адресацию, оптимизированную для поисковых систем (SEO) и для передачи репрезентативного состояния (REST).
4. Поддержка использования разметки в существующих файлах страниц ASP.NET (ASPX), элементов управления (ASCX) и главных страниц (MASTER) как шаблонов представлений. Вместе с платформой ASP.NET MVC можно использовать существующие функции ASP.NET, например, вложенные главные страницы, декларативные серверные элементы управления, шаблоны, привязку данных, локализацию и другие.
5. Поддержка существующих функций ASP.NET. ASP.NET MVC позволяет использовать такие функции, как проверка подлинности с помощью форм и Windows, проверка подлинности по URL-адресу, членство и роли, кэширование вывода и данных, управление состоянием сеанса и профиля, наблюдение за работоспособностью, система конфигурации и архитектура поставщика.

Платформа ASP.NET MVC предоставляет поддержку для выбора механизмов визуализации. В более ранних версиях MVC использовался стандартный механизм визуализации ASP.NET, который обрабатывал ASPX‑страницы с применением оптимизированной версии синтаксиса разметки Web Forms. В версии платформы MVC 3 был введен механизм визуализации Razor, который использует совершенно другой синтаксис. Visual Studio обеспечивает поддержку средства IntelliSense для обоих механизмов визуализации, максимально упрощая внедрение и ответ на данные представления, предоставленные контроллером.

При обработке запросов фреймворк ASP.NET MVC опирается на систему маршрутизации, которая сопоставляет все входящие запросы с определенными в системе маршрутами, которые указывают какой контроллер и метод должен обработать данный запрос. Встроенный маршрут по умолчанию предполагает трехзвенную структуру: контроллер/действие /параметр [9].

Одной из наиболее популярных книг по ASP.NET MVC является книга «ASP.NET MVC 5 с примерами на C# 5.0» [7]. Она даёт подробное описание архитектуры ASP.NET MVC 5, а также инфраструктуры ASP.NET MVC Framework.

В проекте активно используется взаимодействие с базой данных Oracle. Однако использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, заставляя писать программное обеспечение, которое должно уметь как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранить эти данные в реляционной форме. Эта постоянная необходимость в преобразовании между двумя разными формами данных не только сильно снижает производительность, но и создает трудности при программировании, так как обе формы данных накладывают ограничения друг на друга.

Одним из решений проблемы семантического разрыва при использовании реляционной базы данных является Object-Relational Mapping (ORM) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». В качестве ORM-решения в проекте используется NHibernate. Это бесплатная библиотека с открытым исходным кодом, являющаяся портом на .NET популярной на платформе Java библиотеки Hibernate. NHibernate позволяет отображать объекты бизнес-логики на реляционную базу данных. По заданному XML-описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов [10]. Помимо официальной документации по Nhibernate [11], хорошим источником информации является руководство для начинающих по Nhibernate 3 [12].

По сравнению с остальными ORM-решениями библиотека Nhibernate обладает следующими преимуществами:

* разнообразные способы выборки: HQL, ICriteria, QueryOver, LINQ, SQL (включая хранимые процедуры) и трансформации результатов с использованием IResultTransformer;
* кэш второго уровня;
* большое количество точек расширения, открывающих широкие возможности по модификации поведения;
* разнообразные способы маппинга объектов на базу данных (xml, атрибуты, маппинг кодом);
* встроенная поддержка для протоколирования генерируемых sql-команд;
* более 10 разнообразных id-генераторов, включая наиболее эффективные для ORM HiLo и guid.comb;
* поддержка readonly свойств;
* поддержка коллекций элементов (например, IList<int>);
* сильное сообщество программистов.

1. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Типичное веб-приложение, написанное с использованием шаблона MVC, имеет модульную структуру. В данном проекте можно выделить пять основных модулей или блоков, определяющих функциональность приложения для пользователя: блок оформления заявлений, блок работы со списком заявлений, блок работы с организациями, блок работы с неоформленными разрешениями, блок работы с архивом разрешений. Данным блокам в ASP.NET MVC соответствуют контроллеры, которые являются центральным компонентом в архитектуре MVC. Контроллер получает ввод пользователя, обрабатывает его и посылает обратно результат обработки, например, в виде представлений. Также к каждому данному блоку относятся наборы моделей и соответствующих представлений. Отдельно выделен блок хранения промежуточных данных, используемый блоком оформления заявления для временного хранения введённой информации при подаче заявления. Блок авторизации используется для обеспечения авторизации пользователей и ограничения доступа к функциям приложения неавторизованных пользователей. Приложение активно использует доступ к базе данных, соответственно выделен блок работы с базой данных, который используется практически всеми остальными функциональными блоками. Таким образом, обобщённая структура приложения представлена на чертеже ГУИР.400201.050 C1.

На каждый логически выделенный блок программы возлагаются определенные задачи. Кроме того, каждый блок программы так или иначе связан с некоторыми остальными блоками, чтобы обеспечить работоспособность всего приложения в целом. Связь, как правило, реализуется посредством обмена данными между блоками.

*Блок оформления заявлений* по сути является основной целью данного приложения. Он представляет собой класс-контроллер, а также набор классов-моделей и соответствующих им представлений пользовательского интерфейса. Данный блок определяет всю логику подачи заявления. Процесс подачи заявления состоит из нескольких шагов, причём каждый шаг представлен отдельным методом контроллера:

1. Указание типа специального разрешения, срока перевозки, количества рейсов, а также имени и телефона ответственного за оформление.
2. Указание принадлежности транспортного средства, организации-перевозчика, а также состава автопоезда. Перевозчик выбирается из базы данных по учётному номеру плательщика (УНП) или по именованию либо информация о нём вводится вручную. Транспортные средства для автопоезда выбираются из базы данных для данного перевозчика либо вводятся вручную.
3. Ввод габаритных параметров автопоезда, заполнение схемы автопоезда в виде осевых нагрузок, межосевых расстояний, типов осей.
4. Установка маршрута движения. Маршрут состоит из набора точек: начальная, промежуточные, конечная. Каждая точка маршрута может быть представлена в виде: населённого пункта, дороги, пункта пропуска.
5. Предварительный просмотр заявление перед отправкой, а также ввод дополнительной информации в поле Примечание.

Так как структура сущностей в базе данных отличается от структуры данных на различных этапах подачи заявления, то для каждого представления используются так называемые модели представления (ViewModel), структура которых соответствует представлению.

*Блок хранения промежуточных данных* представлен файлами HTTP cookie в браузере пользователя, используемыми для хранения данных между этапами оформления заявления до его окончательной отправки. Таким образом, пользователь может продолжить оформление заявления в любой момент, пока в браузере не будут очищены файлы cookie.

Интерфейс *блока работы со списком заявлений* является первым, что видит пользователь приложения. Данный блок представлен классом-контроллером и набором моделей и представлений, которые предоставляют пользователю возможность просмотра и выполнения различных операций над поданными заявлениями. В табличной форме отображаются информация о поданных заявлениях, по которым ещё не принято решение о выдаче или не выдаче специального разрешения. Предоставляются такие операции, как удаление заявления (если ещё не выполнен расчёт платы), просмотр заявления в формате pdf, просмотр расчёта платы в формате pdf, просмотр технических заключений. Предоставляется возможность ввода информации о произведённой оплате после того как выставлен счёт на оплату специального разрешения.

*Блок работы с архивом разрешений* предоставляет операции для работы с заявлениями, по которым получены специальные разрешения. Информация о данных заявлениях и соответствующих им специальных разрешениях представлена в табличной форме. Возможен выбор данных за определённый период, а также поиск заявления в базе данных по номеру расчёта, номеру специального разрешения или государственному номеру транспортного средства. Предоставляются операции просмотра заявления или расчёта платы по данному заявлению в формате pdf. Также существует функция использования существующего заявления как шаблона для создания нового заявления. При этом все данные выбранного заявления копируются из базы данных в соответствующие cookie браузера пользователя и происходит переход к блоку оформления заявлений, поля которого уже заполнены данными выбранного заявления.

*Блок работы с неоформленными разрешениями* содержит операции с заявлениями, по которым по тем или иным причинам не выданы специальные разрешения. Причина отказа в оформлении разрешения указывается в специальном столбце таблицы. Так же, как и в блоке работы с архивом разрешений возможен выбор данных за определённый период, поиск заявления в базе данных по номеру расчёта или государственному номеру транспортного средства, использование заявления как шаблона, предоставляется возможность просмотра заявления или расчёта платы в формате pdf.

*Блок работы с организациями* предоставляет возможность выполнения некоторых операций с организациями-перевозчиками, которые фигурировали в поданных заявлениях клиента. В частности, возможны просмотр данных, редактирование некоторой информации об организации.

*Блок авторизации* используется для ограничения доступа к функциям приложения неавторизованных пользователей. Данный блок используется всеми вышеперечисленными блоками. Блок представлен фильтром авторизации AuthorizeBdcAppAttribute, а также классом Authorize. Фильтр в ASP.NET MVC представляет собой некоторый класс, реализующий логику, которая должна отрабатывать до вызова действий контроллера. Фильтры реализованы как атрибуты, которые могут применять ко всему классу-контроллеру, так и к отдельным его методам, свойствам и полям. Фильтры авторизации срабатывают после фильтров аутентификации и до запуска остальных фильтров и вызова методов действий. Цель фильтров авторизации – разграничить доступ пользователей, чтобы к определенным ресурсам приложения имели доступ только определенные пользователи. Фильтры авторизации реализуют интерфейс IAuthorizationFilter. И если при получении запроса окажется, что к запрашиваемому действию контроллера применяется данный фильтр, то сначала срабатывает метод OnAuthorization данного интерфейса. И если фильтр одобрит запрос, то далее вызывается действие. Иначе действие не будет работать. В случае данного приложения фильтр AuthorizeBdcAppAttribute наследует встроенный атрибут авторизации AuthorizeAttribute, который реализует вышеописанный интерфейс. В фильтре AuthorizeBdcAppAttribute происходит обращение к классу Authorize. Данный класс реализует проверку, аутентифицирован и авторизован ли пользователь, с использованием аутентификации форм (Forms Authentication).

Аутентификация с помощью форм – это система аутентификации ASP.NET общего назначения. Посредством аутентификации с помощью форм платформа ASP.NET создает cookie-набор безопасности для зарегистрированных пользователей, обслуживает их и автоматически поддерживает контекст безопасности для последующих запросов. Аутентификация с помощью форм основана на билетах (также называемых маркерами). Это значит, что, когда пользователь регистрируется, он получает так называемый билет с базовой информацией о себе. Информация сохраняется в зашифрованном cookie-наборе, который присоединяется к ответу, так что автоматически отправляется в каждом последующем запросе.

Стоит отметить, что регистрация и процесс ввода логина и пароля в данном приложении не реализованы, так как данный проект – часть единой системы проектов информационного центра дорожного хозяйства с общей системой авторизации, и процесс регистрации происходит в другом приложении. Класс Authorize в данном приложении только получает имя зарегистрированного пользователя (с установленным cookie-набором) с помощью классов аутентификации форм и проверяет наличие пользователя в базе данных и наличие необходимых привилегий.

*Блок работы с базой данных* используется практически всеми остальными вышеперечисленными блоками. Данный блок позволяет обращаться к таблицам в базе данных и выполнять с ними различные операции: чтения и поиск, вставка и удаление объектов. Работа с базой данных производится с использованием технологии Object-Relational Mapping (ORM), которая связывает таблицы базы данных с классами бизнес-объектов приложения. В качестве ORM-решения используется бесплатная библиотека Nhibernate для платформы Microsoft .NET. По заданному XML-описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов. Однако, в данный проект не включены файлы c XML-описанием сущностей, так как используется дополнительная библиотека Nhibernate Attributes. Она автоматически генерирует XML-описание при наличии атрибутов у полей класса бизнес-объекта с описанием связи данного поля с объектом в базе данных.

В данном приложении работа с базой данных осуществляется с помощью классов-менеджеров, каждый из которых отвечает за работу с группой близких сущностей. Классы-менеджеры наследуются от одного и того же класса AbstractManager. Данный класс производит настройку Nhibernate, соединение с базой данных, а также предоставляет базовые операции: сохранения, получения и удаления объекта по идентификатору и другие. Вызовы всех функций менеджеров производятся через объект класса TktsRequestService, который можно считать реализацией шаблона фасад – структурного шаблона проектирования, позволяющего скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы. К тому же, класс TktsRequestService наследуется от класса TemplateService, который в свою очередь позволяет получать текущего пользователя из контекста текущего HTTP-запроса. Таким образом, достигается разделение ответственности.

Как было упомянуто выше, данный проект – часть единой системы проектов информационного центра дорожного хозяйства. В связи с этим необходимо отметить, что классы, наследуемые от AbstractManager, а также классы сервисов (такие как TktsRequestService) вынесены в разные проекты. Каждому приложению системы в данных проектах соответствует свой каталог, где находятся классы-менеджеры или классы сервисов.

Также к блоку работы с базой данных можно отнести класс DataProvider, который выполняет преобразование объектов доменных классов-моделей в объекты классов моделей представления. Доменные модели представляют собой бизнес-объекты приложения, которые соответствуют таблицам в базе данных, тогда как структура моделей представлений соответствует структуре представлений пользовательского интерфейса. Модель представления отличается от доменной модели, так как на одной странице пользовательского интерфейса могут быть как лишь небольшая часть информации из доменной модели, так и некоторая дополнительная информация.

1. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе подробно рассматривается функционирование программных модулей. Перечислены соответствующие классы и большинство их компонентов, описана структура и связи таблиц в базе данных. Состав и отношения основных классов показаны также на диаграмме классов (чертёж ГУИР.400201.050 РР.1).

* 1. Классы, реализующие непосредственно логику работы веб-приложения

К данным классам относятся прежде всего классы контроллеров. Контроллеры обрабатывают входящие запросы, вводимые пользователями данные и их действия, а также реализуют необходимую логику веб-приложения. Класс контроллера вызывает отдельный компонент представления, который создает в качестве ответа HTML-разметку. Базовым классов всех контроллеров в данном проекте является класс Controller, который является реализацией контроллера по умолчанию и отвечает за следующие этапы обработки:

* поиск метода действия, который нужно вызвать, и проверка допустимости его вызова;
* получение значений, используемых как аргументы метода действия;
* обработка всех ошибок, которые могут возникнуть при выполнении метода действия.

Все методы контроллера с модификатором public являются так называемыми методами действий. Методы действий (action methods) представляют такие методы контроллера, которые обрабатывают запросы по определенному URL. Возвращаемым результатом метода действия является объект класса, производного от ActionResult. ActionResult представляет собой абстрактный класс, в котором определен один метод ExecuteResult, переопределяемый в классах наследниках. В данном методе, как правило, генерируется ответ для клиента и производится его запись в выходной поток HTTP-ответа. В данном проекте в качестве результатов действий используются в основном встроенные классы фреймворка ASP.NET MVC:

* ViewResult – производит рендеринг представления и отправляет результаты рендеринга в виде html-страницы клиенту;
* JsonResult – возвращает в качестве ответа объект или набор объектов в формате JSON;
* PartialViewResult – производит рендеринг частичного представления в выходной поток;
* RedirectToRouteResult – перенаправляет пользователя по определенному адресу URL, указанному через параметры маршрута;
* HttpStatusCodeResult – результат действия, который возвращает клиенту определенный статусный код HTTP;
* HttpNotFoundResult – производный от HttpStatusCodeResult, возвращает клиенту ответ в виде статусного кода HTTP 404, указывая, что запрошенный ресурс не найден.

Также в данном подразделе будут описаны классы работы с авторизацией, классы работы с HTTP cookie, которые используются классами контроллеров.

* + 1. Класс MakeRequestController

Данный класс контроллера реализует процесс подачи заявления, который состоит из нескольких шагов. Каждый шаг представлен определённым методом действия контроллера.

Константы:

* IsReusableCookieName – константа типа string, задающая имя HTTP cookie, указывающего на тип специального разрешения (обычное или многоразовое).

Поля:

* \_dataProvider – поле типа DataProvider, содержащее ссылку на объект, предоставляющий операции доступа к данным;
* \_cookieManager – поле типа RequestCookieManager, содержащее ссылку на объект, предоставляющий операции с файлами cookie в процессе подачи заявления.

Методы:

* ActionResult RequestInfo() – метод действия контроллера, отправляющий клиенту страницу с полями для заполнения предварительной информации о заявке (тип специального разрешения, дата, имя и другое);
* ActionResult RequestInfo (RequestInfoModel requestInfo) – метод действия, который получает от клиента заполненную модель с предварительной информацией о заявке и сохраняет её в файл cookie, а также перенаправляет клиента на следующий шаг;
* ActionResult RequestInfoPage (RequestInfoModel requestInfo) – вспомогательный метод, который перед отправкой клиенту модели с предварительной информацией о заявке загружает в объект ViewBag некоторую дополнительную информацию (в частности, список областных групп управления безопасной эксплуатации автомобильных дорог);
* ActionResult VehicleDetails() – метод действия, который формирует для клиента представление с формой и полями, позволяющими выбрать перевозчика, а также сформировать состав сцепки транспортных средств (автопоезда), для которой необходимо специальное разрешение;
* ActionResult VehicleDetails (VehicleDetailsModel vehicleDetails, string next, string back) – метод действия, который получает от клиента модель с данными о перевозчике и составом автопоезда и сохраняет её в файл cookie, а также перенаправляет клиента на следующий или предыдущий шаг в зависимости от параметров next и back;
* ActionResult VehicleDetailsPage (VehicleDetailsModel vehicleDetails) – вспомогательный метод, который перед отправкой клиенту модели VehicleDetailsModel заполняет некоторые её поля дополнительной информацией (в частности, идентификатором и наименованием организации-клиента);
* ActionResult GetVehicles (bool isOwn, int carrierOrgId) – метод действия, который формирует для клиента список транспортных средств, принадлежащих организации-перевозчику с заданным идентификатором;
* ActionResult VehicleScheme() – метод действия, формирующий для клиента страницу для заполнения схемы автопоезда и его габаритных параметров;
* ActionResult VehicleScheme (VehicleSchemeModel vehicleSchemeModel, string next, string back) – метод действия, получающий от клиента модель схемы автопоезда и габаритных параметров и сохраняющий её в файл cookie, а также перенаправляющий пользователя на следующий или предыдущий шаг;
* ActionResult Route() – метод действия, формирующий для клиента страницу для заполнения маршрута движения транспортных средств;
* ActionResult Route(RouteModel routeModel, string next, string back) – метод действия, который получает от клиента модель с маршрутом движения транспортных средств, сохраняет её в HTTP cookie, а также перенаправляет пользователя на следующий или предыдущий шаг;
* ActionResult RouteForReusable (RouteForReusableModel routeForReusableModel, string next, string back) – метод действия, который получает от клиента модель с маршрутом движения транспортных средств при подаче заявления на многоразовое специальное разрешение, сохраняет её в файл cookie, а также перенаправляет пользователя на следующий или предыдущий шаг;
* ActionResult RequestPreview() – метод действия, формирующий для клиента страницу с предварительным просмотром данных заявления перед отправкой;
* ActionResult RequestPreview(string note, string next, string back) – метод действия, который принимает от клиента примечание к заявлению, а также либо отправляет заявление на рассмотрение, либо перенаправляет клиента на предыдущий шаг;
* ActionResult CopyRequest(int id) – метод действия, копирующий существующее заявление из базы данных в файлы cookie и перенаправляющий клиента на первую страницу подачи заявления.
  + 1. Класс RequestsListController

Данный класс контроллера отвечает за работу со списком заявлений, находящихся на рассмотрении.

Поля:

* \_dataProvider – поле типа DataProvider, содержащее ссылку на объект, предоставляющий операции доступа к данным.

Методы:

* ActionResult Index() – метод действия контроллера, формирующий для клиента страницу со списком поступивших заявлений, находящихся на рассмотрении;
* ActionResult DeleteRequest(int id) – метод действия, который удаляет заявление с указанным в аргументе метода идентификатором, причём возможно удаление только заявлений, для которых ещё не выполнен расчёт платы;
* ActionResult EnterPayment(int requestId) – метод действия, формирующий для клиента форму для ввода оплаты для выбранного заявления, для которого произведён расчёт платы;
* ActionResult SavePayment(PaymentModel payment) – метод действия, принимающий модель с информацией об оплате специального разрешения;
* ActionResult DeletePayment(int paymentId) – метод действия, позволяющий удалить запись о введённой оплате.
  + 1. Класс PermitsArchiveController

Данный класс отвечает за работу со списком заявлений, по которым были выданы специальные разрешения.

Поля:

* \_dataProvider – поле типа DataProvider, содержащее ссылку на объект, предоставляющий операции доступа к данным.

Методы:

* ActionResult Index() – метод действия контроллера, формирующий для клиента страницу со списком заявлений, по которым выданы специальные разрешения;
* ActionResult FindPermitsByDateRange (DateTime startDate, DateTime endDate) – метод действия, формирующий список заявлений, по которым выданы специальные разрешения в диапазоне дат, который задан параметрами startDate и endDate;
* ActionResult FindPermitsByFilter (string vehicleNumber, string requestNumber, string permitNumber) – метод действия, формирующий список заявлений, по которым выданы специальные разрешения и у которых характеристики совпадают с параметрами: vehicleNumber – номер транспортного средства, requestNumber – номер заявления, permitNumber – номер специального разрешения.
  + 1. Класс UnissuedPermitsController

Класс отвечает за работу со списком заявлений, по которым отказано в выдаче специального разрешения по тем или иным причинам.

Поля:

* \_dataProvider – поле типа DataProvider, содержащее ссылку на объект, предоставляющий операции доступа к данным.

Методы:

* ActionResult Index() – метод действия контроллера, формирующий для клиента страницу со списком заявлений, по которым отказано в выдаче специального разрешения;
* ActionResult FindRequestsByDateRange (DateTime startDate, DateTime endDate) – метод действия, формирующий список заявлений, по которым отказано в выдаче специального разрешения в диапазоне дат, который задан параметрами startDate и endDate;
* ActionResult FindRequestsByFilter (string vehicleNumber, string requestNumber) – метод действия, формирующий список заявлений, по которым отказано в выдаче специального разрешения и у которых характеристики совпадают с параметрами: vehicleNumber – номер транспортного средства, requestNumber – номер заявления.
  + 1. Класс OrganizationsController

Класс предоставляет возможность работы со списком организаций-перевозчиков, которые когда-либо фигурировали в поданных заявлениях организации-клиента. Также предоставляется возможность добавить новую организацию-перевозчика.

Поля:

* \_dataProvider – поле типа DataProvider, содержащее ссылку на объект, предоставляющий операции доступа к данным.

Методы:

* ActionResult Index() – метод действия контроллера, формирующий для клиента страницу со списком организаций-перевозчиков, которые были указаны в качестве перевозчика в поданных заявлениях;
* ActionResult UpdateOrganization(OrgModel org) – метод действия, принимающий от клиента модель с изменёнными данными организации для сохранения изменений в базе данных (для изменения разрешены только данные организации-клиента);
* ActionResult NewCarrierForm() – метод действия, формирующий для клиента форму с полями для внесения информации об организации-перевозчике;
* ActionResult AddNewCarrier (OrganizationModel carrier) – метод действия, который получает от клиента модель с информацией об организации-перевозчике, добавляет в базу данных и возвращает клиенту идентификатор добавленного перевозчика для дальнейшего использования;
  + 1. Класс HomeController

Класс предоставляет некоторые общие методы, используемые в представлениях других контроллеров, в частности перенаправление при первом входе на сайт, получение документов для заявления в формате pdf, смена языка, выход из приложения.

Методы:

* ActionResult Index() – метод действия контроллера, который запускается при первом заходе на сайт, выполняет некоторые операции (в частности, проверка неподдерживаемых версий браузеров и регистрация посещения) и производит перенаправление клиента на первую станицу приложения (список поданных заявлений);
* ActionResult ChangeCulture(string lang) – метод действия, устанавливающий файл cookie, который указывает выбранный пользователем язык интерфейса;
* ActionResult GetReport(int requestId, ReportType type) – метод действия, который отправляет клиенту файл отчёта в формате pdf для заявления с указанным идентификатором и типом отчёта;
* ActionResult Exit() – метод действия, осуществляющий выход из приложения.
  + 1. Класс AutocompleteController

Данный класс контроллера используется для группировки методов, предоставляющих данные в текстовом формате JSON для автодополнений (autocomplete) – элементов интерфейса, предусматривающих интерактивный ввод текста по дополнению текста по введённой его части. Автодополнения реализованы на клиенте с помощью JavaScript-библиотеки Jquery.

Каждый метод действия данного контроллера возвращает результат действия типа JsonNetResult, который в свою очередь для преобразования объекта в формат JSON использует библиотеку Newtonsoft.Json.

Поля:

* \_dataProvider – поле типа DataProvider, содержащее ссылку на объект, предоставляющий операции доступа к данным.

Методы:

* ActionResult GetCarriers(string term) – метод действия контроллера, формирующий для клиента объект формата JSON со списком автоперевозчиков, наименование которых включает в себя строку term;
* ActionResult GetTowns(string term) – метод действия, формирующий для клиента объект формата JSON со списком населённых пунктов Беларуси, наименование которых включает в себя строку term;
* public ActionResult GetRoutePoints(string term, RoutePointType searchType) – метод действия, формирующий для клиента объект формата JSON со списком маршрутных точек указанного типа (населённый пункт, дорога, пункт пропуска), наименование которых включает в себя строку term.
  + 1. Класс RequestCookieManager

Данный класс отвечает за работу с файлами c cookie, которые используются во время процесса оформления заявления в качестве временного хранилища данных перед отправкой заявления. Используется вышеописанным классом контроллера MakeRequestController, входя в его состав по ссылке.

Поля:

* const IsReusableCookieName – константа типа string, задающая имя файла cookie, который используется в качестве флага типа заявления (обычное специальное разрешение или многоразовое);
* \_httpContext – поле типа HttpContext, предоставляющее информацию о HTTP-запросе, для которого идёт работа с cookie.

Методы:

* RequestCookieManager(HttpContext httpContext) – конструктор класса, инициализирующий поле \_httpContext;
* T GetObject<T>(RequestStep requestStep) – метод, получающий объект определённого типа из файла cookie, который соответствует заданному в аргументе шагу подачи заявления;
* void SaveObject(RequestStep requestStep, object objectForSave) – метод, сохраняющий объект в файл cookie, который соответствует заданному в параметре метода шагу подачи заявления;
* bool CheckPreviousCookies (RequestStep curStep) – метод проверяет наличие файлов cookie, соответствующих шагам оформления заявления, которые предшествуют заданному параметром метода шагу;
* void SetReusableCookie(bool isReusable) – метод, устанавливающий файл cookie, который используется в качестве флага типа специального разрешения (обычное или многоразовое);
* bool? CheckReusableCookie() – метод, получающий значения файла cookie, использующегося в качестве флага типа специального разрешения;
* HttpCookie CreateCookie(string name) – метод, создающий cookie с именем, заданным параметром name;
* void RemoveCookie(string name) – метод, который удаляет cookie, имя которого задано параметром name;
* void RemoveRequestCookies() – метод, удаляющий все файлы cookie, устанавливаемых в процессе подачи заявления;
* RequestModel GetRequestModel() – метод, возвращающий модель заявления, которая состоит из моделей, соответствующих каждому шагу заявления. Данные для модели берутся из файлов cookie.
  + 1. Класс CookieExtensions

Класс предоставляет методы расширения для работы с объектом файла cookie.

Методы:

* void SetAndEncryptValue (this HttpCookie cookie, object objectValue) – метод, устанавливающий для заданного файла cookie значение, которое представлено объектом objectValue. При этом объект преобразуется в формат JSON, сжимается по алгоритму GZIP, затем преобразуется в кодировку base64;
* T TryGetDecryptedValue<T>(this HttpCookie cookie) – метод, возвращающий объект типа T, после обратного предыдущему методу преобразования значения файла cookie.
  + 1. Класс AuthorizeBdcAppAttribute

Данный класс является фильтром авторизации. Используется в качестве атрибута, устанавливаемого над контроллерами или отдельными методами действий контроллера, для которых требуется пройти авторизацию. Класс наследуется от встроенного в ASP.NET MVC атрибута AuthorizeAttribute который в свою очередь реализует интерфейс IAuthorizationFilter.

Поля:

* \_actions – поле типа string[], задающее список привилегий, наличие хотя бы одной из которых необходимо пользователю для вызова соответствующего метода контроллера.

Методы:

* AuthorizeBdcAppAttribute(params string[] actions) – конструктор класса, инициализирующий поле \_actions;
* bool AuthorizeCore(HttpContextBase httpContext) – переопределяемый метод базового класса AuthorizeAttribute, возвращает булево значение в зависимости от того, прошёл ли пользователь авторизацию;
* void HandleUnauthorizedRequest (AuthorizationContext filterContext) – переопределяемый метод, который обрабатывает HTTP-запрос, не прошедший авторизацию.
  + 1. Класс Authorize

Данный класс используется для получения информации о текущем пользователе по установленному cookie-набору системы аутентификации с помощью форм ASP.NET.

Поля:

* SessionCurrentUser – константа типа string, содержащая имя объекта в сессии, который представляет текущего авторизированного пользователя;
* SessionCurrentUser – константа типа string, содержащая имя объекта в сессии, который представляет список привилегий текущего авторизированного пользователя.

Методы:

* AuthUser User() – метод, возвращающий объект текущего авторизированного пользователя. Объект берётся из текущей сессии. Если он там отсутствует, то происходит загрузка из базы данных по логину (взятому из cookie-набора), затем сохранение в сессию;
* AuthUser GetCurrentUser() – метод, возвращающий объект, описывающий пользователя из базы данных по имени, полученному из cookie-набора аутентификации форм;
* void ClearSession() – метод, очищающий текущую сессию;
* List<string> userActions() – метод, получающий привилегии пользователя в приложении;
* bool HaveActions<T> (EnterElements enterType, params T[] actions) – метод, проверяющий наличие перечня действий в списке привилегий пользователя.
  + 1. Класс CultureAttribute

Данный класс является фильтром действий, который устанавливается в качестве атрибута над действиями контроллеров. Класс наследуется от встроенного абстрактного класса ActionFilterAttribute и переопределяет метод OnActionExecuting, который вызывается перед вызовом метода действия.

Данный класс предназначен для установки текущей культуры приложения, которая в дальнейшем используется для выбора языка интерфейса и влияет на формат отображения числовых данных. Для выбора устанавливаемой культуры данным классом используется класс CultureManager, описанный ниже.

* + 1. Класс CultureManager

Данный класс предоставляет набор методов для работы с текущей культурой приложения.

Поля:

* Cultures – поле типа List<string>, задающее список культур, реализуемых в приложении.

Методы:

* string GetImplementedCulture(string cultureName) – метод, возвращающий заданную в параметре культуру, если она содержится в списке Cultures, иначе возвращает культуру по умолчанию (первая в списке);
* bool IsImplementedCulture(string cultureName) – метод, проверяющий, реализована ли указанная культура в приложении;
* string GetDefaultCulture() – метод, возвращающий культуру по умолчанию (первая в списке);
* string GetCultureFromHttpRequest(HttpContextBase httpContext) – метод, возвращающий культуру на основе HTTP-запроса клиента с учётом установленного файла cookie, отвечающего за культуру, а также настроек браузера;
* string GetSuitableUserCulture (HttpContextBase httpContext) – метод, возвращающий наиболее подходящую культуру с учётом реализованных в приложении культур, а также настроек языков в браузере пользователя.
  1. Структура таблиц базы данных и классы моделей данных

В проекте используется обращение к реляционной базе данных, работающей под управлением системы управления базами данных (СУБД) Oracle. Каждой таблице в базе данных соответствует класс доменной модели в приложении, который инкапсулирует всю необходимую для программы информацию об объекте предметной области. Как упоминалось выше, для отображения объектов классов доменных моделей на таблицы базы данных используется ORM-решение Nhibernate, а также дополнение NHibernate Attributes. Над каждым классом доменной модели устанавливается атрибут Nhibernate с наименованием соответствующей таблицы в базе данных. Также над каждым свойством класса устанавливается атрибут с наименованием соответствующего поля в таблице. Существуют специальные атрибуты для отображения отношений таблиц «один-ко-многим», «многие-к-одному» – стороне «многие» соответствует коллекция объектов, входящая в состав класса.

Структура таблиц и связи между ними показаны на чертеже модели данных (ГУИР.400201.050 РР.2), а также подробно рассмотрены далее.

* + 1. Таблица REQUEST

Таблица содержит сведения о заявлениях на выдачу специального разрешения.

Поле первичного ключа: IDREQUEST – содержит уникальный идентификатор заявления.

Поля внешних ключей:

* IDORG\_REQUEST – поле, представляющее собой идентификатор организации, которая подаёт заявление на выдачу специального разрешения. Поле ссылается на первичный ключ IDORG таблицы ORG;
* IDORG\_CAR – поле, представляющее собой идентификатор организации, которая выступает в качестве перевозчика. Поле ссылается на первичный ключ IDORG таблицы ORG;
* GROUP\_UTK – поле, представляющее собой идентификатор областной группы УБЭАД, в которую подаётся заявление для рассмотрения. Ссылается на первичный ключ IDDEPARTMENT таблицы DEPARTMENT.

Поля:

* DT\_REQUEST – дата подачи заявления;
* GRUZ – описание груза, перевозимого на транспортном средстве, для которого необходимо получение специального разрешения;
* MASSA\_GRUZ – масса груза в килограммах;
* LENGTH\_TS – длина в метрах всей сцепки транспортных средств (автопоезда), на которую оформляется специальное разрешение;
* WIDTH\_TS – ширина сцепки транспортных средств в метрах;
* HEIGTH\_TS – высота в метрах самого высокого транспортного средства в сцепке;
* JUT\_GRUZ – выступ груза за габарит транспортного средства в метрах;
* CNT\_TRIP – количество рейсов, на которое оформляется специальное разрешение;
* DT\_SROK – дата, начиная с которой будет производиться перевозка;
* CNT\_DAY – количество дней, в течение которых действительно специальное разрешение (максимум 30) или количество месяцев, в течение которых действительно многоразовое специальное разрешение;
* WHO\_FILL – ответственный за оформление заявления;
* PHONE\_FILL – номер телефона ответственного за оформление заявления;
* NUM\_REQUEST – номер заявления с начала года для определённой областной группы УБЭАД;
* SUM\_CALC – сумма платы за специальное разрешение по расчёту;
* SUM\_CALC\_REGISTR – сумма платы за оформление специального разрешения;
* IS\_REUSABLE – поле, указывающее, подаётся ли заявление на многоразовое специальное разрешение;
* NM\_ROUT – краткое описание маршрута движение;
* NOTE\_REQUEST – примечание к заявлению.

Данной таблице соответствует класс доменной модели REQUEST, содержащий поля, подобные вышеописанным полям таблицы.

* + 1. Таблица ORG

Данная таблица содержит данные об организациях.

Поле первичного ключа: IDORG – содержит уникальный идентификатор организации.

Поля:

* UNP – учётный номер плательщика (УНП);
* ADDRESS – юридический адрес организации;
* ORGNAME – наименование организации;
* ORGTYP – тип организации;
* ORG\_EMAIL – адрес электронной почты организации.

Данной таблице соответствует класс доменной модели Organization.

* + 1. Таблица DEPARTMENT

Таблица содержит сведения о подразделениях организаций.

Поле первичного ключа: IDDEPARTMENT – содержит уникальный идентификатор подразделения организации.

Поля внешних ключей:

* IDORG – идентификатор организации, к которой относится подразделение. Поле ссылается на первичный ключ IDORG таблицы ORG.

Поля:

* DEPNM – наименование подразделения;
* ORD – номер подразделения в пределах иерархии.

Данной таблице соответствует класс доменной модели SafeRoadControlGroup, который представляет областную группу УБЭАД.

* + 1. Таблица WEB\_USER

Таблица содержит сведения о зарегистрированных в системе пользователях.

Поле первичного ключа: IDWEBUSER – содержит уникальный идентификатор зарегистрированного пользователя.

Поля внешних ключей:

* IDORG – идентификатор организации, в которой состоит зарегистрированный пользователь. Поле ссылается на первичный ключ IDORG таблицы ORG.

Поля:

* FIRSTNAME\_RUS – имя зарегистрированного пользователя;
* LASTNAME\_RUS – фамилия зарегистрированного пользователя;
* PARNAME – отчество зарегистрированного пользователя;
* ID\_NUM – личный (идентификационный) номер по паспорту;
* EMAIL – адрес электронной почты пользователя.

Данной таблице соответствует класс доменной модели AuthUser, который представляет авторизированного пользователя в рамках определённого HTTP-запроса.

* + 1. Таблица VEHICLE

Данная таблица содержит сведения о транспортных средствах, на которые оформляются заявления.

Поле первичного ключа: IDVEHICLE – содержит уникальный идентификатор транспортного средства.

Поля внешних ключей:

* IDREQUEST – идентификатор заявления, к которому относится данное транспортное средство. Поле ссылается на первичный ключ IDREQUEST таблицы REQUEST;
* TYPE\_TS – идентификатор класса (типа) транспортного средства. Поле ссылается на первичный ключ IDCLSTKTS таблицы CLSTKTS.

Поля:

* STATE\_NUM – номер транспортного средства в сцепке по порядку;
* MARKA\_TS – марка транспортного средства;
* MODEL\_TS – модель транспортного средства;
* MASSA\_TS – масса транспортного средства;

Данной таблице соответствует класс доменной модели Vehicle.

* + 1. Таблица AXIS

Данная таблица содержит сведения о параметрах осей транспортного средства.

Поле первичного ключа: IDAXIS – содержит уникальный идентификатор оси транспортного средства.

Поля внешних ключей:

* IDVEHICLE – идентификатор транспортного средства, к которому относится данная ось. Поле ссылается на первичный ключ IDVEHICLE таблицы VEHICLE;
* TYPE\_AXIS – идентификатор класса (типа) оси транспортного средства. Поле ссылается на первичный ключ IDCLSTKTS таблицы CLSTKTS.

Поля:

* NUM\_AXIS – порядковый номер оси в сцепке транспортных средств;
* LOAD\_AXIS – нагрузка на ось в тоннах;
* M\_AXIS – межосевое расстояние в метрах;
* IS\_AXIS\_PNEUMO – поле, указывающее, имеется ли у оси пневмоподвеска;
* IS\_AXIS\_LEAD – поле, указывающее, является ли ось ведущей.

Данной таблице соответствует класс доменной модели Axis.

* + 1. Таблица VEHICLE\_FEATURE

Таблица содержит сведения об особенностях транспортного средства, на которое оформляется заявление. К ним могут относится, например, наличие изотермического кузова или 40-футового ISO-контейнера.

Поле первичного ключа: IDVEHICLE\_FEATURE – содержит уникальный идентификатор особенности транспортного средства.

Поля внешних ключей:

* IDVEHICLE – идентификатор транспортного средства, к которому относится данная особенность. Поле ссылается на первичный ключ IDVEHICLE таблицы VEHICLE;
* FEATURE – идентификатор класса (типа) особенности транспортного средства. Поле ссылается на первичный ключ IDCLSTKTS таблицы CLSTKTS.

Данной таблице соответствует класс доменной модели VehicleFeature.

* + 1. Таблица STATUS

Таблица содержит сведения о статусах заявлений на выдачу специального разрешения.

Поле первичного ключа: IDSTATUS – содержит уникальный идентификатор статуса заявления.

Поля внешних ключей:

* IDREQUEST – идентификатор заявления, к которому относится данная запись о статусе. Поле ссылается на первичный ключ IDREQUEST таблицы REQUEST;
* TYPE\_STATUS – идентификатор класса (типа) статуса. Поле ссылается на первичный ключ IDCLSTKTS таблицы CLSTKTS.

Поля:

* DT\_STATUS – дата получения статуса;
* LOGIN – логин пользователя, установившего данный статус;
* NOTE – примечание к статусу;

Таблице соответствует класс доменной модели Status.

* + 1. Таблица CLSTKTS

Данная таблица является классификатором и содержит записи классов или типов объектов с их текстовыми описаниями.

Поле первичного ключа: IDCLSTKTS – содержит уникальный идентификатор класса или типа.

Поля:

* IDGROUP – идентификатор группы классификатора, которая представляет классы или типы одной сущности;
* NUM – номер класса или типа классифицируемой сущности;
* CLSTEXT – краткое текстовое описание класса;
* BIGTEXT – полное текстовое описание класса.

Таблице соответствует класс доменной модели TktsСlassifier.

* + 1. Таблица PAYMENT

Таблица содержит записи об оплате специального разрешения для определённого заявления.

Поле первичного ключа: IDPAYMENT – содержит уникальный идентификатор записи об оплате.

Поля внешних ключей:

* IDREQUEST – идентификатор заявления, к которому относится данная запись об оплате. Данный внешний ключ ссылается на первичный ключ IDREQUEST таблицы REQUEST.

Поля:

* BASIS – основание для оплаты;
* DT\_PAYMENT – дата оплаты;
* NUM\_DOC – номер документа об оплате;
* CURRENCY – валюта оплаты;
* SUM\_PAYMENT – сумма оплаты;
* BIK – код банка.

Таблице соответствует класс доменной модели – Payment.

* + 1. Таблица PERMIT

Таблица содержит сведения о выданных специальных разрешениях.

Поле первичного ключа: IDPERMIT – содержит уникальный идентификатор специального разрешения.

Поля внешних ключей:

* IDREQUEST – идентификатор заявления, к которому относится данное специальное разрешение. Поле ссылается на первичный ключ IDREQUEST таблицы REQUEST.

Поля:

* DT\_PERMIT – дата выдачи специального разрешения;
* NUM\_PERMIT – номер специального разрешения;
* NM\_ROUT – краткое описание маршрута движения для данного специального разрешения.

Таблице соответствует класс доменной модели Permit.

* + 1. Таблица ROUT

Таблица содержит записи о точках маршрута движения транспортных средств, на которые оформляется заявление.

Поле первичного ключа: IDROUT – содержит уникальный идентификатор точки маршрута.

Поля внешних ключей:

* IDREQUEST – идентификатор заявления, к которому относится данная точка маршрута. Поле ссылается на первичный ключ IDREQUEST таблицы REQUEST;
* IDTN – идентификатор населённого пункта, к которому относится данная точка маршрута. Поле ссылается на первичный ключ OBJECT\_NUMBER таблицы ATE\_TE;
* IDRD – идентификатор дорожного объекта (дороги, транспортной развязки, подъезда к дороге), к которому относится данная точка маршрута. Поле ссылается на первичный ключ IDRD таблицы RD;
* IDRDCHECKPNT – идентификатор пункта пропуска через государственную границу. Поле ссылается на первичный ключ IDRDCHECKPNT таблицы RDCHECKPNT.

Поля:

* RDKM – километр дороги, на котором находится точка маршрута;
* LONGITUDE – географическая долгота;
* LATITUDE – географическая широта;
* NUM – порядковый номер точки маршрута;
* NOTE – примечание к точке маршрута.

Таблице соответствует класс доменной модели Route.

* + 1. Таблица RD

Таблица содержит сведения о дорогах Республики Беларусь.

Поле первичного ключа: IDRD – содержит уникальный идентификатор дороги (транспортной развязки, подъезда к дороге).

Поля:

* RDNAME – наименование дороги;
* RDNM – номер дороги;
* RDKIND – тип дороги (дорога, транспортная развязка или подъезд к дороге).

Таблице соответствует класс доменной модели Road.

* + 1. Таблица RDCHECKPNT

Таблица содержит сведения о пунктах пропуска через государственную границу.

Поле первичного ключа: IDRDCHECKPNT – содержит уникальный идентификатор пункта пропуска.

Поля:

* NAME – наименование пункта пропуска.

Таблице соответствует класс доменной модели Checkpoint.

* + 1. Таблица ATE\_TE

Таблица содержит сведения об административно-территориальных единицах Республики Беларусь.

Поле первичного ключа: OBJECT\_NUMBER – содержит уникальный идентификатор административно-территориальной единицы.

Поля:

* OBJECT\_NAME – наименование административно-территориальной единицы;
* OBJECT\_NAME\_FULL – полное наименование административно-территориальной единицы;
* ATE\_TYP – тип административно-территориальной единицы.

Таблице соответствует класс доменной модели AteTe.

* 1. Классы работы с базой данных и доступа к данным

К категории классов работы с базой данных прежде всего относятся классы, наследуемые от класса AbstractManager. Каждый из них производит операции чтения и записи в базу данных близкой группы сущностей.

Вызовы методов всех классов, наследуемых от AbstractManager, объединены в один класс – TktsRequestService, который имеет доступ к контексту текущего HTTP-запроса. TktsRequestService наследуется от TemplateService.

Класс доступа к данным DataProvider инкапсулирует в себе методы класса TktsRequestService и некоторые другие, а также производит преобразование типов данных.

Далее подробно рассмотрен каждый из упомянутых классов.

* + 1. Класс AbstractManager

Работа с базой данных производится через библиотеку Nhibernate. В статическом конструкторе класса AbstractManager создаётся объект типа ISessionFactory, так называемая фабрика сессий, являющаяся частью библиотеки Nhibernate. Фабрика сессий позволяет получать объекты типа ISession, через методы которого происходит непосредственно работа с базой данных.

Объект сессии Nhibernate создаётся в конструкторе класса AbstractManager при создании объекта производного класса. Класс AbstractManager реализует интерфейс IDisposable через метод Dispose, в котором разрывается соединение с базой данных, и сессия закрывается.

Поля:

* Logger – поле типа BaseLogger, представляющее собой ссылку на объект класса регистратора действий;
* CurrentUser – поле типа AuthUser, содержащее ссылку на объект, содержащий информацию о текущем авторизированном пользователе;
* ConnectionString – поле типа string, представляющее собой строку подключения, которая содержит информацию, необходимую для подключения Nhibernate к базе данных;
* SessionFactory – поле типа ISessionFactory, содержащее ссылку на объект фабрики сессий Nhibernate;
* session – поле типа ISession, содержащее ссылку на объект текущей сессии Nhibernate;
* connection – поле типа IDbConnection, содержащее ссылку на объект, представляющий собой открытое соединения к источнику данных.

Свойства:

* IdWebUser – свойство типа int, реализующее защищённый от записи доступ к идентификатору текущего авторизированного пользователя с использованием поля CurrentUser;
* Session – свойство типа ISession, реализующее защищённый от записи доступ к полю session.

Методы:

* static AbstractManager() – статический конструктор класса, в котором выполняется настройка Nhibernate, создание фабрики сессий;
* AbstractManager(ISession session, AuthUser user) – конструктор класса, выполняющий создание сессии и подключения к базе данных;
* void Dispose() – метод, производящий отключение от базы данных и закрытие сессии;
* T SelectSingle<T>(int idItem) – метод, позволяющий получить из базы данных объект типа T по его идентификатору;
* int InsertEntity<T>(T entity) – метод, позволяющий вставить объект entity типа T в базу данных;
* bool UpdateEntity<T>(T entity) – метод, позволяющий обновить в базе данных строку, соответствующую объекту entity типа T;
* bool DeleteEntity<T>(T entity) – метод, позволяющий удалить из базы данных строку, соответствующую объекту entity типа T.
  + 1. Класс GeoManager

Класс предоставляет возможность работы с различными географическими объектами (страны, населённые пункты, дороги, пункты пропуска). Наследуется от AbstractManager.

Поля:

* AteTypeOfTown – константа типа int, задающая тип административно-территориальной единицы – населённый пункт.

Методы:

* GeoManager(AuthUser user) и GeoManager(ISession session, AuthUser user) – конструкторы класса;
* IList<AteTe> GetTownsByName(string name, bool isExactMatch) – метод, возвращающий список населённых пунктов по полному или частичному совпадению имени (в зависимости от значения параметра isExactMatch);
* IEnumerable<CountryShort> GetCountriesRus() – метод, возвращающий список всех стран мира на русском языке;
* IEnumerable<CountryShort> GetCountriesEng() – метод, возвращающий список всех стран мира на английском языке;
* IList<Checkpoint> GetCheckpointsByName(string name) – метод, возвращающий список пунктов пропуска через государственную границе по частичному совпадению имени;
* IList<Road> GetRoadsByFilter(RoadFilter filter) – метод, возвращающий список дорог и дорожных объектов (транспортных развязок, подъездов к дорогам) по объекту-фильтру, передаваемому через параметр метода.
  + 1. Класс OrgManager

Класс используется для работы с организациями и организационными структурами. Наследуется от AbstractManager.

Поля:

* StreetGroupId – константа типа int, которая определяет группу в справочнике базы данных, соответствующую типам улиц.

Методы:

* OrgManager(AuthUser user) и OrgManager(ISession session, AuthUser user) – конструкторы класса;
* IList<Organization> GetOrganizationsForCurrent Organization() – метод, возвращающий список организаций, доступных для работы с организацией авторизированного пользователя (к ним относятся все организации, которые фигурировали в заявлениях на специальное разрешения в качестве перевозчиков, а также сама организация текущего пользователя);
* IList <SafeRoadControlGroup> GetSafeRoadControl Groups() – метод, возвращающий список областных групп управления безопасной эксплуатации автомобильных дорог (УБЭАД);
* IList<Carrier> GetCarriersForCurrentOrganization() – метод, возвращающий список организаций-перевозчиков, фигурировавших в заявлениях организации авторизированного пользователя в качестве перевозчика;
* IList<Carrier> GetCarriersByUnp(string unp) – метод, возвращающий список организаций-перевозчиков по их учётному номеру плательщика (УНП);
* IList<Carrier> GetCarriersByName(string name, bool isExactMatch) – метод, возвращающий список организаций-перевозчиков по частичному или полному совпадению их наименования с параметром name;
* Carrier GetCarrierById(int id) – метод, возвращающий объект, представляющий собой информацию об организации-перевозчике с заданным идентификатором;
* int InsertCarrierOrganization (Organization carrier) – метод, добавляющий в базу данных запись о перевозчике;
* int UpdateCurrentOrganization (Organization carrier) – метод, обновляющий запись в базе данных, соответствующую организации-перевозчику, заданной в параметре метода;
* StreetTypeInfo GetStreetTypeInfoByStreetType(int streetType) – вспомогательный метод, возвращающий объект, предоставляющий информацию о типе улицы (используется при заполнении адреса организации).
  + 1. Класс ReportManager

Данный класс предоставляет данные для создания отчётов и документов. В частности, доступны данные для создания заявления, листов согласования, технического заключения, расчёта платы. Класс наследуется от AbstractManager.

Методы:

* ReportManager (AuthUser user) и ReportManager (ISession session, AuthUser user) – конструкторы класса;
* DataTable GetDataForCrystalReport(int requestId, ReportType reportType) – возвращает таблицу с данными для определённого типа документа и заявления с заданным идентификатором;
* int GetReportDatabaseType(ReportType type) – вспомогательный метод, возвращающий номер типа отчёта, используемый в хранимой процедуре базы данных, по типу отчёта, заданному в аргументе метода.
  + 1. Класс RequestManager

Данный класс используется для работы с объектами, описывающими заявления на получение специального разрешения. Наследуется от AbstractManager.

Поля:

* RequestStatusGroupId – константа типа int, которая определяет группу в справочнике базы данных, соответствующую статусам заявки.

Методы:

* RequestManager (AuthUser user) и RequestManager (ISession session, AuthUser user) – конструкторы класса;
* Request GetRequest(int id) – метод, возвращающий объект заявления по его идентификатору;
* IList<RequestsListElement> GetRequestsList() – метод, возвращающий список поданных заявлений, находящихся на рассмотрении;
* IList<RequestsListElement> GetUnissuedRequests (DateTime startDate, DateTime endDate) – метод, возвращающий список заявлений, по которым отказано в выдаче специального разрешения, поданных в диапазоне дат, задаваемым параметрами метода startDate и endDate;
* IList<RequestsListElement> GetUnissuedRequests (RequestFilter filter) – метод, возвращающий список заявлений, по которым отказано в выдаче специального разрешения и отфильтрованных согласно объекту-фильтру, заданному в параметре метода;
* IList<RequestsListElement> GetPermits(DateTime startDate, DateTime endDate, int pageNumber, int recordsPerPage, out int pagesCount) – метод, возвращающий список заявлений, по которым выданы специальные разрешения в диапазоне дат, задаваемых параметрами метода startDate и endDate, а также относящихся к странице pageNumber при количестве записей на страницу recordsPerPage;
* IList<RequestsListElement> GetPermits (RequestFilter filter) – метод, возвращающий список заявлений, по которым выданы специальные разрешения, отфильтрованный согласно объекту-фильтру, заданному в параметре метода;
* int InsertRequest (Request request) – метод, сохраняющий заявление и все связанные с ним объекты (транспортные средства, оси, маршрутные точки) в базе данных и возвращающий идентификатор сохранённого заявления;
* void UpdateRouteTextForRequest(int requestId) – вспомогательный метод, сохраняющий в базу данных краткий маршрут на основе совокупности маршрутных точек из сохранённого ранее заявления;
* int DeleteRequest(int id) – метод, удаляющий заявление и все связанные с ним объекты из базы данных;
* int AddPayment(Payment payment) – метод, сохраняющий в базе данных запись об оплате специального разрешения;
* int DeletePayment(int id) – метод, удаляющий из базы данных запись об оплате специального разрешения по её идентификатору;
* IList<Payment> GetPayments(int requestId) – метод, возвращающий список оплат для заявления с заданным идентификатором.
  + 1. Класс VehiclesManager

Класс используется для работы с объектами, описывающими транспортные средства и связанные с ними объекты (оси транспортного средства, особенности транспортного средства). Наследуется от AbstractManager.

Поля:

* VehicleGroupId – константа типа int, которая определяет группу в справочнике базы данных, соответствующую типам транспортных средств;
* AxisGroupId – константа типа int, которая определяет группу в справочнике базы данных, соответствующую типам осей транспортных средств;
* FeatureGroupId – константа типа int, которая определяет группу в справочнике базы данных, соответствующую типам особенностей транспортных средств.

Методы:

* VehiclesManager (AuthUser user) и VehiclesManager (ISession session, AuthUser user) – конструкторы класса;
* IList<Vehicle> GetCarrierVehicles(int carrierId) – метод, возвращающий список транспортных средств, принадлежащих организации-перевозчику с идентификатором, который указан в параметре метода;
* TktsСlassifier GetVehicleСlassifierByVehicleType (int vehicleType) – метод, возвращающий объект-классификатор транспортного средства;
* IList<Axle> GetAxlesByVehicleId(int vehicleId) – метод, возвращающий список осей транспортного средства по его идентификатору;
* TktsСlassifier GetAxleСlassifierByAxleType (int axleType) – метод, возвращающий объект-классификатор оси транспортного средства;
* TktsСlassifier GetFeatureСlassifierByFeatureType (int featureType) – метод, возвращающий объект-классификатор особенности транспортного средства.
  + 1. Класс TemplateService

Класс является базовым классом для сервиса доступа к данным. Имеет доступ к контексту текущего HTTP-запроса, и таким образом, может получать объект текущего авторизированного пользователя и передавать его в конструктор классов, наследуемых от AbstractManager.

Свойства:

* Principal – свойство типа BdcAppPrincipal, предоставляющее доступ к специальному объекту, описывающему текущего авторизированного пользователя.

Методы:

* AuthUser GetUser() – метод, возвращающий объект, описывающий авторизированного пользователя.
  + 1. Класс TktsRequestService

Данный класс инкапсулирует вызовы всех методов классов, наследуемых от AbstractManager. При вызове каждого из методов создаётся объект соответствующего класса-менеджера, при этом в конструктор передаётся объект текущего авторизированного пользователя. Также создание объектов классов-менеджеров и вызов соответствующего метода происходит внутри оператора using, который гарантирует вызов метода Dispose, в котором закрывается сессия Nhibernate и разрывается соединение с базой данных.

Пример метода класса TktsRequestService:

public DataTable GetDataForCrystalReport(int requestId,

ReportType reportType)

{

using (var manager = new ReportManager(GetUser()))

{

return manager

.GetDataForCrystalReport(requestId, reportType);

}

}

* + 1. Класс DataProvider

Данный класс предоставляет данные непосредственно для классов контроллеров, получая данные от класса TktsRequestService, а также генерируя некоторые объекты самостоятельно. Для некоторых методов производит преобразования объектов классов доменных моделей (которые соответствуют таблицам в базе данных) в объекты моделей, более близким по структуре к данным, используемым в представлениях интерфейса. В моделях, в которые преобразует данный класс, содержится вся логика валидации данных (в виде атрибутов MVC).

Поля:

* \_requestService – поле типа TktsRequestService, содержащее ссылку на объект, предоставляющий доступ к объектам приложения из базы данных.

Методы:

* IEnumerable <SafeRoadControlGroup> GetSafeRoad ControlGroups() – метод, возвращающий список областных групп управления безопасной эксплуатации автомобильных дорог (УБЭАД);
* SelectList GetSafeRoadControlGroupsSelectList() – метод, возвращающий список групп УБЭАД в виде списка, элементы которого представляют собой значение и описание, а также один из элементов может быть выбран пользователем;
* IList <CarrierModel> GetCarriersForCurrent Organization() – метод, возвращающий список перевозчиков, которые фигурировали в заявлениях организации текущего авторизированного пользователя;
* IList<CarrierModel> GetCarriersByUnp(string unp) – метод, возвращающий список перевозчиков по совпадения учётного номера плательщика (УНП);
* IList<CarrierModel> GetCarriersByName(string name, bool isExactMatch) – метод, возвращающий список перевозчиков по полному или частичному совпадению их наименования с параметром name (при isExactMatch == true – полное совпадение, иначе – частичное);
* CarrierModel GetCarrierById(int carrierId) – метод, возвращающий объект, описывающий перевозчика с заданным в параметре метода идентификатором;
* IList <OrganizationModel> GetOrganizationsFor CurrentOrganization() – метод, возвращающий список организаций, доступных для работы с организацией текущего авторизированного пользователя;
* int AddNewCarrierOrganization (OrganizationModel orgModel) – метод, добавляющий новую организацию-перевозчика;
* int UpdateCurrentOrganization (OrganizationModel orgModel) – метод, обновляющий данные организации авторизированного пользователя;
* IList <VehicleModel> GetCarrierVehicles (int carrierOrgId) – метод, возвращающий список моделей представления транспортных средств, принадлежащих перевозчику с заданным в параметре идентификатором;
* IList<VehicleModel> GetVehicleModelsFromVehicles (IList<Vehicle> vehicles) – метод, преобразующий список объектов доменных моделей транспортных средств в список моделей представления транспортных средств;
* IList<AxleModel> GetVehicleAxles(int vehicleId, int requiredAxlesCount) – метод, возвращающий список с заданным количеством осей транспортного средства по его идентификатору;
* IList <VehicleAxlesModel> GetVehicleAxlesModels FromVehicles (IList<Vehicle> vehicles) – метод, преобразующий список объектов доменных моделей транспортных средств в список моделей представления совокупностей осей транспортных средств;
* RequestInfoModel GetDefaultRequestInfoModel() – метод, возвращающий объект модели с предварительной информацией о заявлении по умолчанию;
* VehicleDetailsModel GetDefaultVehicleDetailsModel() – метод, возвращающий объект модели с составом автопоезда и его принадлежностью по умолчанию;
* IList<FeatureOption> GetDefaultFeatureOptions() – метод, возвращающий список особенностей транспортного средства по умолчанию (все особенности считаются отсутствующими);
* IList<AxleModel> GetDefaultAxles(VehicleModel vehicle) – метод, возвращающий список осей транспортного средства с характеристиками по умолчанию;
* SelectList GetCountriesSelectList() – метод возвращающий список всех стран мира;
* IList<AteTe> GetTownsByName(string name, bool isExactMatch) – метод, возвращающий список населённых пунктов по полному или частичному совпадению имени;
* IEnumerable<RoutePointModel> GetRoadRoutePoints (string name, RoutePointType routePointType) – метод, возвращающий список маршрутных точек, относящихся к категории дорожных объектов (дороги, транспортные развязки, подъезды к дорогам), по наименованию или номеру;
* string GetRoadFullName(string roadNumber, string roadName) – вспомогательный метод, возвращающий полное наименование дороги по её названию и номеру;
* IEnumerable<RoutePointModel> GetTownRoutePoints (string name) – метод, возвращающий список маршрутных точек, относящихся к категории населённых пунктов;
* bool CheckIfTownLargeByType(LargeTownType type) – метод, проверяющий, относится населённый пункт данного типа к районным центрам;
* IEnumerable<RoutePointModel> GetCheckPoints(string name) – метод, возвращающий список маршрутных точек по совпадению наименования, относящихся к категории пунктов пропуска через государственную границу;
* IList <RoutePointModel> GetRoutePointModelsFrom Routes(IList<RoutePoint> routePoints) – метод, преобразующий список объектов доменных моделей маршрутных точек в список моделей представления маршрутных точек;
* IList <RequestsListElement> GetRequestsList() – метод, возвращающий список поданных заявлений организации авторизированного пользователя, находящихся на рассмотрении;
* IList<RequestsListElement> GetUnissuedRequestsList (DateTime startDate, DateTime endDate) – метод, возвращающий список заявлений, по которым отказано в выдаче специального разрешения, поданных в диапазоне дат, заданному параметрами startDate и endDate;
* IList<RequestsListElement> GetUnissuedRequestsList (RequestFilter filter) – метод, возвращающий список заявлений, по которым отказано в выдаче специального разрешения, отфильтрованных согласно объекту-фильтру, заданному в параметре метода;
* IList<RequestsListElement> GetPermits(DateTime startDate, DateTime endDate, int pageNumber, int recordsPerPage, out int pagesCount) – метод, возвращающий список заявлений, по которым выданы специальные разрешения, поданных в диапазоне дат, заданных параметрами метода и относящихся к странице, заданной параметром pageNumber; причём количество возращённых заявлений не превышает значения параметра recordsPerPage;
* IList <RequestsListElement> GetPermits (RequestFilter filter) – метод, возвращающий список заявлений, по которым выданы специальные разрешения, отфильтрованных согласно объекту-фильтру, заданному в параметре метода;
* bool SendRequest(RequestModel requestModel) – метод, отправляющий заявление на рассмотрение;
* int DeleteRequest(int id) – метод, удаляющий заявление, у которого идентификатор равен id;
* RequestModel GetRequestModel(int id) – метод, возвращающий объект модели заявления на специальное разрешение;
* DataTable GetDataForCrystalReport(int requestId, ReportType reportType) – метод, возвращающий таблицу с данными для создания документа для дальнейшего скачивания пользователем;
* PreviewModel GetRequestPreview (RequestModel requestModel) – метод, возвращающий модель для предварительного просмотра данных заявления перед отправкой;
* int AddPayment(PaymentModel payment) – метод, добавляющий запись об оплате для заявления;
* int DeletePayment(int id) – метод, удаляющий запись об оплате с заданным идентификатором;
* IList<PaymentModel> GetPayments(int requestId) – метод, возвращающий список записей об оплате для заявления с заданным идентификатором.

1. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ
   1. Авторизация

Одной из важнейших частей приложения является блок авторизации. Он представлен классом AuthorizeBdcAppAttribute, являющимся фильтром авторизации. Данный класс также реализует аутентификацию, то есть проверку подлинности данных пользователя. Класс наследуется от встроенного в ASP.NET MVC фильтра AuthorizeAttribute, переопределяя метод AuthorizeCore, который определяет, авторизирован пользователь или нет:

protected override bool

AuthorizeCore(HttpContextBase httpContext)

{

if (httpContext == null)

return false;

if (!httpContext.User.Identity.IsAuthenticated)

return false;

if (Authorize.User == null)

return false;

return \_actions.Length <= 0 ||

Authorize

.HaveActions(EnterElements.Any, \_actions);

}

В приведённом методе можно увидеть, что сначала проверяется, аутентифицирован ли пользователь. Свойство User.Identity, относящееся к контексту текущего HTTP-запроса автоматически устанавливается встроенным модулем аутентификации форм (Forms Authentication), если в браузере пользователя установлен аутентификационный cookie-набор. Затем производится обращение к свойству User класса Authorize, которое возвращает объект класса AuthUser, представляющий текущего авторизированного пользователя. Далее проверяется наличие привилегий у текущего пользователя с помощью метода HaveActions. Листинг свойства User показан ниже.

public static AuthUser User

{

get {

if (HttpContext.Current == null ||

!HttpContext.Current.Request.IsAuthenticated)

return null;

if (HttpContext.Current

.Session[SessionCurrentUser] == null)

HttpContext.Current.Session[SessionCurrentUser]

= GetCurrentUser();

return HttpContext.Current

.Session[SessionCurrentUser] as AuthUser;

}

}

Свойство User возвращает объект авторизированного пользователя из текущей сессии. При его отсутствии производится загрузка объекта в сессию из базы данных с помощью метода GetCurrentUser, который в свою очередь использует общий для нескольких приложений SystemService.

* 1. Оформление заявления

Реализация процесса оформления заявления является главной целью данного приложения. За это отвечает класс MakeRequestController, который является классом контроллера и наследуется от встроенного в ASP.NET MVC класса Controller. Он содержит методы действий, которые обрабатывают входящие клиентские запросы, соответствующие различным шагам подачи заявления.

На первом шаге подачи заявления для пользователя формируется страница с полями, содержащими общую информацию о заявке:

[HttpGet]

public ActionResult RequestInfo()

{

var requestInfo =

\_cookieManager.GetObject<RequestInfoModel>

(RequestStep.RequestInfo) ??

\_dataProvider.GetDefaultRequestInfoModel();

return RequestInfoPage(requestInfo);

}

Данный метод вызывается при заходе пользователя на первую страницу подачи заявления. Атрибут метода соответствует типу HTTP-запроса GET. При наличии файла cookie, соответствующего данному шагу, из него загружается объект модели, который отправляется для генерации представления для пользователя. Иначе – отправляется объект модели, заполненный значениями по умолчанию. Данная логика характерна для всех GET-методов, соответствующих шагам оформления заявления.

При отправке данных с первого шага подачи заявления выполняется метод, соответствующий HTTP-запросу POST:

[HttpPost]

Public ActionResult

RequestInfo(RequestInfoModel requestInfo)

{

if (!ModelState.IsValid)

return RequestInfoPage(requestInfo);

\_cookieManager

.SetReusableCookie(requestInfo.IsReusable);

\_cookieManager

.SaveObject(RequestStep.RequestInfo, requestInfo);

return RedirectToAction("VehicleDetails");

}

Параметром метода является объект модели, заполненной данными, отправляемыми клиентом (браузером). Как можно видеть в листинге метода, сначала с помощью свойства ModelState.IsValid проверяется прошла ли модель валидацию. Далее с использованием объекта класса RequestCookieManager устанавливается файл cookie, указывающее на тип специального разрешения (многоразовое или обычное), а также в cookie сохраняется объект полученной от клиента модели.

Упомянутое выше свойство ModelState.IsValid устанавливается на стадии привязки модели – перед запуском метода действия фреймворк ASP.NET MVC производит сопоставление данных из HTTP-запроса с полями параметров метода действия, а затем производит их валидацию. Логика валидации может содержится в специальных атрибутах полей модели. Также валидация может быть реализована в методе Validate модели, при этом класс модели реализует интерфейс IValidatableObject.

Отрывок класса модели RequestInfoModel:

public class RequestInfoModel : IValidatableObject

{

[Display(Name = "RequestInfo\_DateOfTrasportation",

ResourceType = typeof(Resources.Models))]

[Required(ErrorMessageResourceType =

typeof(Resources.MakeRequestValidation),

ErrorMessageResourceName = "RequiredMessage")]

public DateTime DateOfTrasportation { get; set; }

[Display(Name = "RequestInfo\_CountOfTrips",

ResourceType = typeof(Resources.Models))]

[Required(ErrorMessageResourceType =

typeof(Resources.MakeRequestValidation),

ErrorMessageResourceName = "RequiredMessage")]

[Range(1, 999, ErrorMessageResourceType =

typeof(Resources.MakeRequestValidation),

ErrorMessageResourceName = "RangeMessage")]

public int CountOfTrips { get; set; }

// остальные поля...........

public IEnumerable<ValidationResult>

Validate(ValidationContext validationContext)

{

if (DateOfTrasportation < DateTime.Today ||

DateOfTrasportation > DateTime.Today.AddMonths(2))

yield return new ValidationResult

(Resources.MakeRequestValidation

.RequestInfoModel\_DateOfTrasportation,

new[] { "DateOfTrasportation" });

}

}

Как можно видеть из вышеприведённого отрывка класса в качестве атрибутов валидации используются Required (значение поля не может быть пустым), Range (значение поля должно принимать значение из определённого в параметрах диапазона). В методе же Validate реализованы дополнительные проверки, которые не могут быть обеспечены атрибутами. Стоит отметить, что при использовании атрибутов возможна валидация на стороне клиента в браузере с использованием Microsoft jQuery Unobtrusive Validation – при рендеринге представлений для полей с атрибутами генерируется html-атрибуты с сообщениями валидации.

На втором шаге подачи заявления устанавливается владелец и состав сцепки транспортных средств.

[HttpGet]

public ActionResult VehicleDetails()

{

if (\_cookieManager.CheckPreviousCookies

(RequestStep.VehicleDetails) == false)

return RedirectToAction("RequestInfo");

var vehicleDetails = \_cookieManager

.GetObject<VehicleDetailsModel>(RequestStep.VehicleDetails)

?? \_dataProvider.GetDefaultVehicleDetailsModel();

return VehicleDetailsPage(vehicleDetails);

}

При этом на втором и последующих шагах при отсутствии файлов cookie, соответствующих предыдущим шагам подачи заявления, происходит перенаправление на первую страницу подачи заявления: return RedirectToAction("RequestInfo").

[HttpPost]

Public ActionResult

VehicleDetails (VehicleDetailsModel vehicleDetails,

string next, string back)

{

if (!ModelState.IsValid)

return VehicleDetailsPage(vehicleDetails);

var vehicleStructure = vehicleDetails.VehicleStructure;

if (vehicleStructure != null)

{

int minId = vehicleStructure.Select(vehicle =>

vehicle.Id).Concat(new int[] {0}).Min();

foreach (var vehicle in vehicleStructure

.Where(vehicle => vehicle.Id == 0))

vehicle.Id = --minId;

}

\_cookieManager.SaveObject(RequestStep.VehicleDetails,

vehicleDetails);

if (!String.IsNullOrEmpty(next)) // Next clicked

return RedirectToAction("VehicleScheme");

return RedirectToAction("RequestInfo"); // Back clicked

}

При отправке информации со страницы выбора транспортных средств вызывается приведённый выше метод VehicleDetails с атрибутом HttpPost. Помимо ранее описанных действий с полученной моделью здесь для каждого нового добавленного транспортного средства в сцепке, ранее отсутствовавшего в базе данных, назначается новый отрицательный идентификатор. Это делается для различения транспортных средств от транспортных средств с положительными идентификаторами, которые уже присутствуют в базе данных. Также на этом шаге и на последующих присутствуют параметры next и back, показывающие была ли нажата кнопка перехода на следующий или предыдущий шаг заявления соответственно.

[HttpGet]

public ActionResult VehicleScheme()

{

if (\_cookieManager.CheckPreviousCookies

(RequestStep.VehicleScheme) == false)

return RedirectToAction("RequestInfo");

var vehicleDetails = \_cookieManager.

GetObject<VehicleDetailsModel>(RequestStep.VehicleDetails);

if (vehicleDetails == null ||

vehicleDetails.VehicleStructure == null)

return RedirectToAction("VehicleDetails");

var vehicleScheme = \_cookieManager.

GetObject<VehicleSchemeModel>(RequestStep.VehicleScheme);

if (vehicleScheme == null)

vehicleScheme = \_dataProvider

.GetDefaultVehicleSchemeModel(vehicleDetails);

else \_dataProvider

.FillVehicleScheme(vehicleScheme, vehicleDetails);

return VehicleSchemePage(vehicleScheme);

}

На очередном шаге заполняется информация об осях транспортных средств, выбранных на предыдущем шаге и габаритные параметры сцепки транспортных средств (метод действия VehicleScheme). При этом для каждого транспортного средства, существующего в базе данных, загружаются значения параметров осей из базы данных (вызов метода \_dataProvider .GetDefaultVehicleSchemeModel(vehicleDetails)). Если же существует файл cookie, соответствующий данному шагу заявления (значение vehicleScheme не равно null), то из базы данных загружаются только значения осей для транспортных средств, отсутствующих в данном файле cookie (метод \_dataProvider.FillVehicleScheme(vehicleScheme, vehicleDetails)).

[HttpGet]

public ActionResult Route()

{

if (\_cookieManager

.CheckPreviousCookies(RequestStep.Route) == false)

return RedirectToAction("RequestInfo");

var routeModel =

\_cookieManager.GetObject<RouteModel>(RequestStep.Route)

?? new RouteModel();

if (\_cookieManager.CheckReusableCookie())

{

var routeForReusableModel =

new RouteForReusableModel() {Route = routeModel

.RouteForReusable};

return View("RouteForReusable",

routeForReusableModel);

}

return View("Route", routeModel);

}

В методе действия Route с атрибутом HttpGet на этапе выбора маршрута движения в зависимости от типа специального разрешения пользователю отправляется различные модель и представление. При этом данному этапу соответствуют два различных метода с атрибутом HttpPost, принимающих данные с маршрутами от пользователя: Route и RouteForReusable. Данные методы вызываются в зависимости от того, является ли специальное разрешение многоразовым. Логика данных методов повторяет вышеописанные методы с атрибутом HttpPost: валидация модели, сохранение объекта в cookie и перенаправление на следующий или предыдущий шаг заявления в зависимости от нажатой кнопки.

[HttpGet]

public ActionResult RequestPreview()

{

if (\_cookieManager.CheckPreviousCookies(RequestStep

.RequestPreview) == false)

return RedirectToAction("RequestInfo");

string note = \_cookieManager

.GetObject<string>(RequestStep.RequestPreview);

ViewData["Note"] = note;

var makeRequestModel = \_cookieManager.GetRequestModel();

var previewModel =

DataProvider.GetRequestPreview(makeRequestModel);

if(previewModel == null)

return RedirectToAction("RequestInfo");

return View(previewModel);

}

На последнем шаге в методе RequestPreview предоставляется предварительный просмотр всех данных заявления перед отправкой. При этом из всех файлов cookie, соответствующих предыдущим шагам заявления, собирается единая модель заявления (\_cookieManager.GetRequestModel()). Затем из неё собирается модель предварительного просмотра (DataProvider .GetRequestPreview(makeRequestModel)), из которой генерируется отправляемое клиенту представление. Примечание к заявлению, если оно было ранее введено и содержалось в файле cookie, извлекается оттуда и передаётся представлению через коллекцию объектов ViewData, которая доступна в представлении.

[HttpPost]

public ActionResult RequestPreview(string note, string next,

string back) {

if (!String.IsNullOrEmpty(next)) { // Send clicked

if (\_cookieManager.CheckPreviousCookies

(RequestStep.RequestPreview) == false)

return RedirectToAction("RequestInfo");

var requestModel =

\_cookieManager.GetRequestModel();

\_dataProvider.SendRequest(requestModel);

\_cookieManager.RemoveRequestCookies();

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

}

\_cookieManager // Back clicked

.SaveObject(RequestStep.RequestPreview, note);

return RedirectToAction("Route");

}

При отправке заявления со страницы предварительного просмотра используется метод RequestPreview с атрибутом HttpPost. При этом из всех cookie заявления собирается единая модель заявления (\_cookieManager .GetRequestModel()). Затем происходит сохранение модели в базе данных для дальнейшего рассмотрения (вызов \_dataProvider.SendRequest (requestModel)). Далее удаляются все cookie шагов заявления (вызов метода RemoveRequestCookies) и пользователь перенаправляется на страницу списка заявлений. Последовательность взаимодействия объектов во времени при отправке заявления показана на диаграмме последовательности (чертёж ГУИР.400201.050 РР.3).

* 1. Работа с cookie

При подаче заявления для временного хранения информации, отправленной пользователем, используются файлы cookie, отправляемые браузером при каждом HTTP-запросе. Это также позволяет продолжить оформление ранее не отправленного заявления, не внося уже введённую информацию. За работу с cookie отвечает класс RequestCookieManager.

public void SaveObject

(RequestStep requestStep, object objectForSave)

{

HttpCookie cookie =

CreateCookie(requestStep.ToString());

cookie.SetAndEncryptValue(objectForSave);

\_httpContext.Response.Cookies.Add(cookie);

}

Метод SaveObject сохраняет объект в cookie с именем, соответствующим шагу подачи заявления. При этом значение устанавливается методом расширения класса HttpCookie:

public static void SetAndEncryptValue(this HttpCookie cookie,

object objectValue)

{

string valueJsonString =

JsonConvert.SerializeObject(objectValue);

byte[] valueBytes = valueJsonString.GetBytes();

byte[] compressedValue = Compress(valueBytes);

cookie.Value =

HttpServerUtility.UrlTokenEncode(compressedValue);

}

Данный метод преобразует объект в текстовый формат JSON, сжимает полученную строку алгоритмом Deflate, затем преобразует данные в кодировку base64 (вызов метода UrlTokenEncode). Base64 представляет собой кодировку с использованием 64 печатных символов ASCII. Использование кодировки base64 необходимо, так как наличие не ASCII-символов в cookie может некорректно обрабатываться различными браузерами, что может повредить хранимые данные.

Чтение объекта из cookie производит метод GetObject:

public T GetObject<T>(RequestStep requestStep)

where T : class

{

HttpCookie cookie =

\_httpContext.Request.Cookies[requestStep.ToString()];

return cookie.TryGetDecryptedValue<T>();

}

В данном методе из контекста HTTP-запроса получается объект cookie, соответствующий определённому шагу заявления. Затем с помощью метода расширения класса HttpCookie производится попытка получить раскодированный объект:

public static T TryGetDecryptedValue<T>

(this HttpCookie cookie) where T : class

{

if (cookie == null) return null;

try

{

byte[] decodedValue =

HttpServerUtility.UrlTokenDecode(cookie.Value);

byte[] decompressedValue =

Decompress(decodedValue);

string jsonValue =

StringExtensions.GetString(decompressedValue);

return JsonConvert

.DeserializeObject<T>(jsonValue);

}

catch (Exception)

{

return null;

}

}

В методе TryGetDecryptedValue по сути выполняется последовательность операций, обратная последовательности в методе SetAndEncryptValue: раскодирование из base64, распаковка алгоритмом Deflate и преобразование из строки формата JSON в объект заданного типа. При любом возникшем исключении в описанных операциях возвращается пустое значение.

public void RemoveCookie(string name)

{

if (\_httpContext.Request.Cookies[name] == null)

return;

var c = new HttpCookie(name)

{ Expires = DateTime.Now.AddDays(-1) };

\_httpContext.Response.Cookies.Add(c);

}

public void RemoveRequestCookies()

{

foreach (var stepName in

Enum.GetNames(typeof(RequestStep)))

RemoveCookie(stepName);

RemoveCookie(IsReusableCookieName);

}

Метод RemoveCookie удаляет файл cookie с заданным именем путём записи в ответ одноимённого cookie с параметром срока действия, равным вчерашней дате. В результате файл cookie заменятся новым cookie и удаляется браузером в связи с истечением срока действия. Метод RemoveRequestCookies удаляет все cookie, соответствующие шагам подачи заявления, путём прохода в цикле по элементам перечисления шагов.

1. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Платформа ASP.NET MVC предоставляет богатые возможности по тестированию всех частей веб-приложения на всех уровнях разработки (от отдельной части программы до приложения в целом). Система тестирования включает в себя следующие возможности [8]:

1. Юнит-тесты, позволяющие быстро и автоматически протестировать отдельные участки кода независимо от остальной части программы. При надлежащем составлении юнит-тесты вполне могут покрыть большую часть кода приложения.
2. Интеграционные тесты. Даже если охватить юнит-тестами практически всю логику приложения, все равно могут быть моменты, которые будут работать не так, как надо. Кроме того, сложно протестировать представления. Подобные вещи тестируются с помощью интеграционных тестов, выполняющихся на уровне веб-браузера.
3. Нагрузочные тесты, которые призваны протестировать работу сайта в условиях высокой нагрузки. Подобные тесты позволяют выявить узкие места при работе с базой данных или при обращении к диску и ряд других проблем, которые сложно выявить другими способами.

В процессе работы над проектом заказчик решил отказаться от полноценного тестирования в угоду ускорению процесса разработки, поэтому тестирование данного проекта является исключительно функциональным.

В основном тестирование проводилось на веб-сервере IIS Express 8.0, входящим в состав среды разработки Visual Studio 2013. На IIS Express можно быстро запустить любой сайт из директории на диске. Не требуется никаких шагов регистрации или настройки. При отладке приложения Visual Studio автоматически запустит IIS Express и будет использовать его для запуска/отладки приложений. В отличии от IIS Web Server, который встроен в Windows, IIS Express не требует прав администратора для запуска/отладки приложений из Visual Studio. Однако стоит отметить, что проект планируется к публикации в системе Windows Server 2008 R2, в которую встроен IIS Web Server 7.5, таким образом, тестирование производилось в том числе и на IIS 7.5.

На стороне клиента тестирование производилось на всех современных наиболее популярных браузерах: Firefox, Google Chrome, Internet Explorer, Opera. Проверка производилась в том числе на минимально поддерживаемых версиях данных браузеров. Стоит отметить, что при использовании неподдерживаемого браузера пользователю предлагается обновить браузер на последнюю версию. Это избавляет от необходимости тестировать возможные непредвиденные ситуации при использовании устаревших браузеров.

Важную роль при тестировании сыграло информирование о необработанных исключительных ситуациях и их протоколирование. Для этого в ASP.NET MVC используются фильтры исключений. Они срабатывают, если при выполнении метода действия будет выброшено необработанное исключение. С одной стороны, можно поместить всю логику выполнения метода в блок try...catch и отследить исключение. Однако область работы фильтров исключений несколько шире. Они позволяют отследить не только исключения, возникающие в самом методе, но исключения, генерируемые результатами действий, а также другими применяемыми к данному действию фильтрами. В этом и состоит мощь данного типа фильтров.

Как и все остальные фильтры действий в ASP.NET MVC фильтры исключений устанавливаются в виде атрибутов над методами действий или классами контроллеров. Также можно установить атрибут глобально для всех контроллеров, зарегистрировав фильтр в методе RegisterGlobalFilters класса FilterConfig.

Все фильтры исключений должны применять интерфейс IExceptionFilter. Единственный метод OnException данного интерфейса вызывается, когда приложение выбрасывает необрабатываемое исключение. Во фреймворке есть встроенная реализация данного интерфейса – атрибут HandleErrorAttribute. В данном проекте в качестве фильтра исключения используется класс TktsHandleErrorAttribute, который наследуется от встроенного HandleErrorAttribute и переопределяет метод OnException:

public class TktsHandleErrorAttribute : HandleErrorAttribute

{

public override void

OnException(ExceptionContext filterContext)

{

if (filterContext.ExceptionHandled ||

!filterContext.HttpContext.IsCustomErrorEnabled)

{

return;

}

LoggerManager.GetLogger()

.Error(filterContext.Exception);

if (filterContext.HttpContext.Request

.IsAjaxRequest() && filterContext.Exception != null)

{

filterContext.HttpContext.Response.StatusCode =

(int)HttpStatusCode.InternalServerError;

filterContext.Result = new JsonResult

{

JsonRequestBehavior =

JsonRequestBehavior.AllowGet,

Data = new

{

filterContext.Exception.Message,

filterContext.Exception.StackTrace

}

};

filterContext.ExceptionHandled = true;

}

else

{

base.OnException(filterContext);

}

}

}

В метод OnException передаётся параметр ExceptionContext, который является объектом, производным от ControllerContext. Класс ExceptionContext обладает свойствами, позволяющими получить информацию об исключении:

* свойство, содержащее информацию о методе действия, на котором было выброшено исключение;
* свойство, представляющее само необработанное исключение;
* значение, показывающее, считается ли исключение обработанным;
* результат метода действия, к которому применяется фильтр исключения.

В методе OnException производится протоколирование ошибок через их запись в базу данных с использованием внешнего компонента LoggerManager из другого проекта: LoggerManager.GetLogger().Error (filterContext.Exception). В метод Error передаётся текущее необработанное исключение.

Далее проверяется, является ли HTTP-запрос запросом типа AJAX (Asynchronous Javascript and XML). При AJAX-запросе страница в браузере не перезагружается полностью, а изменения в неё вносит скрипт на языке JavaScript с учётом полученной в ответе информации. Таким образом, при возникновении необработанного исключения скрипту на стороне клиента в формате JSON передаётся информация об ошибке: filterContext .Result = new JsonResult{…}.

Если же HTTP-запрос не является AJAX-запросом, то вызывается метод OnException из встроенного базового класса HandleErrorAttribute. При этом фильтр исключений посылает статусный код HTTP 500, свидетельствующий о внутренней ошибке сервера, а также генерирует для клиента указанное представление. В данном проекте в представлении отображается информационное сообщение с подробностями ошибки.

Таким образом, несмотря на отсутствие юнит- и интеграционных тестов, в проекте были созданы условия для довольно информативного, хоть и не автоматизированного тестирования.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для работы с данным веб-приложением необходим любой современный браузер с включённым JavaScript. Минимальные поддерживаемые версии браузеров: Internet Explorer 9.0, Firefox 15, Opera 12.

Разрабатываемое веб-приложение планируется к размещению на сервере информационного центра дорожного хозяйства (i.centr.by). Регистрация и вход в систему производится там же. Затем необходимо выбрать приложение из списка в разделе информационных ресурсов.

При входе в приложение по умолчанию открыта вкладка «Список заявлений» (рисунок 6.1).

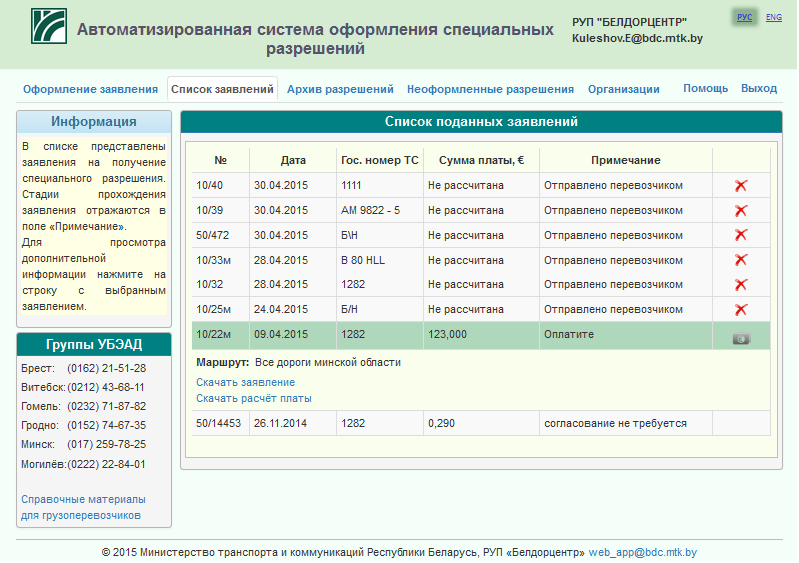


Рисунок 6.1 – Вкладка «Список заявлений»

В данной вкладке отображаются заявления, находящиеся на рассмотрении. Доступна информация о номере заявления, дате подачи, номере транспортного средства, сумме платы. В столбце «Примечание» отображается информация, позволяющая пользователю знать, на какой стадии находится в данный момент рассмотрение данного заявления. Для просмотра дополнительной информации о заявлении необходимо кликнуть на строке с выбранным заявлением. В открывающейся строке доступна информация о маршруте, а также ссылке на доступные для скачивания документы в формате pdf (само заявление, расчёты плат, листы согласования и другие).

До выполнения расчёта платы можно удалить заявление, нажав на соответствующую кнопку в последнем столбце таблицы. Если расчёт платы произведён, то пользователь, оплатив данную сумму, может внести информацию об оплате на странице «Ввод оплаты» (рисунок 6.2). Туда можно попасть, нажав на соответствующую кнопку в строке заявления.

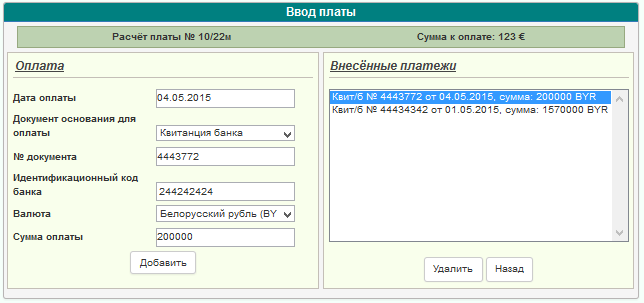


Рисунок 6.2 – Интерфейс страницы ввода платы

Для оформление нового заявления необходимо перейти на вкладку «Оформление заявления». На первом шаге пользователю необходимо указать тип разрешения, срок перевозки, количество рейсов (для обычного разрешения), количество месяцев (для многоразового разрешения), группу УБЭАД, полное имя и телефон ответственного за оформление. Для перехода к следующему шагу необходимо нажать кнопку «Далее».

На втором шаге подачи заявления необходимо предоставить информацию об автоперевозчике и транспортных средствах (рисунок 6.3). Автоперевозчиком может быть сам заявитель, подающий заявление, либо заявитель может быть представителем автоперевозчика. В последнем случае необходимо указать автоперевозчика, введя часть его имени или УНП в строку поиска и выбрав из выпадающего списка. При отсутствии результатов поиска можно вручную добавить запись об автоперевозчике, нажав на кнопку «Добавить нового перевозчика». После выбора перевозчика отобразится список его транспортных средств, которые были указаны ранее при подаче заявлений. Необходимо указать состав сцепки транспортных средств в таблице «Составное транспортное средство». Для этого можно выбрать транспортные средства из списка транспортных средств перевозчика, нажав на кнопку «Добавить» в последнем столбце таблицы. Также можно добавить транспортное средство вручную, нажав на соответствующую кнопку под таблицей. При добавлении и редактировании транспортных средств необходимо указать такую информацию, как тип, регистрационный номер, марка, модель, масса.

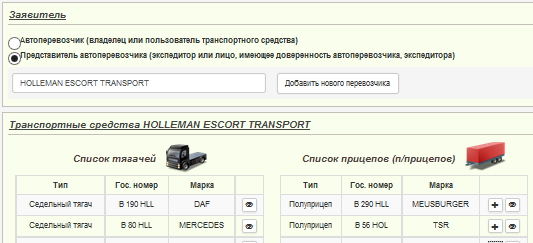




Рисунок 6.3 – Выбор автоперевозчика и транспортных средств

На очередном шаге необходимо указать наличие, вид и массу груза, заполнить схему сцепки транспортных средств, где указать нагрузки на оси, межосевые расстояния, типы осей, является ли ось ведущей и имеет ли пневмоподвеску (рисунок 6.4). При этом общая масса сцепки (с грузом) должна быть равна сумме нагрузок на оси, а сумма межосевых расстояний не должна превышать длину всей сцепки. Также на данной странице заполняются габаритные параметры в метрах: длина, высота, ширина, выступ груза за габариты по длине.

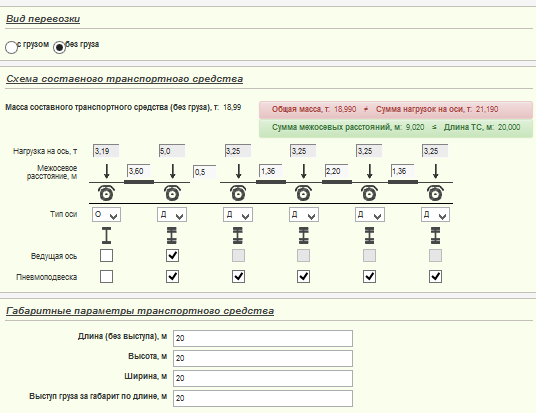


Рисунок 6.4 – Схема сцепки и габаритные параметры

На следующем этапе необходимо установить маршрут движения (рисунок 6.5). Он состоит из как минимум начальной и конечной точки и произвольного количества промежуточных точек.

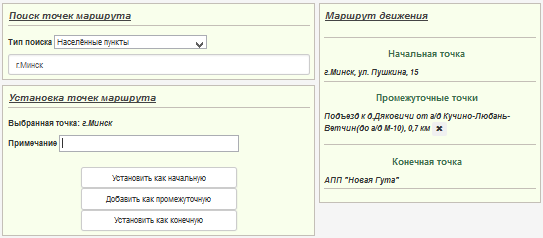


Рисунок 6.5 – Выбор маршрута

Для установки точки необходимо: указать тип поиска (населённый пункт, дорога и другие), ввести часть имени точки, выбрать точку из выпадающего списка, добавить при необходимости примечание, указать начальный километр (для дороги), нажать на кнопку, соответствующую типу точки.

Последний шаг оформления заявления – предварительный просмотр заявления перед отправкой (рисунок 6.6). Можно ввести дополнительную информацию в поле «Примечание». Для отправки заявления нужно нажать кнопку «Отправить заявление».

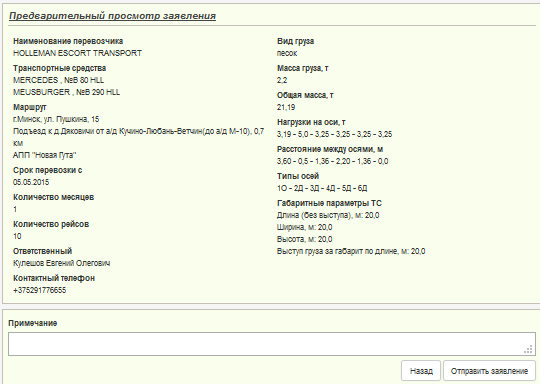


Рисунок 6.6 – Предварительный просмотр заявления

На вкладке «Архив разрешений» доступен список заявлений, по которым получены специальные разрешения (рисунок 6.7). Возможен выбор данных за определённый период, поиск заявления по номеру заявления, номеру специального разрешения или государственному номеру транспортного средства.

На вкладке «Неоформленные разрешения» располагается список заявлений, по которым отказано в выдаче специального разрешения. Возможен выбор данных за определённый период и поиск заявления.

Как и на вкладке «Список заявлений», на вкладках «Архив разрешений» и «Неоформленные разрешения» при нажатии на строку таблицы с интересуемой записью возможен просмотр дополнительной информации и ссылок на скачивание документов. Также можно воспользоваться имеющимся заявлением как шаблоном при оформлении нового заявления. Для этого необходимо нажать на кнопку «Копия заявления» в последнем столбце таблицы.

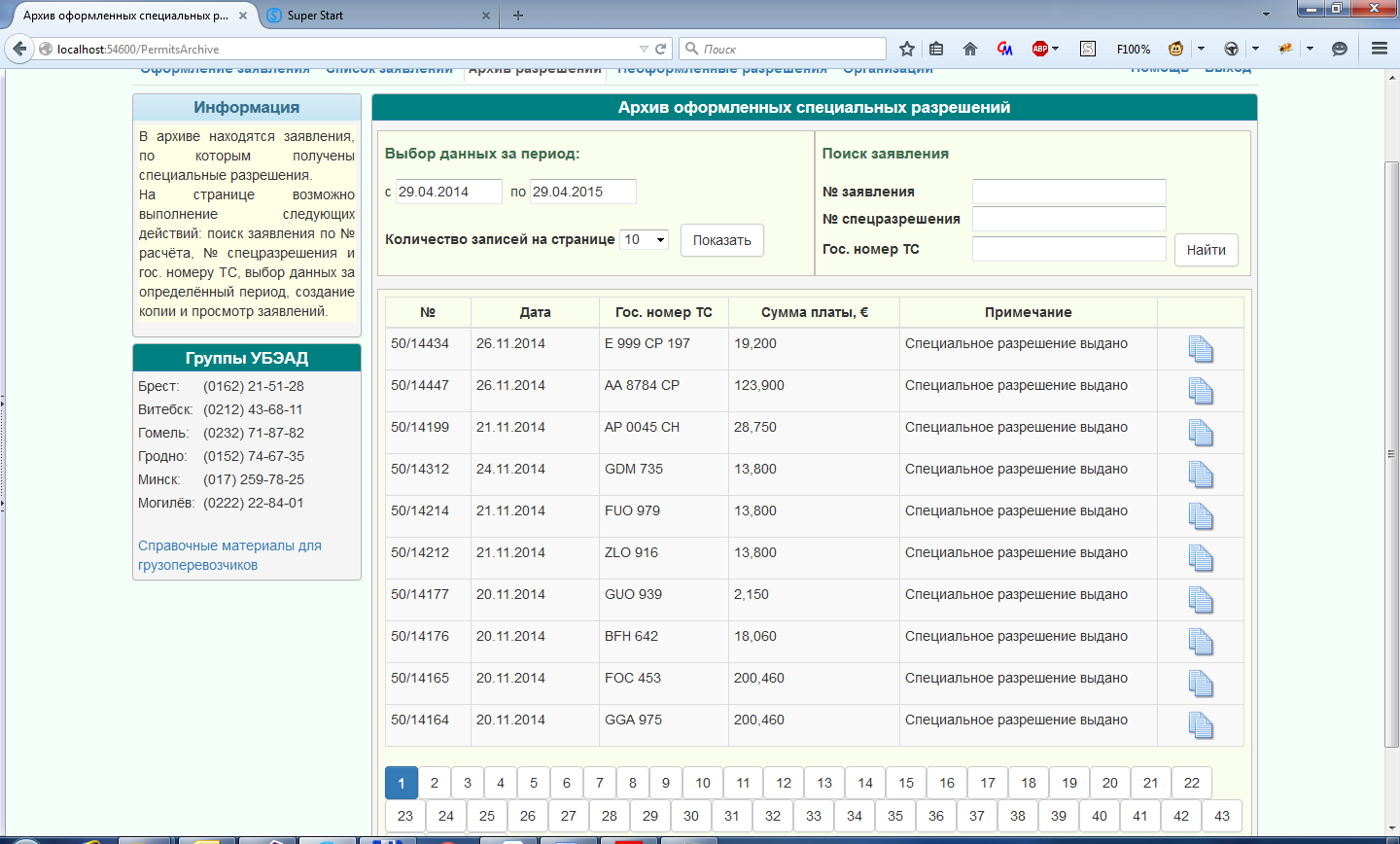


Рисунок 6.7 – Архив разрешений

На вкладке «Организации» (рисунок 6.8) можно увидеть список организаций, фигурировавших в качестве автоперевозчика в ранее поданных заявлениях. Можно просмотреть дополнительную информацию об организации или изменить данные организации-заявителя, нажав на соответствующие кнопки в последнем столбце таблицы.

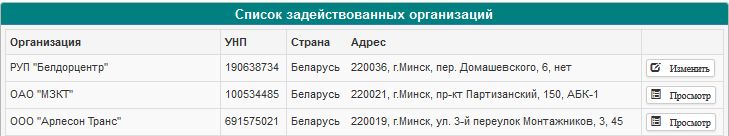


Рисунок 6.8 – Интерфейс вкладки «Организации»

1. ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОГО ПРОЕКТА
   1. Описание проекта

Web-приложение для подачи заявлений на выдачу специальных разрешений предназначено для предоставления автоперевозчикам возможности удалённо подать заявку на выдачу специального разрешения для проезда транспортного средства, максимальные весовые и (или) габаритные размеры которых превышают допустимые параметры, установленные для проезда по автомобильным дорогам общего пользования Республики Беларусь.

Основная цель данной системы – автоматизация процесса оформления специальных разрешений.

При подаче заявления производится автоматизированный процесс контроля поступающей информации, включая проверку соответствия нагрузок и габаритов транспортного средства установленным нормативам. Это экономит время и повышает производительность сотрудников, принимающих решение о выдаче специального разрешения.

Кроме того, данная система экономит время автоперевозчиков: в организацию, выдающую разрешения необходимо прийти только один раз – непосредственно за получением специального разрешения. В последующем можно воспользоваться поданным ранее заявлением как шаблоном. При оформлении новых заявлений для ранее использованных транспортных средств уже не надо повторно вводить данные о транспортном средстве – достаточно выбрать его из списка.

Таким образом, положительный экономический эффект у заказчика может быть получен вследствие:

* уменьшения бумажного документооборота (снижения затрат на расходные материалы);
* сокращения сроков оформления специальных разрешений;
* повышения эффективности работы сотрудников, принимающих решение о выдаче специального разрешения.
  1. Расчёт сметы затрат и цены ПО

Разрабатываемый программный продукт относится к третьей категории сложности, поскольку не относится к ПО, планирующемуся к использованию под нагрузкой, либо в различных наукоёмких или требующих оптимизации системах. Программный продукт является ПО общего назначения и относится к категории новизны В (Кн = 0,7).

При расчете сметы затрат будут использоваться данные, приведенные в таблице 7.1. Они отражают текущую финансовую ситуацию.

Таблица 7.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Буквенные обозначения | Единицы измерения | Количество |
| 1 | 2 | 3 | 4 |
| Группа сложности | - | - | 3 |
| Коэффициент новизны | Кн | - | 0,7 |
| Коэффициент, учитывающий использование стандартных модулей | Кт | - | 0,8 |
| Дополнительный коэффициент сложности | Ксд | - | 0,06 |
| Установленная плановая продолжительность разработки | Тр | лет |  |
| Количество дней в году | Дг | дней | 365 |
| Количество праздничных дней | Дп | дней | 6 |
| Количество выходных дней | Дв | дней | 104 |
| Количество дней отпуска | До | дней | 24 |
| Тарифная ставка 1-го разряда | Тм1 | руб. | 700 000 |
| Продолжительность рабочего дня | Тч | часов | 8 |
| Установленный фонд рабочего времени | Фрв | часов | 170 |
| Норматив дополнительной заработной платы | Нд | % | 20 |
| Ставка отчислений в ФСЗН | Нсз | % | 34 |
| Ставка отчислений на обязательное социальное страхование | Нсс | % | 0,6 |
| Норма расхода материалов от основной заработной платы | Нмз | % | 3 |
| Цена одного машинного часа | Цм | руб. | 4500 |
| Норматив расхода машинного времени | Нмв | ч. / 100 строк кода | 12 |
| Норматив расходов на командировки | Нрнк | % | 15 |
| Норматив прочих затрат | Нпз | % | 20 |
| Норматив накладных расходов | Нрн | % | 50 |
| Уровень рентабельности | Урп | % | 20 |
| Ставка налога на добавленную стоимость | Ндс | % | 20 |
| Норматив расходов на освоение | Но | % | 10 |
| Норматив расходов на сопровождение | Нс | % | 20 |
| Ставка налога на прибыль | Нп | % | 18 |

Отправной точкой для расчёта плановой сметы затрат на разработку ПО, требуется определить общий объем программного продукта (*V*о). В качестве единицы измерения примем количество строк исходного кода (Lines of Code, LOC). Прогнозируемый общий объём ПО определяется по каталогу функций. Каталог функций данного программного продукта представлен в таблице 7.2.

Таблица 7.2 – Каталог функций ПО

|  |  |  |
| --- | --- | --- |
| Код функции | Наименование (содержание) функции | Объем функций (LOC) |
| 101 | Организация ввода информации | 150 |
| 102 | Контроль, предварительная обработка информации | 450 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 320 |
| 111 | Управление вводом/выводом | 2400 |
| 208 | Организация поиска и поиск в базе данных | 5480 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 |
| 507 | Обеспечение интерфейса между компонентами | 970 |
| 707 | Графический вывод результатов | 480 |
| – | Общий объём (*VО*) | 10660 |

На основе общего объёма и категории сложности ПО определяется нормативная трудоёмкость, которая, в данном случае, для *V*о = 10660 и третьей категории сложности, составит Тн = 228 человеко-дней.

Наличие интерактивного интерфейса позволяет применить к объёму ПО коэффициент Кс, который определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.1) |

где К*i* – коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

*n* – количество учитываемых характеристик.

|  |  |  |
| --- | --- | --- |
|  | . |  |

Исходя из нормативной трудоёмкости можно определить общую трудоёмкость То по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.2) |

где Кс – дополнительный коэффициент сложности;

Кт – коэффициент использования типовых программ и модулей;

Кн – коэффициент новизны.

Подставив значения в формулу (7.2), получим:

|  |  |  |
| --- | --- | --- |
|  | . |  |

Имея общую трудоёмкость, определяется численность исполнителей проекта, либо срок его разработки. Данный проект делался на заказ, при этом заранее было определено, что работа будет выполнена одним человеком, таким образом требуется определить второй параметр.

Для определения срока разработки проекта, необходимо рассчитать эффективный фонд времени одного работника (Фэф):

|  |  |  |
| --- | --- | --- |
|  |  | (7.3) |

где Дг – количество дней в году;

Дп – количество праздничных дней в году;

Дв ­– количество выходных дней в году;

До – количество дней отпуска.

Таким образом, по формуле (7.3) фонд эффективного времени составит:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Срок разработки проекта (Тр) определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Чр – численность исполнителей проекта;

То – общая трудоемкость разработки проекта, человеко-дней;

Фэф – эффективный фонд времени работы одного работника.

Подставив значения в формулу (7.4), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Основой для расчёта сметы затрат является основная заработная плата разработчиков проекта. В данном случае имеется один работник – инженер-программист II-й категории (тарифный разряд – 12, тарифный коэффициент – 2,84). Месячная тарифная ставка исполнителя (Тм) определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.5) |

где Тм1 – месячная тарифная ставка первого разряда, руб.;

Тк – тарифный коэффициент.

Месячная тарифная ставка, определённая по формуле (7.5) составит:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Исходя из месячной тарифной ставки рассчитывается часовая тарифная ставка (Тч):

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

где Фр – среднемесячная норма рабочего времени, ч.

При подстановке значений в формулу (7.6), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Основная заработная плата исполнителей рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.7) |

где Тч*i* – часовая тарифная ставка *i*-го исполнителя, руб.;

Тч – количество часов работы в день, ч;

Фп – плановый фонд рабочего времени *i*-го исполнителя;

К – коэффициент премирования, принятый равным 1,4.

Учитывая число разработчиков *n* = 1, определим основную заработную плату по формуле (7.7):

|  |  |  |
| --- | --- | --- |
|  |  |  |

Дополнительная заработная плата (Зд*i*) включает в себя оплаты отпусков и другие выплаты, предусмотренные законодательством, и определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.8) |

где Нд – норматив дополнительной заработной платы (15%).

Тогда получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Отчисления в фонды социальной защиты и социального страхования определяются по следующим формулам:

|  |  |  |
| --- | --- | --- |
|  |  | (7.9) |
|  |  | (7.10) |

где Нсз – норматив отчислений в фонд социальной защиты населения (34%);

Нсс – норматив отчислений в фонд социального страхования (0,6%).

По формулам (7.9) и (7.10) получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Расходы по статье «Материалы» отражают расходы на бумагу, тонер и прочие вещи, необходимые для разработки ПО. Сумма затрат на расходные материалы определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.11) |

где Нмз ­– норма расхода материалов от основной заработной платы (3%).

Подставив значения в формулу (7.11), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Расходы по статье «Машинное время» включает оплату машинного времени, необходимого для разработки и отладки ПО и определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.12) |

где Цм*i* – цена одного машино-часа, руб.;

*V*о*i* – общий объем ПО (строк исходного кода);

Нмв – норматив расхода машинного времени на отладку 100 строк исходного кода, машино-часов.

В современных условиях разработки используется понижающий коэффициент 0,4. Подставляя значения в формулу (7.12), с учётом понижающего коэффициента получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Расходы по статье «Научные командировки» определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.13) |

где Нрнк – норматив расходов на командировки по организации (15%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.14) |

где Нпз – норматив прочих затрат в целом по организации (20%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Затраты по статье «Накладные расходы» связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.15) |

где Нрн – норматив накладных расходов в целом по организации.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Общая сумма расходов по смете (Сп*i*) определяется как сумма выше рассчитанных показателей:

|  |  |  |
| --- | --- | --- |
|  |  | (7.16) |

Подставив рассчитанные ранее значения в формулу (7.16), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Прогнозируемая прибыль от создаваемого ПО определяется как:

|  |  |  |
| --- | --- | --- |
|  |  | (7.17) |

где Сп*i* – себестоимость ПО, руб.;

Урп*i* – уровень рентабельности ПО (20%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

На основе прогнозируемой прибыли определяется прогнозируемая цена ПО без налогов:

|  |  |  |
| --- | --- | --- |
|  |  | (7.18) |

Подставляя значения в формулу (7.18), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

При расчёте отпускной цены дополнительно учитывается налог на добавочную стоимость (НДС):

|  |  |  |
| --- | --- | --- |
|  |  | (7.19) |

где Ндс – норматив налога на добавленную стоимость (20%).

По формуле (7.19) налог на добавочную стоимость равен:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Таким образом, с учётом НДС отпускная цена рассчитывается как:

|  |  |  |
| --- | --- | --- |
|  |  | (7.20) |

Подставив значения в формулу (7.20), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

В дополнение к выше рассчитанным параметрам, определяются расходы на освоение (Ро*i*) и сопровождение (Рс*i*) ПО:

|  |  |  |
| --- | --- | --- |
|  |  | (7.21) |
|  |  | (7.22) |

где Но – норматив расходов на освоение ПО (10%);

Нс – норматив расходов на сопровождение ПО (20%).

Используя формулы (7.21) и (7.22), определим значения расходов:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Все выше рассчитанные параметры, а также результирующие показатели сведены в таблицу 7.3.

Таблица 7.3 – Расчёт себестоимости и отпускной цены

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статьи | Норматив | Формула расчёта | Значение, руб. |
| 1 | 2 | 3 | 4 |
| Основная ЗП | – |  | 30 255 021 |
| Дополнительная ЗП | Нд = 15% |  | 4 538 253 |
| Отчисления в фонд соцзащиты | Нсз = 34% |  | 11 829 713 |
| Отчисления в фонд соцстраха | Нсс = 0,6% |  | 208 760 |
| Материалы и комплектующие | Нмз = 3% |  | 907 651 |

*Продолжение таблицы 7.3*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Машинное время | Нмв = 12 ч |  | 2 302 560 |
| Научные командировки | Ннк = 15% |  | 4 538 253 |
| Прочие затраты | Нпз = 20% |  | 6 051 004 |
| Накладные расходы | Нрн = 50% |  | 15 127 511 |
| Общая сумма по смете | – |  | 75 758 726 |
| Прогнозируемая прибыль | Урп = 20% |  | 15 151 745 |
| Прогнозируемая цена без налогов | – |  | 90 910 471 |
| НДС | Ндс = 20% |  | 18 182 094 |
| Отпускная цена | – |  | 109 092 565 |
| Освоение ПО | Но = 10% |  | 7 575 873 |
| Сопровождение ПО | Нс = 20% |  | 15 151 745 |

Учитывая налог на прибыль, можно рассчитать итоговую сумму, которая останется разработчику и будет является его экономическим эффектом:

|  |  |  |
| --- | --- | --- |
|  |  | (7.23) |

где ΔПч – чистая прибыль;

Ппс – прогнозируемая прибыль;

Нп – норматив налога на прибыль (18%).

Подставив значения в формулу (7.23), определим чистую прибыль:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Чистая прибыль от реализации ПО (ΔПч = 12 424 431 рублей) остается организации-разработчику и представляет собой экономический эффект от создания нового программного средства.

* 1. Расчёт экономического эффекта от применения программного средства у заказчика

Для определения экономического эффекта от использования нового ПО у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПО (расходы на заработную плату с начислениями, материалы, машинное время) с расходами по соответствующим статьям при использовании прежнего варианта ПО. При сравнении базового и нового вариантов ПО в качестве экономического эффекта будет выступать общая экономия всех видов ресурсов относительно базового варианта. При этом создание нового ПО окажется экономически целесообразным лишь в том случае, если все капитальные затраты окупятся за счет получаемой экономии в ближайшие 2–3 года. Исходные данные для расчёта экономического эффекта показаны в таблице 7.4.

Таблица 7.4 – Исходные данные для определения экономического эффекта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование показателей | Обо­значе­ние | Единицы из­мерения | Значение показателя | |
| в базовом варианте | в новом варианте |
| 1 | 2 | 3 | 4 | 5 |
| Капитальные вложения, включая затраты поль­зователя на приобрете­ние ПO | Кпр | руб. | - | 109 092 565 |
| Затраты на освоение ПО | Кос | руб. | - | 7 575 873 |
| Затраты на сопровожде­ние ПО | Кс | руб. | - | 15 151 745 |
| Затраты на укомплекто­вание ВТ техническими средствами в связи с внедрением нового ПО | Ктс | руб. | - | 7 575 873 |
| Затраты на пополнение оборотных средств в связи с эксплуатацией нового ПО | Коб | руб. | - | 2 500 000 |
| Среднемесячная ЗП од­ного программиста | Зсм | руб. | 2 800 000 | 2 800 000 |
| Коэффициент начисле­ний на зарплату | Кнз |  | 1,5 | 1,5 |
| Среднемесячное коли­чество рабочих дней | Др | день | 21,5 | 21,5 |

*Продолжение таблицы 7.4*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Количество типовых за­дач, решаемых за год | Зт1, Зт2 | задача | 1 300 | 1 300 |
| Объем выполняемых работ за год | А1, А2 | задача | 1 300 | 1 300 |
| Средняя трудоемкость работ | Тс1, Тс2 | чел.-час на задачу | 4 | 0,5 |
| Средний расход машин­ного времени | Мв1, Мв2 | маш.-час на задачу | 4 | 0,5 |
| Цена 1-го машино-часа работы ЭВМ | Цм | руб. | 4 500 | 4 500 |
| Количество часов ра­боты в день | Тч | ч | 8 | 8 |
| Ставка налога на при­быль | Нп | % | 18 | 18 |

Общие капитальные вложения заказчика (потребителя) рассчитываются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.24) |

где Кпр – затраты пользователя на приобретение ПО по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации, руб.;

Кос – затраты пользователя на освоение ПС, руб.;

Кс – затраты пользователя на оплату услуг по сопровождению ПО, руб.;

Ктс – затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПО, руб.;

Коб – затраты на пополнение оборотных средств в связи с использованием нового ПО, руб.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия затрат на заработную плату (Сз) при использовании нового ПО в расчете на объем выполненных работ определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.25) |

где Сзе – экономия затрат на заработную плату при решении задач c использованием нового ПО на 1 задачу, руб.;

А2 – объем выполненных работ с использованием нового ПО (задач).

Экономия затрат на заработную плату в расчете на 1 задачу рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.26) |

где Зсм – среднемесячная заработная плата одного программиста, руб.;

Тс1, Тс2 – трудоемкости работ в расчете на 1 задачу, человеко-часов;

Тч – количество часов работы в день, ч;

Др – среднемесячное количество рабочих дней.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Тогда, подставляя значения в формулу (7.25), рассчитаем экономию затрат на заработную плату:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия с учетом начисления на зарплату вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.27) |

где Кнз – коэффициент начислений на зарплату (1,5).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия затрат на оплату машинного времени (См) в расчете на выполненный объем работ в результате применения нового ПО:

|  |  |  |
| --- | --- | --- |
|  |  | (7.28) |

где Сме – экономия затрат на оплату машинного времени в расчёте на 1 задачу с использованием нового ПО.

Экономия затрат на оплату машинного времени в расчете на 1 задачу определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.29) |

где Цм – цена одного машино-часа работы ЭВМ;

Мв1, Мв2 – средний расход машинного времени при применении соответственно базового и нового ПО.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Таким образом, по формуле (7.28) определим экономию затрат на оплату машинного времени в расчете на выполненный объем работ:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Общая годовая экономия текущих затрат, связанных с использованием нового ПО является важным фактором, влияющим в дальнейшем на расчеты, производимые в данном разделе и вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.30) |

Подставляя ранее полученные значения в формулу (7.30), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Внедрение нового ПО позволит заказчику сэкономить на текущих затратах 131 579 651 рубля, то есть практически получить на эту сумму дополнительную прибыль. Для заказчика в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении, которая определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.31) |

где Нп – ставка налога на прибыль (18%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2015 год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α*t*, который рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.32) |

где *E* – норматив приведения разновременных затрат и результатов (c учётом безрисковой ставки процента по валютным депозитам, уровня инфляции, роста спроса и стабильности дохода примем *E* = 24%);

*t* – номер года, результаты и затраты которого приводятся к расчётному (2015 год – 1, 2016 год – 2, 2017 год – 3, 2018 год – 4);

*tp* – номер расчётного года (2015).

Таким образом, получим следующие значения коэффициентов дисконтирования:

2015 год: 

2016 год: 

2017 год: 

2018 год: 

Сведем данные расчета экономического эффекта в таблицу 7.5.

Таблица 7.5 – Расчет экономического эффекта от использования нового ПО

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2015 | 2016 | 2017 | 2018 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| *Результаты:* | | | | | |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 107 895 314 | 107 895 314 | 107 895 314 |
| То же с учетом фактора времени | руб. |  | 87 012 350 | 70 171 250 | 56 589 718 |
| *Затраты:* | | | | | |
| Приобретение ПО (Кпр) | руб. | 109 092 565 |  |  |  |
| Освоение ПО (Кос) | руб. | 7 575 873 |  |  |  |

*Продолжение таблицы 7.5*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Сопровождение (Кс) | руб. | 15 151 745 |  |  |  |
| Доукомплектование ВТ техническими средствами (Ктс) | руб. | 7 575 873 |  |  |  |
| Пополнение оборотных средств (Коб) | руб. | 2 500 000 |  |  |  |
| Всего затрат | руб. | 141 896 056 |  |  |  |
| То же с учетом фактора времени | руб. | 141 896 056 |  |  |  |
| *Экономический эффект:* | | | | | |
| Превышение результатов над затратами | руб. | -141 896 056 | 87 012 350 | 70 171 250 | 56 589 718 |
| То же нарастающим итогом | руб. | -141 896 056 | -54 883 706 | 15 287 544 | 71 877 262 |
| Коэффициент приведения | ед. | 1 | 0,8065 | 0,6504 | 0,5245 |

В данном разделе была рассчитана себестоимость проекта, на основании которой была сформирована рыночная цена. Был рассчитан экономический эффект для разработчика: чистая прибыль от реализации ПО составила 12 424 431 рублей.

Кроме этого, была рассчитана экономическая эффективность для заказчика, выраженная в чистом дисконтированном доходе, составившем 71 877 262 рублей. Стоит отметить, что затраты заказчика окупятся менее, чем за два года. Данный эффект получается в результате снижения трудоемкости решения задач, снижения затрат машинного времени, снижения затрат на бумажный документооборот. Кроме этого, дополнительный положительный эффект может создаться за счёт пересмотра кадрового состава на основе анализа производимой конкретными сотрудниками работы, учитывая новые условия труда.

Таким образом, учитывая снижение трудоёмкости, значительную экономию затрат, достаточно быстрый срок окупаемости, можно сделать вывод, что проект является полезным и экономически эффективным для заказчика.

1. ПРОФИЛАКТИКА ПЕРЕУТОМЛЕНИЯ РАБОТНИКОВ, ЗАНЯТЫХ РЕШЕНИЕМ ЗАДАЧ ПО СОСТАВЛЕНИЮ ПРОГРАММ

Цель данного дипломного проекта – разработка веб-приложения для подачи заявлений на выдачу специальных разрешений для проезда крупногабаритного транспорта. Разработка данного приложения является достаточно трудоёмким процессом, требующим повышенного напряжения внимания, сильной зрительной активности, что может привести к переутомлению и стать препятствием в работе.

В первой части данного раздела рассмотрены общее понятие работоспособности и утомления, а также определяющие их факторы для работников умственного труда, к которым относятся и программисты. Во второй части рассмотрены основные требования к работе за персональным компьютером (в том числе применительно к разработке данного проекта), выполнение которых предотвращает развитие утомления, приведены упражнения для сохранения высокой работоспособности.

* 1. Понятие работоспособности и утомления, определяющие их факторы для работников умственного труда

Умственный труд – труд, который предполагает восприятие и переработку большого количества информации с включением памяти, внимания, активизацией процесса мышления. К работникам умственного труда относят инженеров, экономистов, бухгалтеров, программистов, научных работников, конструкторов и других.

Работоспособность человека определяется его стойкостью к различным видам утомления – физическому, умственному и другим и характеризуется продолжительностью качественного выполнения соответствующей работы.

Умственная работоспособность зависит от возраста, состояния здоровья, моральных и материальных стимулов, кроме того в значительной мере от психофизиологических качеств человека. К их числу следует отнести общую выносливость, в том числе и физическую, быстроту мыслительной деятельности, способность к переключению и распределению, концентрации и устойчивость внимания, эмоциональную устойчивость [13].

Умственная работоспособность не постоянна, она изменяется на протяжении рабочего дня. В начале она низкая (период врабатывания), затем поднимается и какое-то время удерживается на высоком уровне (период устойчивой работоспособности), после чего снижается (период некомпенсированного утомления) [13].

Умственная работоспособность человека в значительной мере зависит от времени суток. Суточный физиологический ритм функций систем организма определяет повышенную интенсивность деятельности органов и систем в дневное время и пониженную – в ночное время.

Умственная работоспособность изменяется и в течении недели. На понедельник приходится стадия врабатывания, на вторник, среду и четверг – высокая работоспособность, а развивающееся утомление приходится на пятницу и субботу. Именно поэтому в воскресенье следует больше внимания уделять физической подготовке и занятиям спортом. Они снижают утомление [13].

Утомление – временное снижение работоспособности под влиянием длительного воздействия нагрузки. Утомление по своей биологической сущности нормальный физиологический процесс, который выполняет определённую защитную функцию в организме, предохраняя его от перенапряжения и возможного в связи с этим повреждения. Если человек возобновляет работу на фоне медленно развивающегося утомления, то это приводит к переутомлению, то есть к хроническому утомлению, которое не ликвидируется за обычный период отдыха [14].

Возникает утомление вследствие истощения внутренних ресурсов индивида и рассогласования в работе обеспечивающих деятельность систем.

Утомление имеет разнообразные проявления на поведенческом (снижение производительности труда, уменьшение скорости и точности работы), физиологическом (затруднение выработки условных связей, повышение инерционности в динамике нервных процессов), психологическом (снижение чувствительности, нарушение внимания, памяти, интеллектуальных процессов, сдвиги в эмоционально-мотивационной сфере) уровнях. Сопровождается формированием комплекса субъективных переживаний усталости. Специфика проявлений усталости зависит от вида нагрузки, локализации ее воздействия, времени, необходимого для восстановления оптимального уровня работоспособности [14].

В зависимости от вида выполняемой работы выделяют умственное и физическое утомление, при котором учитывают отклонения энергетических показателей обмена, например, изменение температуры тела, биоэлектрических потенциалов.

В связи с тем, что обнаружилась принципиальная общность физического и умственного утомления, приобретает большое распространение классификация, основанная на преимущественной локализации утомление в звеньях нервной системы, обеспечивающей деятельность человека. Так, различают сенсорное утомление, и его разновидности (перцептивное и информационное) и эффекторное утомление. Кроме того, выделяют как обобщённую форму общее утомление. Однако та или иная классификация зависит от принятой физиологической теории утомления [14].

Сенсорное утомление развивается в результате длительного или интенсивного воздействия раздражителя (например, сильный шум, свет), при котором первичные изменения возникают в сенсорных системах, начиная от рецептора и кончая корковым концом анализатора. Перцептивное утомление, локализованное преимущественно в корковом конце анализатора, связано с трудностью обнаружения сигнала (например, при больших помехах, при его малой интенсивности, трудности дифференцирования). Информационное утомление развивается вследствие недостаточности информации или при информационной перегрузке. Эффекторное утомление возникает при локализации изменений преимущественно в отделах центральной нервной системы, формирующих двигательный акт [14].

Выраженное утомление отрицательно влияет на организм, уменьшая производительность труда, и может привести сразу к предпатологической фазе срыва, а при нерациональном отдыхе и к развитию патологического состояния переутомления. Чрезмерное утомление может сопровождаться неврозами и сосудистыми заболеваниями [15].

* 1. Основные пути снижения утомления программистов

Борьба с утомлением осуществляется внедрением рациональных режимов труда и отдыха, улучшением условий труда, внедрением эргономических рекомендаций по организации рабочего места, рациональным распределением функций между человеком и машиной.

Для того чтобы организовать комфортную среду при работе с персональным компьютером (ПК), необходимо изучить требования к ней, регламентированные соответствующими нормативно-техническими документами, и возможные средства и способы защиты от неблагоприятных факторов в случае превышения в реальности допустимых величин.

Существуют санитарные нормы и правила "Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами" (СанПиН от 28.06.2013 № 59) [16]. Они устанавливают требования к:

* видеодисплейным терминалам (ВДТ), электронно-вычислительным машинам (ЭВМ), персональным электронно-вычислительным машинам (ПЭВМ), в том числе к портативным (нетбуки, ноутбуки и другое), и периферийным устройствам;
* помещениям для работы с ВДТ, ЭВМ и ПЭВМ;
* микроклимату, содержанию аэроионов и вредных химических веществ в воздухе на рабочих местах, оборудованных ВДТ, ЭВМ и ПЭВМ;
* освещению на рабочих местах, оборудованных ВДТ, ЭВМ и ПЭВМ;
* организации и оборудованию рабочих мест с ВДТ, ЭВМ и ПЭВМ;
* организации режима труда и отдыха при работе с ВДТ, ЭВМ и ПЭВМ.

Одним из важных факторов, которые влияют на работоспособность и состояние здоровья пользователей ПК является организация рабочего места. Неправильная организация рабочего места приводит к общей усталости, головным болям, усталости мышц рук, болям в спине и шее.

Такие негативные моменты чаще всего возникают из-за несоответствия помещений и организации рабочих мест санитарно-производственным нормам.

Площадь одного рабочего места для пользователей ВДТ, ЭВМ и ПЭВМ на базе электронно-лучевой трубки должна составлять не менее 6 м2, на базе плоских дискретных экранов (жидкокристаллические, плазменные и другое – не менее 4,5 м2. Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы. Конструкция рабочего стула (кресла) должна обеспечивать поддержание рациональной рабочей позы, позволять изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления. Рабочий стул (кресло) должен быть подъемно-поворотным, регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию. Высота рабочей поверхности стола для взрослых пользователей должна регулироваться в пределах 680-800 мм; при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм. Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм [16].

Экран видеомонитора должен находиться на расстоянии 600-700 мм от глаз пользователя, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов. Клавиатуру следует располагать на поверхности стола на расстоянии 100-300 мм от края, обращенного к взрослому пользователю, или на специальной, регулируемой по высоте рабочей поверхности, отделенной от основной столешницы [16].

Освещенность также влияет на состояние здоровья и работоспособность программистов. В данном случае пользователь будет работать за дисплеем, а особенностью такой работы является постоянное и значительное напряжение функций зрительного анализатора, обусловленного необходимостью различения самосветящихся объектов (символов, знаков) при наличии бликов на экране, строчной структурой экрана, мельканием изображения, не достаточной четкостью объектов различения. Для того чтобы избежать перенапряжения и болей в глазах при работе с ПК установлены специальные требования к освещению на рабочих местах, оснащённых ПЭВМ.

Рабочие места по отношению к световым проемам должны располагаться не ближе 3 м так, чтобы естественный свет падал сбоку, преимущественно слева. Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300-500 люкс. Освещение не должно создавать бликов на поверхности экрана. Освещенность поверхности экрана не должна быть более 300 люкс [16].

Необходимо ограничивать прямую блесткость от источников освещения, при этом яркость светящихся поверхностей (окна, светильники и другое), находящихся в поле зрения, должна быть не более 200 кд/м2. Необходимо ограничивать отраженную блесткость на рабочих поверхностях (экран, стол, клавиатура и другое) за счет правильного выбора типов светильников и расположения рабочих мест по отношению к источникам естественного и искусственного освещения, при этом яркость бликов на экране ВДТ, ЭВМ и ПЭВМ не должна превышать 40 кд/м2 и яркость потолка не должна превышать 200 кд/м2. Следует ограничивать неравномерность распределения яркости в поле зрения пользователя ВДТ, ЭВМ и ПЭВМ, при этом соотношение яркости между рабочими поверхностями не должно превышать 3:1 – 5:1, а между рабочими поверхностями и поверхностями стен и оборудования – 10:1 [16].

Шум неблагоприятен для человека, особенно при длительном воздействии. У программистов это выражается в снижении работоспособности (например, скорость обработки текста уменьшается на 10 – 15%), в ускорении развития зрительного утомления, изменении цветоощущения, повышении расхода энергии (на 17%). Продолжительный и интенсивный шум значительно снижает производительность труда и приводит к росту количества ошибок в работе. В научно-техническом центре основными источниками шума выступают телефонные звонки и разговоры, кулеры в системных блоках ПК, системы кондиционирования и вентилирования воздуха, внешние источники шума [17].

Разработка данного дипломного проекта проводилась в помещении, в котором для работы используются преимущественно ПЭВМ. Такие помещения не должны граничить с помещениями, в которых уровни шума и вибрации превышают нормируемые значения для рабочих мест программистов [16]. В соответствии с Санитарными нормами и правилами, устанавливающими ПДУ шума на рабочих местах, в помещениях жилых и общественных зданий [18], для рабочих мест программистов вычислительных машин уровни звука и эквивалентные по энергии уровни звука непостоянного шума не должны превышать 50 дБА.

При работе с ПЭВМ очень важную роль играет соблюдение правильного режима труда и отдыха. Суммарное время регламентированных перерывов устанавливается в зависимости от категории работы за ПЭВМ. Работу по разработке веб-приложения данного дипломного проекта можно отнести ко II категории (для уровня нагрузки до 4 часов при 8-часовой смене). Суммарное время перерывов при этом должно составлять не менее 50 минут [16].

Эффективность перерывов повышается при сочетании с производственной гимнастикой или организации специального помещения для отдыха персонала с удобной мягкой мебелью, аквариумом, зеленой зоной.

Для снижения утомления при работе за ПЭВМ рекомендуется выполнять упражнения производственной гимнастики. В положении сидя могут быть выполнены следующие упражнения [19]:

1. Приподнять ступни ног над полом, напрягая мышцы. Выполнять движения ступнями в различных направлениях.
2. Прижать пятки к ножкам стула и напрягать мышцы.
3. Выпрямлять ноги в коленях и напрячь мышцы ног. Если условия не позволяют выпрямлять ноги, обхватить носками ножки стула и напрягать мышцы.
4. Напрягая мышцы плечевого пояса, выполнять движения плечами в различных направлениях.
5. Напрягая мышцы спины, прогибаться.
6. Напрягая мышцы шеи, выполнять движения головой.

Комплекс упражнений, снимающих утомление органов зрения [19]:

1. Сидя, крепко зажмурить глаза – 3-5 секунд, широко раскрыть глаза – 3-5 секунд. Повторить 5-6 раз.
2. Сидя, быстрые моргания в течение одной минуты.
3. Сидя, посмотреть на кончик носа – 3-4 секунд, перевести взгляд вдаль – 3-5 секунд. Повторить 4-5 раз.
4. Сидя, опустить веки и массировать их круговыми движениями пальцев в течение одной минуты (проводить массаж без болезненных ощущений).
5. Сидя, закрыть глаза, расслабиться, расслабить мышцы лица.

В данном разделе дипломного проекта были определены факторы, определяющие работоспособность и утомление, изложены требования к рабочему месту работников, занятых решением задач по составлению программ. Созданные условия должны обеспечивать комфортную работу.

На основании изученной литературы по данной проблеме, были указаны оптимальные размеры рабочего стола и кресла, рабочей поверхности, а также проведен выбор системы оптимального освещения производственного помещения и уровня шума на рабочем месте.

Соблюдение условий, определяющих оптимальную организацию рабочего места программиста, проведение перерывов и выполнение производственной гимнастики позволят сохранить хорошую работоспособность в течение всего рабочего дня, повысят как в количественном, так и в качественном отношениях производительность труда, что в свою очередь будет способствовать быстрейшей разработке и отладке программного продукта.

ЗАКЛЮЧЕНИЕ

Ограничения на проезд тяжеловесных и крупногабаритных транспортных средств по автомобильным дорогам являются необходимыми для сохранения дорог и дорожных сооружений, а также для обеспечения безопасности движения. Проезд таких транспортных средств возможен только при наличии специального разрешения.

Как уполномоченные органы по выдаче специальных разрешений, так и автоперевозчики заинтересованы в том, чтобы оформление специального разрешения происходило как можно быстрее и с минимальными издержками. Разработанный проект во многом способствует этому, предоставляя автоперевозчику возможность быстро и дистанционно через Интернет подать заявление на выдачу специального разрешения. При этом процесс подачи заявления организован таким образом, чтобы автоперевозчик не мог ввести недопустимые данные или не указать обязательные сведения, что предотвращает рассмотрение невостребованных заявлений и экономит время. Также автоперевозчику доступен список поданных заявлений с информацией о текущей стадии рассмотрения заявления, возможен просмотр списка выданных разрешений и списка невостребованных заявлений с причинами отказа в выдаче специального разрешения.

В качестве усовершенствования программы в дальнейшем планируется:

* организация обратной связи между заявителем и областными группами управления безопасной эксплуатации автомобильных дорог с целью обмена дополнительной информацией и корректировки заявлений;
* реализация оповещения пользователя по электронной почте о смене статуса заявления.

СПИСОК ЛИТЕРАТУРЫ

1. Портал государственных услуг Российской Федерации [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.gosuslugi.ru/pgu/service/10000012382\_29.html#!\_description.
2. Электронное лицензирование республика Казахстан [Электронный ресурс]. – Электронные данные. – Режим доступа: http://goo.gl/RAlRnU.
3. TxPROS Permitting System [Электронный ресурс]. – Электронные данные. – Режим доступа: https://txpros.txdmv.gov/.
4. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер. – СПб. : Питер, 2013. – 896 с.
5. Библиотека MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/default.aspx.
6. C# Version 5.0 Specification [Электронный ресурс] : Specification / Microsoft Corporation. – Режим доступа: CSharp Language Specification.docx.
7. Фримен, А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов / А. Фримен. – М. : Вильямс. – 736 с.
8. Руководство по ASP.NET MVC 5 [Электронный ресурс]. – Электронные данные. – Режим доступа: http://metanit.com/sharp/mvc5/.
9. Общие сведения о ASP.NET MVC [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/dd381412%28v=vs.108%29.aspx.
10. Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/NHibernate.
11. NHibernate Reference Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: http://nhibernate.info/doc/nh/en/.
12. Schenker, G. Nhibernate 3 Beginner's Guide / G. Schenker. – Packt Publishing, 2011. – 368 с.
13. Чекалин, Н. А. Охрана труда в электротехнической промышленности / Н. А. Чекалин, Г. Н. Полухина, Г. Тугуши. – М. : Энергоатомиздат, 1984. – 272 с.
14. Розенблат, В. В. Проблема утомления / В. В. Розенблат. – М. : Наука, 1975. – 220 с.
15. Виноградов, М. И. Физиология трудовых процессов / Виноградов М. И. – Л. : Изд-во Ленинград. гос. ун-та, 1958. – 461 с.
16. СанПиН от 28.06.2013 № 59 «Требования при работе с видеодисплейными терминалами и электронно-вычислительными машинами» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://tnpa.by/KartochkaDoc.php?UrlRN=300621.
17. Семич, В. П. Охрана труда при работе на персональных электронно-вычислительных машинах и другой офисной технике : практ. пособие / В. П. Семич, А. В. Семич. – Минск : ЦОТЖ, 2001. – 75 с.
18. СанПиН от 16.11.2011 № 115 «Шум на рабочих местах, в транспортных средствах, в помещениях жилых, общественных зданий и на территории жилой застройки» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://tnpa.by/KartochkaDoc.php?UrlRN=273413.
19. Григорович, Е. С. Производственная гимнастика для работников основных групп умственного труда : метод. реком. / Е. С. Григорович, А. М. Трофименко, И. Н. Малуха. – Минск : МГМИ, 2000. – 39 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный текст класса MakeRequestController

[AuthorizeBdcApp]

[Culture]

public class MakeRequestController : Controller

{

private readonly DataProvider \_dataProvider

= new DataProvider();

private readonly RequestCookieManager \_cookieManager

= new RequestCookieManager(System.Web.HttpContext.Current);

public ActionResult CopyRequest(int id)

{

var requestModel = \_dataProvider.GetRequestModel(id);

if (requestModel != null)

{

\_cookieManager.SaveObject(RequestStep.RequestInfo,

requestModel.RequestInfo);

\_cookieManager.SaveObject(RequestStep.VehicleDetails,

requestModel.VehicleDetails);

\_cookieManager.SaveObject(RequestStep.VehicleScheme,

requestModel.VehicleScheme);

\_cookieManager.SaveObject(RequestStep.Route,

requestModel.Route);

\_cookieManager.SaveObject(RequestStep.RequestPreview,

requestModel.Note);

}

return RedirectToAction("RequestInfo");

}

#region Request info step

[HttpGet]

public ActionResult RequestInfo()

{

var requestInfo =

\_cookieManager.GetObject<RequestInfoModel>

(RequestStep.RequestInfo)

?? \_dataProvider.GetDefaultRequestInfoModel();

return RequestInfoPage(requestInfo);

}

[HttpPost]

public ActionResult

RequestInfo(RequestInfoModel requestInfo)

{

if (!ModelState.IsValid)

return RequestInfoPage(requestInfo);

\_cookieManager.SetReusableCookie(requestInfo.IsReusable);

\_cookieManager

.SaveObject(RequestStep.RequestInfo, requestInfo);

return RedirectToAction("VehicleDetails");

}

private ActionResult

RequestInfoPage(RequestInfoModel requestInfo)

{

ViewBag.GroupsSelectList =

\_dataProvider.GetSafeRoadControlGroupsSelectList();

return View(requestInfo);

}

#endregion

#region Vehicle details step

[HttpGet]

public ActionResult VehicleDetails()

{

if (\_cookieManager.CheckPreviousCookies

(RequestStep.VehicleDetails) == false)

return RedirectToAction("RequestInfo");

var vehicleDetails =

\_cookieManager.GetObject<VehicleDetailsModel>

(RequestStep.VehicleDetails)

?? \_dataProvider.GetDefaultVehicleDetailsModel();

return VehicleDetailsPage(vehicleDetails);

}

[HttpPost]

public ActionResult VehicleDetails(VehicleDetailsModel

vehicleDetails, IList<VehicleModel> vehicleStructure,

string next, string back)

{

vehicleDetails.VehicleStructure = vehicleStructure;

if (!String.IsNullOrEmpty(next)) // Next clicked

{

ModelState.Clear();

if (TryValidateModel(vehicleDetails) == false)

{

vehicleDetails.VehicleStructure = null;

return VehicleDetailsPage(vehicleDetails);

}

}

if (vehicleStructure != null)

{

int minId = vehicleStructure.Select(vehicle =>

vehicle.Id).Concat(new int[] {0}).Min();

foreach (var vehicle in

vehicleStructure.Where(vehicle => vehicle.Id == 0))

vehicle.Id = --minId;

}

\_cookieManager.SaveObject

(RequestStep.VehicleDetails, vehicleDetails);

if (!String.IsNullOrEmpty(next)) // Next clicked

return RedirectToAction("VehicleScheme");

return RedirectToAction("RequestInfo"); // Back clicked

}

private ActionResult

VehicleDetailsPage(VehicleDetailsModel vehicleDetails)

{

vehicleDetails.CurrentOrgName = Authorize.User.OrgName;

int carrierId = vehicleDetails.CarrierId;

if (carrierId > 0)

{

var carrier =

\_dataProvider.GetCarrierById(carrierId);

if (carrier == null)

{

vehicleDetails.IsOwn = true;

vehicleDetails.CarrierName =

vehicleDetails.CurrentOrgName;

vehicleDetails.CarrierId = Authorize.User.IdOrg;

}

}

return View("VehicleDetails", vehicleDetails);

}

[HttpGet]

public ActionResult

GetVehicles(bool isOwn, int carrierOrgId)

{

if (isOwn)

carrierOrgId = Authorize.User.IdOrg;

var vehiclesList =

\_dataProvider.GetCarrierVehicles(carrierOrgId);

return partialView

("Partials/VehicleTables/VehicleTables", vehiclesList);

}

#endregion

#region Vehicle scheme step

[HttpGet]

public ActionResult VehicleScheme()

{

if (\_cookieManager.CheckPreviousCookies

(RequestStep.VehicleScheme) == false)

return RedirectToAction("RequestInfo");

var vehicleDetails =

\_cookieManager.GetObject<VehicleDetailsModel>

(RequestStep.VehicleDetails);

if (vehicleDetails == null ||

vehicleDetails.VehicleStructure == null)

return RedirectToAction("VehicleDetails");

var vehicleScheme =

\_cookieManager.GetObject<VehicleSchemeModel>

(RequestStep.VehicleScheme);

if (vehicleScheme == null)

vehicleScheme = \_dataProvider

.GetDefaultVehicleSchemeModel(vehicleDetails);

else

\_dataProvider.FillVehicleScheme(vehicleScheme,

vehicleDetails);

return VehicleSchemePage(vehicleScheme);

}

[HttpPost]

public ActionResult VehicleScheme(VehicleSchemeModel

vehicleSchemeModel, string next, string back)

{

if (!String.IsNullOrEmpty(next))

{

if (!ModelState.IsValid)

return VehicleSchemePage(vehicleSchemeModel);

}

if(\_cookieManager.CheckReusableCookie())

vehicleSchemeModel.LoadType =

vehicleSchemeModel.LoadTypeReusable.GetDisplayName();

\_cookieManager.SaveObject(RequestStep.VehicleScheme,

vehicleSchemeModel);

if (!String.IsNullOrEmpty(next)) // Next clicked

return RedirectToAction("Route");

return RedirectToAction("VehicleDetails");

}

private ActionResult

VehicleSchemePage(VehicleSchemeModel vehicleSchemeModel)

{

ViewBag.IsReusable =

\_cookieManager.CheckReusableCookie();

return View("VehicleScheme",vehicleSchemeModel);

}

#endregion

#region Route step

[HttpGet]

public ActionResult Route()

{

if(\_cookieManager

.CheckPreviousCookies(RequestStep.Route) == false)

return RedirectToAction("RequestInfo");

var routeModel = \_cookieManager

.GetObject<RouteModel>(RequestStep.Route)

?? new RouteModel();

if (\_cookieManager.CheckReusableCookie())

{

var routeForReusableModel

= new RouteForReusableModel()

{Route = routeModel.RouteForReusable};

return View("RouteForReusable",

routeForReusableModel);

}

return View("Route", routeModel);

}

[HttpPost]

public ActionResult Route(RouteModel routeModel,

string next, string back)

{

if (!String.IsNullOrEmpty(next)) // Next clicked

{

if (!ModelState.IsValid)

{

routeModel.Clear();

return View(routeModel);

}

}

routeModel.SetIfSerializePostInfo(true);

\_cookieManager

.SaveObject(RequestStep.Route, routeModel);

if (!String.IsNullOrEmpty(next)) // Next clicked

return RedirectToAction("RequestPreview");

return RedirectToAction("VehicleScheme");

}

[HttpPost]

public ActionResult RouteForReusable(RouteForReusableModel

routeForReusableModel, string next, string back)

{

if (!ModelState.IsValid && !String.IsNullOrEmpty(next))

return View(routeForReusableModel);

var routeModel = new RouteModel()

{RouteForReusable = routeForReusableModel.Route};

\_cookieManager

.SaveObject(RequestStep.Route, routeModel);

if (!String.IsNullOrEmpty(next)) // Next clicked

return RedirectToAction("RequestPreview");

return RedirectToAction("VehicleScheme");

}

#endregion

#region Request preview step

[HttpGet]

public ActionResult RequestPreview()

{

if (\_cookieManager.CheckPreviousCookies

(RequestStep.RequestPreview) == false)

return RedirectToAction("RequestInfo");

string note = \_cookieManager

.GetObject<string>(RequestStep.RequestPreview);

ViewData["Note"] = note;

var makeRequestModel = \_cookieManager.GetRequestModel();

var previewModel =

DataProvider.GetRequestPreview(makeRequestModel);

if(previewModel == null)

return RedirectToAction("RequestInfo");

return View(previewModel);

}

[HttpPost]

public ActionResult RequestPreview(string note,

string next, string back)

{

if (!String.IsNullOrEmpty(next)) // Send clicked

{

if (\_cookieManager.CheckPreviousCookies

(RequestStep.RequestPreview) == false)

return RedirectToAction("RequestInfo");

var requestModel = \_cookieManager.GetRequestModel();

\_dataProvider.SendRequest(requestModel);

\_cookieManager.RemoveRequestCookies();

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

}

\_cookieManager

.SaveObject(RequestStep.RequestPreview, note);

return RedirectToAction("Route");

}

#endregion

}

ПРИЛОЖЕНИЕ Б

(обязательное)

Исходный текст класса RequestManager

public class RequestManager : AbstractManager

{

public RequestManager(AuthUser user) : base(user) { }

public RequestManager(ISession session, AuthUser user)

: base(session, user) { }

private const int RequestStatusGroupId = 5;

private const int PaymentBasisGroupId = 7;

private const int CurrencyGroupId = 8;

#region Get Requests

public Request GetRequest(int id)

{

return Session.Query<Request>()

.FirstOrDefault(r => r.IdOrganization ==

CurrentUser.IdOrg && r.IdPk == id);

}

public IList<RequestsListElement> GetRequestsList()

{

var sql = new StringBuilder();

sql.AppendFormat("SELECT \* FROM TABLE({0}(",

TktsInfoPkg.GetRequests)

.Append("cast(:OrgId AS number)))");

try

{

var requests = Session.CreateSQLQuery(sql.ToString())

.AddEntity(typeof(RequestsListElement))

.SetInt32("OrgId", CurrentUser.IdOrg)

.List<RequestsListElement>()

as List<RequestsListElement>;

return requests.DistinctBy(x => x.IdPk)

.OrderByDescending(x => x.IdPk).ToList();

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

public IList<RequestsListElement>

GetUnissuedRequests(DateTime startDate, DateTime endDate)

{

var sql = new StringBuilder();

sql.AppendFormat("SELECT \* FROM TABLE({0}(",

TktsInfoPkg.GetUnissuedRequests)

.Append("cast(:OrgId AS number),

cast(:startDate AS date), cast(:endDate AS date)")

.Append("))");

try

{

return Session.CreateSQLQuery(sql.ToString())

.AddEntity(typeof(RequestsListElement))

.SetInt32("OrgId", CurrentUser.IdOrg)

.SetDateTime("startDate", startDate)

.SetDateTime("endDate", endDate)

.List<RequestsListElement>()

as List<RequestsListElement>;

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

public IList<RequestsListElement>

GetUnissuedRequests(RequestFilter filter)

{

var sql = new StringBuilder();

sql.AppendFormat("SELECT \* FROM TABLE({0}(",

TktsInfoPkg.GetUnissuedRequestsByFilter)

.Append("cast(:OrgId AS number), :requestNumber,

:vehicleNumber))");

try

{

return Session.CreateSQLQuery(sql.ToString())

.AddEntity(typeof(RequestsListElement))

.SetInt32("OrgId", CurrentUser.IdOrg)

.SetParameter("requestNumber",

filter.RequestNumber)

.SetParameter("vehicleNumber",

filter.FirstVehicleNumber)

.List<RequestsListElement>()

as List<RequestsListElement>;

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

public IList<RequestsListElement> GetPermits(DateTime

startDate, DateTime endDate, int pageNumber,

int recordsPerPage, out int pagesCount)

{

var getPermitsSql = new StringBuilder();

getPermitsSql.AppendFormat("SELECT \* FROM TABLE({0}(",

TktsInfoPkg.GetPermits)

.Append("cast(:OrgId AS number), cast(:startDate

AS date), cast(:endDate AS date), cast(:pageNumber AS number), cast(:recordsPerPage AS number)))");

var getPagesCountSql = new StringBuilder();

getPagesCountSql.AppendFormat("SELECT {0}(",

TktsInfoPkg.GetPermitCountPages)

.Append("cast(:OrgId AS number), cast(:startDate

AS date), cast(:endDate AS date), cast(:recordsPerPage AS number)) from dual");

try

{

var tmp = Session

.CreateSQLQuery(getPagesCountSql.ToString())

.SetInt32("OrgId", CurrentUser.IdOrg)

.SetDateTime("startDate", startDate)

.SetDateTime("endDate", endDate)

.SetInt32("recordsPerPage", recordsPerPage)

.UniqueResult();

pagesCount = Convert.ToInt32(tmp);

return Session

.CreateSQLQuery(getPermitsSql.ToString()) .AddEntity(typeof(RequestsListElement))

.SetInt32("OrgId", CurrentUser.IdOrg)

.SetDateTime("startDate", startDate)

.SetDateTime("endDate", endDate)

.SetInt32("pageNumber", pageNumber)

.SetInt32("recordsPerPage", recordsPerPage)

.List<RequestsListElement>()

as List<RequestsListElement>;

}

catch (Exception ex)

{

Logger.Error(ex);

pagesCount = 1;

throw;

}

}

public IList<RequestsListElement>

GetPermits(RequestFilter filter)

{

var sql = new StringBuilder();

sql.AppendFormat("SELECT \* FROM TABLE({0}(",

TktsInfoPkg.GetPermitsByFilter)

.Append("cast(:OrgId AS number), :requestNumber,

:vehicleNumber, :permitNumber))");

try

{

return Session.CreateSQLQuery(sql.ToString())

.AddEntity(typeof(RequestsListElement))

.SetInt32("OrgId", CurrentUser.IdOrg)

.SetParameter("requestNumber",

filter.RequestNumber)

.SetParameter("vehicleNumber",

filter.FirstVehicleNumber)

.SetParameter("permitNumber",

filter.PermitNumber)

.List<RequestsListElement>()

as List<RequestsListElement>;

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

#endregion

#region Insert Requests

public int InsertRequest(Request request)

{

try

{

// определяем номер расчета (максимальный по группе УТК)

DateTime firstDayOfYear =

new DateTime(DateTime.Now.Year, 1, 1);

int? maxNumRequest = Session.Query<Request>()

.Where(r => r.SafeRoadControlGroup == request

.SafeRoadControlGroup

&& r.DateRequest >= firstDayOfYear)

.Max(r => (int?)r.NumRequest) ?? 0;

request.NumRequest = (int)maxNumRequest + 1;

request.IdPk = InsertEntity(request);

// Вставка транспортных средств из заявки

foreach (var vehicle in request.VehicleStructure)

{

vehicle.IdPk = InsertEntity(vehicle);

// Вставка особенностей ТС

foreach (var feature in vehicle.Features)

feature.IdPk = InsertEntity(feature);

// Вставка осей ТС

foreach (var axle in vehicle.Axles)

axle.IdPk = InsertEntity(axle);

}

// Вставка маршрутных точек из заявки

foreach (var point in request.RoutePoints)

point.IdPk = InsertEntity(point);

// Тип статуса "Отправлено перевозчиком"

// из справочника clstkts

var statusType = Session

.Query<RequestStatusСlassifier>()

.FirstOrDefault(s =>

s.GroupId == RequestStatusGroupId

&& s.StatusNumber == 1);

// Вставка статуса

var status = new RequestStatus()

{

Request = request,

Note = "Отправлено перевозчиком",

StatusDate = DateTime.Today,

StatusСlassifier = statusType,

Login = CurrentUser.Email

};

status.IdPk = InsertEntity(status);

// Для заявления на одноразовое спецразрешение

// устанавливаем поле NM\_ROUT

if (!request.IsReusable)

UpdateRouteTextForRequest(request.IdPk);

return request.IdPk;

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

private void UpdateRouteTextForRequest(int requestId)

{

var sql = new StringBuilder();

sql.AppendFormat("UPDATE REQUEST SET NM\_ROUT=(SELECT

ROUTNM FROM TABLE({0}", RdinfoRoutTktsPkg.ListAggRout)

.Append("(:idrequest))) where IDREQUEST=:idrequest");

try

{

Session.CreateSQLQuery(sql.ToString())

.SetInt32("idrequest", requestId)

.ExecuteUpdate();

Session.Flush();

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

#endregion

#region Delete requests

public int DeleteRequest(int id)

{

var isAnyRequest = Session.Query<Request>()

.Any(r => r.IdOrganization == CurrentUser.IdOrg

&& r.IdPk == id);

if (isAnyRequest == false)

return -1;

const string sql = "DELETE FROM REQUEST WHERE

IDREQUEST=cast(:idrequest AS number)";

try

{

return Session.CreateSQLQuery(sql)

.SetInt32("idrequest", id)

.ExecuteUpdate();

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

#endregion

#region Payments

public int AddPayment(Payment payment)

{

payment.Basis = Session.Query<TktsСlassifier>()

.FirstOrDefault(c => c.GroupId == PaymentBasisGroupId

&& c.TypeNumber == payment.BasisTypeNumber);

payment.Currency = Session.Query<TktsСlassifier>()

.FirstOrDefault(c => c.GroupId == CurrencyGroupId &&

c.TypeNumber == payment.CurrencyTypeNumber);

payment.IdPk = InsertEntity(payment);

return payment.IdPk;

}

public int DeletePayment(int id)

{

var isAnyPayment = Session.Query<Payment>()

.Any(p => p.Request.IdOrganization == CurrentUser.IdOrg

&& p.IdPk == id);

if (isAnyPayment == false)

return -1;

const string sql = "DELETE FROM PAYMENT WHERE

IDPAYMENT=cast(:id AS number)";

try

{

return Session.CreateSQLQuery(sql)

.SetInt32("id", id)

.ExecuteUpdate();

}

catch (Exception ex)

{

Logger.Error(ex);

throw;

}

}

public IList<Payment> GetPayments(int requestId)

{

return Session.Query<Payment>()

.Where(p => p.Request.IdPk == requestId &&

p.Request.IdOrganization == CurrentUser.IdOrg)

.ToList();

}

#endregion

}

ПРИЛОЖЕНИЕ В

(обязательное)

Спецификация проекта

ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов