|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» |
| **Димитровградский инженерно-технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** |

Факультет информационных технологий и социальных экономических наук

Кафедра информационных технологий

Направление 09.03.02 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К БАКАЛАВРСКОЙ РАБОТЕ НА ТЕМУ**

Разработка приложения для службы аутентификации проекта в системе   
управления решениями для ООО «АИС Город»

Студент

группы ИСТ-41 М.А. Лялин

(подпись)

Руководитель

ст. преподаватель Н.С. Казынбаев

(подпись)

Заведующий кафедрой

доцент, к.п.н. О.А. Ракова

(подпись)

Димитровград 2023

|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» |
| **Димитровградский инженерно-технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** |

**ФАКУЛЬТЕТ** ИТиСЭН **КАФЕДРА** Информационные технологии

**НАПРАВЛЕНИЕ ПОДГОТОВКИ** 09.03.02 **ГРУППА** ИСТ-41

«УТВЕРЖДАЮ»

Заведующий кафедрой

/Ракова О.А./

«\_\_\_\_\_»\_\_\_\_\_\_\_2023г.

**ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ**

1. Фамилия, имя, отчество Лялин Максим Андреевич

2. Тема работы Разработка приложения для службы аутентификации проекта в системе управления решениями для ООО «АИС Город»

3. Срок сдачи студентом готовой работы

4. Руководитель бакалаврской работы Казынбаев Никита Сергеевич

Дата выдачи задания «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_\_ г.

Руководитель бакалаврской работы

(подпись руководителя)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись студента)

1. Исходные данные к работе:

требования к уровню разработки программного продукта:

а) язык программирования C#;

б) язык программирования Angular.

2. Содержание работы:

*а) теоретическая часть:*

– исследование предметной области;

*б) экспериментальная часть:*

– разработка приложения;

– подробное описание реализации продукта;

– тестирование и отладка программного продукта.

3. Основная литература по теме работы:

а) Изучаем Angular 2 / пер. с англ. Р. Н. Рагимова; под науч. ред. А. Н. Киселева. - М.: ДМК Пресс, 2017. - 354 с.:

б) Разработка веб-приложений с использованием AngularJS. – Пер. c англ. Киселева А. Н., М.: ДМК Пресс, 2014. – 394с.: ил.

в) Angular для профессионалов. — СПб.: Питер, 2018. — 800 с.: ил. — (Серия «Для профессионалов»).

г) Марченко, А. Л. Основы программирования на С# 2.0 [Текст] / А.Л. Марченко. – М. : Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011. – 552 c.

д) Подбельский, В. В. Язык С#. Базовый курс [Текст] / В.В. Подбельский. – М. : Финансы и статистика, Инфра-М, 2011. – 384 c.

4. Отчетный материал работы:

а) пояснительная записка;

б) графический материал:

– макетно-экспериментальная часть программного модуля;

– презентация в формате Power Point.

**КАЛЕНДАРНЫЙ ГРАФИК ВЫПОЛНЕНИЯ БАКАЛАВРСКОЙ РАБОТЫ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №  п/п |  | Объем этапа в % | Сроки  выполнения | | Примечание |
| начало | Конец |
| 1. | Постановка задачи | 2 |  |  |  |
| 2. | Анализ предметной области | 15 |  |  |  |
| 3. | Разработка алгоритма решения основной задачи | 15 |  |  |  |
| 4. | Разработка интерфейса программы | 30 |  |  |  |
| 5 | Реализация алгоритма | 30 |  |  |  |
| 6. | Оформление пояснительной за­писки | 8 |  |  |  |
| 7. | Представление работы на проверку и отзыв руководителя бакалаврской работы | 100 |  |  |  |
| 8. | Представление работы на проверку на антиплагиат | 100 |  |  |  |
| 9. | Представление работы заведующей кафедрой | 100 |  |  |  |
| 10. | Защита на заседании ГЭК | 100 |  |  |  |

Руководитель: /Казынбаев Н.С./

РЕФЕРАТ

Лялин М.А. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ СЛУЖБЫ АУТЕНТИФИКАЦИИ ПРОЕКТА В СИСТЕМЕ УПРАВЛЕНИЯ РЕШЕНИЯМИ ДЛЯ ООО «АИС ГОРОД»: Бакалаврская работа / ДИТИ НИЯУ МИФИ № 09.03.02.05.2023. – Димитровград, 2023. – стр. 53, рис. 12, библ. назв. 15, табл. 1, приложений 5.

Ключевые слова: c#, АВТОРИЗАЦИЯ, АУТЕНТИФИКАЦИЯ, ANGULAR.

Цель работы – разработка приложения для службы аутентификации проекта в системе управления решениями ООО «АИС Город».

Полученные результаты: разработано приложение для аутентификации, исследована предметная область. Аутентификация выполняется с помощью сервиса ЖКХ ID.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc138767972)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 9](#_Toc138767973)

[1.1 Основные функции, цели ООО «АИС Город» 9](#_Toc138767974)

[1.2 Программные продукты ООО «АИС Город» 11](#_Toc138767975)

[1.3 Система управления решениями. 14](#_Toc138767976)

[1.4 Используемые среды разработки 15](#_Toc138767977)

[1.5 Angular 16](#_Toc138767978)

[1.6 Этапы SSO в языке программирования Angular 17](#_Toc138767979)

[1.7 SpaYarpASPNetCore 18](#_Toc138767980)

[Выводы по разделу 1 20](#_Toc138767981)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 21](#_Toc138767982)

[2.1 Исследование предметной области 21](#_Toc138767983)

[2.2 Подключение и настройка AspNetCore.SpaYarp в приложении 21](#_Toc138767984)

[2.3 Добавление авторизации с помощью ЖКХ ID. 23](#_Toc138767985)

[2.4 Добавление политик и прав доступа. 28](#_Toc138767986)

[2.5 Структура базы данных. 35](#_Toc138767987)

[Выводы по разделу 2 38](#_Toc138767988)

[3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 39](#_Toc138767989)

[3.1 Описание видов и методов тестирования 39](#_Toc138767990)

[3.2 Достоинства и недостатки какждого метода: 40](#_Toc138767991)

[3.2.1. Метод «черного ящика» 40](#_Toc138767992)

[3.2.2. Метод «белого ящика» 40](#_Toc138767993)

[3.2.3. Метод «регрессионного тестирования» 41](#_Toc138767994)

[3.2.4. Метод «Интерактивного тестирования» 41](#_Toc138767995)

[3.2.5. Метод «стресс тестирования» 41](#_Toc138767996)

[3.2.6. Метод «тестирования ошибок и исключений» 42](#_Toc138767997)

[3.3 Обоснование выбора метода тестирования 42](#_Toc138767998)

[3.4 Описание тестовых наборов данных 43](#_Toc138767999)

[Выводы по разделу 3 45](#_Toc138768000)

[ЗАКЛЮЧЕНИЕ 46](#_Toc138768001)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 47](#_Toc138768002)

[ПРИЛОЖЕНИЕ А –](#_Toc138768003) [ФРАГМЕНТ КОДА ФАЙЛА PROGRAM.CS 49](#_Toc138768004)

[ПРИЛОЖЕНИЕ Б –](#_Toc138768005) [ФРАГМЕНТ КОДА AUTHORIZATIONCONTROLLER 50](#_Toc138768006)

[ПРИЛОЖЕНИЕ В –](#_Toc138768007) [ФРАГМЕНТ КОДА PROGRAM.CS 51](#_Toc138768008)

[ПРИЛОЖЕНИЕ Г –](#_Toc138768009) [ФРАГМЕНТ КОДА USERHASRIGHTHANDLER 52](#_Toc138768010)

[ПРИЛОЖЕНИЕ Д –](#_Toc138768011) [ФРАГМЕНТ КОДА APIJWTBEAREREVENTS 53](#_Toc138768012)

ВВЕДЕНИЕ

Авторизация и аутентификация являются важнейшими аспектами информационной безопасности в веб-приложениях.

Аутентификация – это процесс проверки подлинности пользователей при доступе к конфиденциальной информации в системе. Авторизация – это процесс проверки прав и доступов пользователей к определенным ресурсам и функциям в системе.

Без надежной процедуры аутентификации и авторизации, злоумышленники могут легко получить доступ к конфиденциальной информации пользователей. Это может привести к утечке личных данных пользователей, а также угрозе их безопасности и приватности.

Кроме того, без аутентификации и авторизации невозможно установить и контролировать уровень доступа пользователей к ресурсам и функциям веб-приложения. Это может привести к нежелательным операциям, избыточным правам доступа, а также к угрозе безопасности системы в целом.

Актуальность темы исследования обусловлена необходимостью разработать приложение для службы аутентификации в проекте «Система управления решениями».

Целью выпускной квалификационной работы является разработать приложение для службы аутентификации в проекте «Система управления решениями», которая позволит ограничить права пользователей.

Объектом исследования выпускной квалификационной работы является аутентификация в проекте.

Предметом исследования является программная реализация службы аутентификации в проекте.

Задачами бакалаврской работы являются:

* выполнить анализ материалов по теме исследования;
* изучить структуру и принцип работы приложения «Система управления решениями»;
* добавить аутентификацию с помощью сервиса ЖКХ ID;
* добавить разграничение пользователя по ролям и добавить политики доступа;
* выполнить тестирование программного продукта.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Основные функции, цели ООО «АИС Город»

Полное наименование: Общество с ограниченной ответственностью «АИС ГОРОД».

Сокращённое наименование: ООО «АИС Город».

Основным видом деятельности компании является Разработка компьютерного программного обеспечения. Также ООО «АИС Город» работает еще по 13 направлениям. Организация насчитывает 2 учрежденные компании. Действует с 27.11.2003. Клиентами компании являются 23 региона России. Более 16 млн человек пользуются личными кабинетами и электронными сервисами, созданными ООО «АИС Город» [1].

Благодаря сотрудничеству с управляющими и ресурсоснабжающими организациями, органами государственной власти и местного самоуправления программные продукты полностью отвечают требованиям законодательства РФ и созданы в соответствии с реальными потребностями всех задействованных сторон. Программные продукты, разработанные компанией АИС Город, успешно используются и высоко ценятся рядом регионов Российской Федерации. Каждая информационная система предлагает реальное решение существующей проблемы отрасли жилищно-коммунального хозяйства.

Работа каждой из систем обеспечивает всех участников рынка ЖКХ достоверной и актуальной информацией, а в комплексе — системы формируют базовые сведения о населении, необходимые во многих отраслях, дают возможность создать консолидированный массив информации по жилому фонду и составу жильцов как по региону в целом, так и по отдельному муниципальному образованию [2].

Основными целями компании являются:

* обеспечение актуальной, достоверной информацией о сооружениях, их конструктивных особенностях, их техническом состоянии, о числе проживающих, об объемах и условиях оказания коммунальных услуг, их качестве, о оплате и начислениях, о работах на объектах жилищного фонда и коммунальной инфраструктуры, о планах осмотров и работ по их выполнении, о текущих плановых и капитальных ремонтах, о работах по благоустройству, о обращениях и жалобах граждан;
* уменьшение временных трудовых затрат на сбор сведений о ЖКХ;
* получение прозрачной информации о финансовых потоках;
* увеличение управляемости отрасли, повышение качества жилищно-коммунальных услуг;
* уменьшение финансовых затрат при оказании мер социальной поддержки при оплате ЖКХ;
* создание в единых диспетчерских с пунктами приема жалоб и обращений, контактными центрами для улучшения качества услуг в ЖКХ;
* создание единого окна ЖКХ «единого окна ЖКХ», внедрение единых квитанций;
* уменьшение временных затрат на общение с поставщиками услуг и бухгалтериями;
* улучшение качества выполняемой персоналом работы за счет наличия механизма сквозного контроля;
* повышение управляемости отраслью за счет наличия детальной достоверной информации;
* снижение недовольства потребителей за счет наличия механизма контроля исполнительской дисциплины.

В качестве рабочего места был предоставлен компьютер с операционной системой Windows 10 и программным обеспечением, которое использовалось для выполнения задания и доступа в Интернет. Для доступа ко всем необходимым ресурсам были выделены личные логин и пароль доступа.

С целью эффективного управления процессом разработки программного обеспечения в компании используется серверное веб-приложение для управления проектами и задачами и отслеживания ошибок под названием Redmine, распространяемое в свободном доступе. Оно позволяет вести систематизированную работу над проектом: создавать задачи, которые необходимо выполнить, назначать их на исполнение отдельным сотрудникам; осуществлять контроль сроков выполнения задач, вести учёт затраченного времени; проводить информирование всех разработчиков касательно развития проекта, его изменений; создавать справочники для разработчиков с информацией об особенностях работы отдельных модулей проекта или всего проекта в целом и многое другое.

Так как Redmine является серверным веб-приложением и устанавливается непосредственно на серверах организации, получить доступ к проектам Redmine закрытых организаций непосредственно из всемирной сети Интернет невозможно. Доступ осуществляется посредством виртуальных частных сетей, иначе называемых VPN (virtual private network). Используя специальные программы для доступа к таким сетям (в качестве такой программы используется Cisco AnyConnect Secure Mobility Client), и, имея соответствующие логин и пароль, можно подключаться к необходимым серверам для работы.

Использовав предоставленные компанией логин и пароль, подключившись с их помощью к корпоративному серверу, и выполнив вход в учётную запись Redmine, можно получить доступ ко всем актуальным задачам по всем разрабатываемым и поддерживаемым проектам компании. Задачи делятся по своему статусу на новые, в разработке, на тестировании, сделанные и т.д.

## 1.2 Программные продукты ООО «АИС Город»

ООО «АИС Город» имеет множество проектов. Одними из самых популярных являются «Паспортный Стол», «РИАС ЖКХ», «АСКУЭ АИС Город. Приборный учет», «Система Начислений», «Центр Льгот», «Сова. АИС Город», «CRM ЖКХ "Панда"», а также личные кабинеты поставщиков услуг ООО «РИЦ», АО «Ситиматик», ООО «ВИЦ», «МП ТРЕСТ ТЕПЛОФИКАЦИЯ», ООО «ЕИРЦ РБ», АО «ЕИРЦ ЛО» [2] и др.

Личные кабинеты поставщиков услуг играют важную роль в обеспечении доступности информации о ЖКХ услугах для пользователей.

Личные кабинеты осуществляют деятельность по приёму и учёту платежей населения за жилое помещение и коммунальные услуги. Сервис Личный кабинет клиента предназначен для пользователей, ежемесячно получающих единый платежный документ на оплату жилищно-коммунальных услуг.

Личный кабинет клиента позволяет:

* вносить показания приборов учета;
* оплачивать услуги ЖКХ;
* контролировать состояние лицевого счета;
* просматривать операции по лицевому счету;
* создавать выписки о начислениях и оплатах по лицевому счету;
* заказывать услуги.
* Личный кабинет клиента предлагает
* удобный интерфейс со всей необходимой информацией;
* простую процедуру регистрации;
* вход по ЕСИА (Единая система идентификации и аутентификации) через портал Госуслуг;
* возможность управлять несколькими лицевыми счетами;
* возможность подключить электронную квитанцию;
* возможность управлять подписками на информационные рассылки;

Услуги сервиса «Гипермаркет ЖКХ». Данный сервис предлагает широкий набор бытовых услуг и товаров, а также позволяет выбрать мастеров для проведения строительных работ по месту своего проживания.

Новый Личный кабинет клиента АО «ЕИРЦ ЛО» интегрирован с единой информационной системой, которая объединяет жилищно-коммунальную сферу Ленинградской области, что в итоге обеспечивает прозрачность информационных потоков в рамках ЖКХ всего региона.

Существуют следующие личные кабинеты следующих поставщиков услуг:

Личный кабинет акционерное общество «Единый информационно-расчётный центр Ленинградской области» (АО «ЕИРЦ ЛО») (рис. 1.1);

Личный кабинет общества с ограниченной ответственностью «Единый информационно-расчётный центр Республики Башкортостан» (ООО «ЕИРЦ РБ») (рис. 1.2);

Личный кабинет общества с ограниченной ответственностью «Энергосбыт Волга» (ООО «ЭСВ») (рис. 1.3).

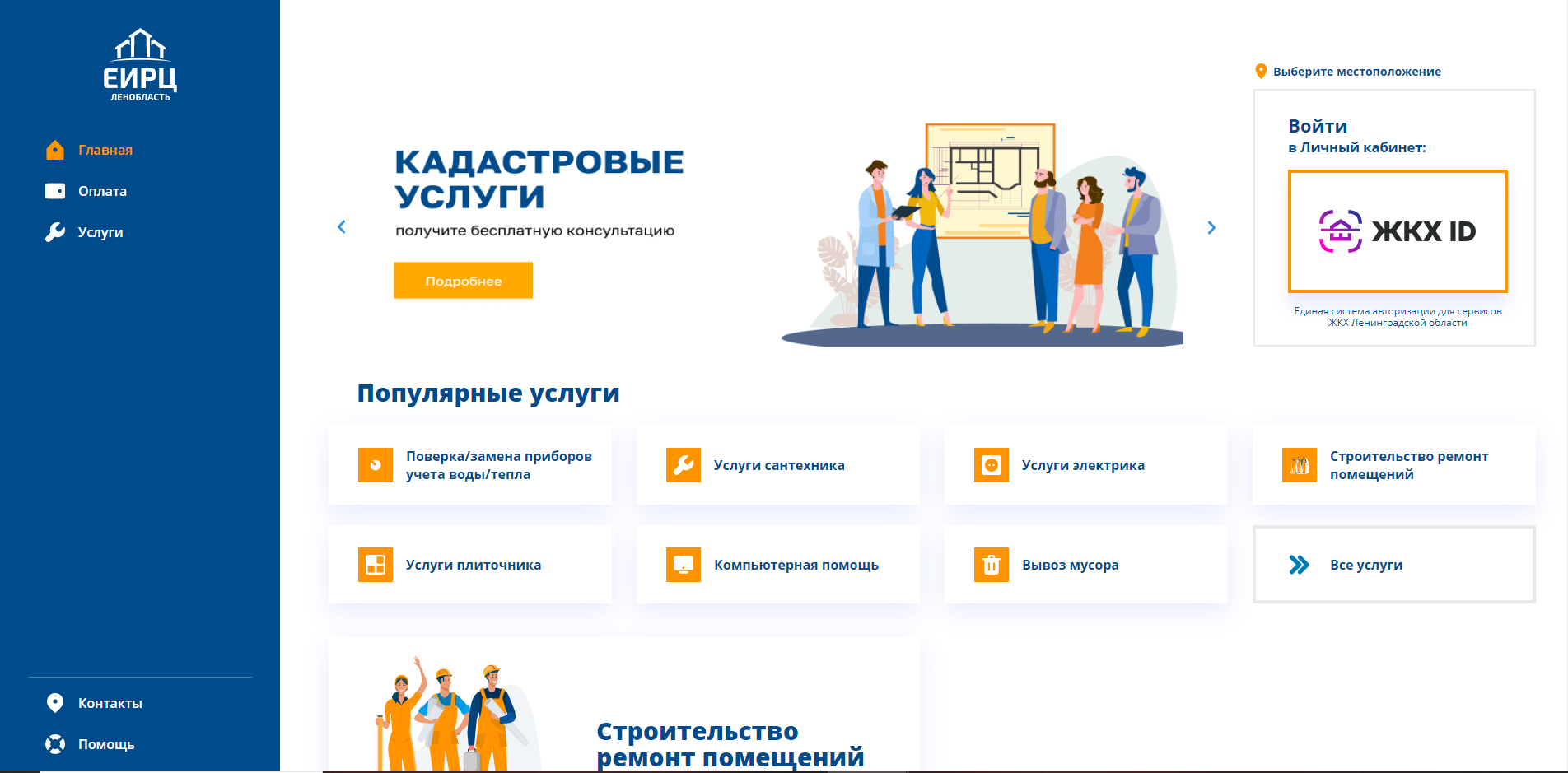


Рисунок 1.1 – Личный кабинет АО «ЕИРЦ ЛО»

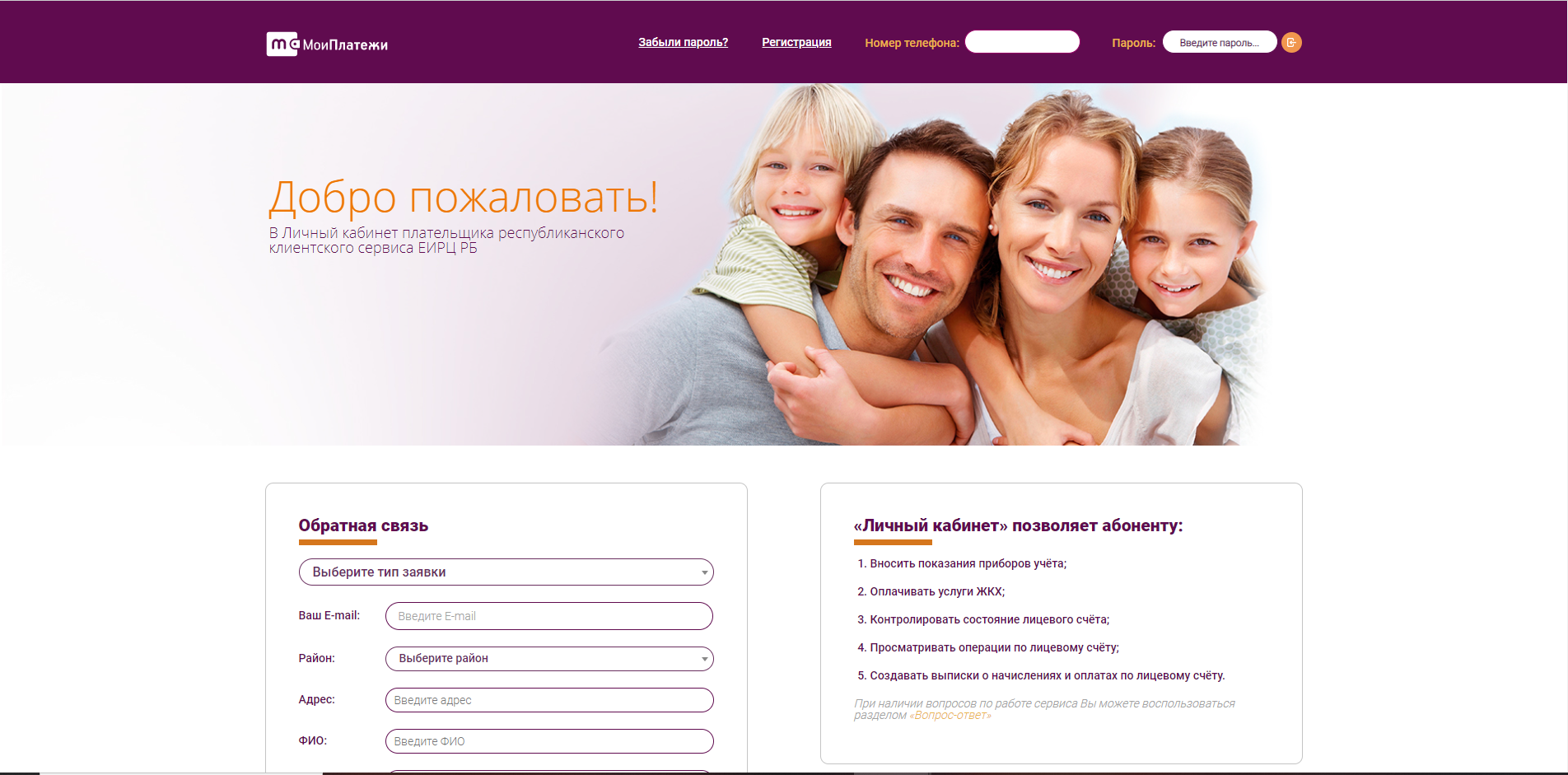


Рисунок 1.2 - Личный кабинет ООО «ЕИРЦ РБ»

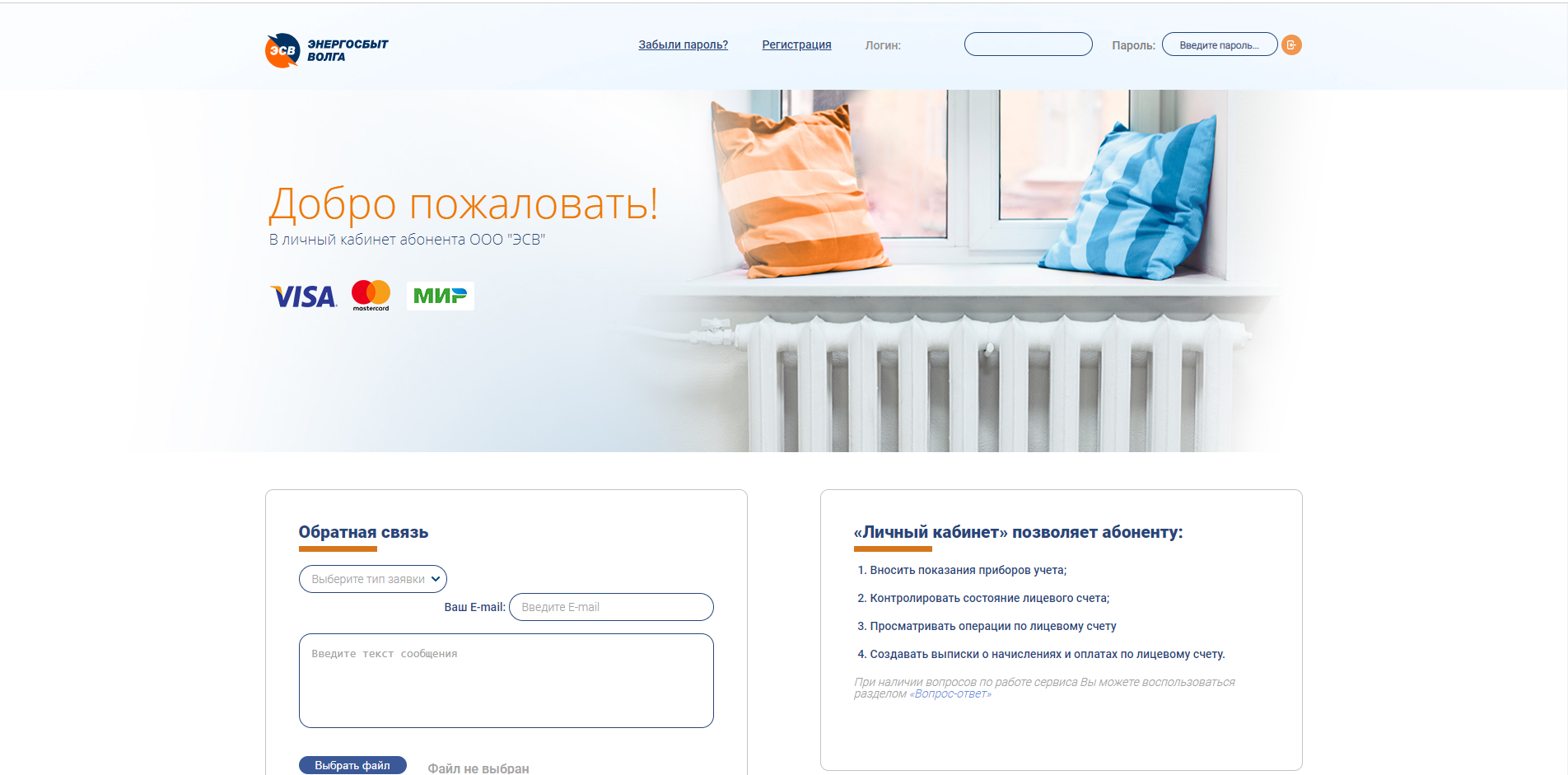


Рисунок 1.3 - Личный кабинет ООО «ЭСВ»

1.3 Система управления решениями.

Система управления решениями это комплексное решение (интерфейсы, сервисы, методы инструментария), для реализации различных типов задач и их параметров, а также их мониторинга жизненного цикла.

Модуль «Задачи» предоставляет комплексное решение для создания и мониторинга выполнения задач в проектах, доступных для администрирования в системе управления решениями. Задачи представляют собой различные команды в виде запросов, инициирующие выполнение действий в проекте и приводящие к результату. Как правило, данные задачи являются полезным инструментом в процессе сопровождения проектов. Доступные виды задач и параметры определяются средой исполнения.

Интерфейс в системе АИС СУП позволяет:

* отслеживать жизненный цикл исполнения задачи,
* посредством HTTP запросов получать доступные задачи для среды,
* получать информацию о необходимых параметрах задачи,
* инициировать выполнение задачи.
  1. Используемые среды разработки

Интегрированная среда разработки Visual Studio − это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки [3].

Visual Studio Code (VS Code) – это бесплатная среда разработки, которая может быть использована для разработки Angular-приложений. Визуальная студия, разработанная Microsoft, предоставляет множество инструментов для создания проектов Angular и упрощение процесса разработки.

Средства разработки в Visual Studio Code для Angular включают:

1. Поддержка TypeScript – VS Code поддерживает TypeScript, что позволяет разрабатывать приложения с использованием типизированного языка.

2. Расширения – Имеется огромное количество расширений для VS Code, которые обеспечивают дополнительные функции в редакторе кода, например, Angular Language Service, Angular Essentials, Angular Snippets.

3. Отладчик – Встроенный отладчик позволяет выполнять отладку кода в Angular, а также отображать значения приложения и переменных в процессе выполнения.

4. Live Server – Расширение Live Server для VS Code предоставляет возможность запускать веб-сервер в режиме реального времени, что позволяет мгновенно обновлять веб-страницы при изменении кода.

5. Git – VS Code включает в себя интеграцию с системой контроля версий git, что упрощает управление версиями кода в Angular-проектах.

6. IntelliSense – IntelliSense позволяет быстро и эффективно написать код благодаря автоматическому завершению кода, встроенным подсказкам и автоматическому форматированию.

7. Терминал – встроенный терминал позволяет выполнять команды оболочки, что упрощает процесс установки и настройки Angular-проектов.

Применение VS Code для разработки Angular-приложений существенно упрощает и ускоряет процесс разработки благодаря широкому набору интегрированных средств и расширений, которые обеспечивают бесперебойную работу и настройку проекта [4].

1. Angular

Angular – это открытый и бесплатный фреймворк для разработки веб-приложений, который был создан компанией Google. Он позволяет создавать динамические и масштабируемые приложения с помощью HTML, CSS и JavaScript [5].

Основные принципы Angular:

* Компонентный подход. Приложение разбивается на независимые компоненты, которые могут быть вновь использованы в других частях приложения.
* Директивы. Позволяют добавлять дополнительную функциональность к HTML элементам.
* Сервисы. Используются для обмена данными между компонентами и хранения общих данных.
* Инъекция зависимостей. Позволяет управлять зависимостями между компонентами и сервисами.

Angular также предоставляет множество инструментов для обеспечения безопасности и удобства использования приложений, таких как:

* Angular Forms. Они позволяют создавать формы с валидацией и обработкой пользовательского ввода.
* Angular Router. Он позволяет управлять навигацией в приложении и защищать определенные страницы от неавторизованного доступа.
* Angular HTTP Client. Он используется для обмена данными с сервером и может быть настроен для использования SSL-шифрования.
* Angular Security предоставляет инструменты для обеспечения безопасности приложения, такие как защита от XSS и CSRF атак.

Для создания авторизации на сайте с использованием механизма SSO в языке программирования Angular можно использовать различные инструменты, такие как:

* Angular Social Login. Это библиотека для аутентификации через социальные сети.
* Angular JWT. Это библиотека для работы с JSON Web Tokens, которые могут быть использованы для аутентификации пользователей.
* Angular Firebase. Это платформа для создания приложений, которая предоставляет инструменты для аутентификации и хранения данных.

В целом, Angular является мощным инструментом для создания безопасных и удобных приложений, который предоставляет множество инструментов и возможностей для разработчиков [6].

1.6 Этапы SSO в языке программирования Angular

SSO (Single Sign-On) – это механизм, который позволяет пользователям авторизоваться один раз и получить доступ к нескольким приложениям без необходимости повторной аутентификации. Это удобно для пользователей, так как они могут работать с несколькими приложениями с одним и тем же логином и паролем, а также для компаний, так как это упрощает управление доступом к приложениям.

Для реализации SSO в Angular можно использовать различные инструменты и технологии, такие как OAuth2, OpenID Connect, SAML и др.

OAuth2 – это протокол авторизации, который позволяет пользователям предоставлять доступ к своим данным третьим лицам без необходимости раскрытия своих учетных данных. Для использования OAuth2 в Angular можно использовать библиотеки, такие как angular-oauth2-oidc или ng2-cordova-oauth.

OpenID Connect – это расширение протокола OAuth2, которое добавляет аутентификацию пользователя. Это расширение использует JSON Web Tokens (JWT) для передачи информации об аутентификации. Для использования OpenID Connect в Angular можно использовать библиотеку angular-auth-oidc-client [7].

SAML (Security Assertion Markup Language) – это открытый стандарт обмена данными аутентификации, основанный на языке XML. Веб-приложения используют SAML, чтобы передавать аутентификационные данные между сторонами процесса: а именно, между системой управления доступами и провайдером услуг. Этот стандарт используется для авторизации пользователей. Для использования SAML в Angular можно использовать библиотеку angular-saml2-js.

Для реализации SSO в Angular необходимо выполнить следующие шаги:

1. Выбрать технологию SSO, которая наилучшим образом соответствует требованиям проекта.
2. Настроить идентификационный провайдер и сервис-провайдер.
3. Настроить Angular для работы с выбранной технологией SSO.
4. Реализовать механизм авторизации в Angular приложении.
5. Протестировать механизм SSO на соответствие требованиям проекта. В целом, реализация SSO в Angular может быть сложной задачей, но использование готовых библиотек и технологий может значительно упростить этот процесс [8].

## SpaYarpASPNetCore

SpaYarpASPNetCore - это фреймворк для разработки веб-приложений на основе [ASP.NET](http://asp.net/) Core и Single Page Application (SPA). Он позволяет разработчикам строить проекты SPA с использованием фронтенд-фреймворков, таких как Angular, React и Vue.js.

SpaYarpASPNetCore предоставляет возможность разработки проектов SPA, в которых весь контент загружается один раз при первом открытии сайта, и все последующие запросы к серверу выполняются асинхронно, без перезагрузки страницы. Это делает приложение более отзывчивым и улучшает пользовательский опыт [9].

Основными преимуществами SpaYarpASPNetCore являются:

1. Упрощение процесса разработки фронтенда и бэкенда. SpaYarpASPNetCore позволяет начинать разработку фронтенда сразу после создания проекта, и бэкенда после того, как разработчик подготовил API.

2. Поддержка множества фреймворков. SpaYarpASPNetCore поддерживает множество фронтенд-фреймворков, что позволяет разработчикам выбирать инструмент, который лучше всего подходит их проекту.

3. Эффективная оптимизация. SpaYarpASPNetCore управляет загрузкой скриптов и стилей, использует кэширование и поддерживает сжатие, что улучшает производительность приложений.

4. Удобство при развертывании. SpaYarpASPNetCore может быть развернут с помощью Azure App Service, Heroku, AWS и других облачных сервисов.

5. Гибкие возможности настройки. SpaYarpASPNetCore обладает гибкими возможностями настройки фронтенда и бэкенда, что позволяет разработчикам создавать более сложные и интегрированные приложения.

SpaYarpASPNetCore содержит в себе ряд удобных инструментов и функциональных компонентов:

1. Набор классов и методов для подключения и настройки приложения SPA к YARP [ASP.NET](http://asp.net/) Core.

2. Возможность расширения и настройки системы маршрутизации запросов для SPA-приложений.

3. Поддержка кэширования и сжатия данных для улучшения производительности приложения.

4. Набор инструментов для работы с пользовательскими запросами и обработки передаваемых данных.

5. Встроенные средства для логирования и мониторинга работы приложения.

Библиотека SpaYarpASPNetCore создана на основе опыта и лучших практик разработки SPA-приложений и позволяет значительно снизить время и усилия, затрачиваемые на разработку и поддержку таких приложений. Она является идеальным решением для разработчиков, которые хотят создавать высококачественные SPA-приложения с максимальной производительностью и эффективностью работы на серверной части [10].

Выводы по разделу 1

В данном разделе приведена характеристика компании ООО «АИС Город», её основные направления деятельности и устройство рабочего процесса. Изучена и подробно описана предметная область. Рассмотрены необходимые средства разработки, такие как: Visual Studio 2022, Visual Studio Code, SpaYarpAspNetCore.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Исследование предметной области

При аутентификации в проекте, пользователя переадресовывает на страницу с авторизацией ЖКХ ID. На ней пользователь вводит свои данные или же входит с помощью соц сетей, после чего на серверную часть приложения отправляются данные об этом пользователе. Далее все эти данные хранятся на сервере и через SpaYarp отправляются во фронтенд по запросу (рис.2.1).

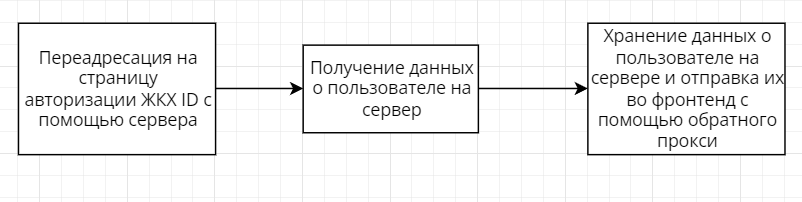


Рисунок 2.1 – Процесс аутентификации

После того, как исследована предметная область, объект и его свойства, необходимо начать разработку структуры модуля в базе данных.

2.2 Подключение и настройка AspNetCore.SpaYarp в приложении

Структура базы данных модуля офлайн-продаж лотерейных билетов для ООО «АИС Город» была реализован на языке T-SQL.

С выпуском [ASP.NET](https://asp.net/) Core Preview 4 команда [ASP.NET](https://asp.net/) Core представила новый опыт работы с шаблонами SPA. Основное преимущество этого нового опыта в том, что теперь можно запускать и останавливать проекты бэкэнда и клиента независимо друг от друга. Это очень приветствуется и ускоряет процесс разработки. Однако он также включает в себя еще одно более спорное изменение. Старые шаблоны обслуживали клиентское приложение как часть хоста [ASP.NET](https://asp.net/) Core и пересылали запросы на SPA. В новых шаблонах URL SPA используется для запуска приложения, а запросы к бэкэнду перенаправляются встроенным прокси-сервером разработчика SPA. AspNetCore.SpaYarp использует другой подход. Вместо использования сервера разработки SPA для пересылки запросов на хост / бэкэнд он использует YARP для пересылки всех запросов, которые не могут быть обработаны хостом, клиентскому приложению. Это работает аналогично старым шаблонам, но с преимуществом новых шаблонов для запуска и остановки проектов бэкэнда и клиента независимо друг от друга. После создания проекта нужно подключить к приложению Spa.Yarp[8].

Для добавления этого пакета, нужно зайти в меню управления пакетами Nuget, найти его там и нажать на кнопку установить. Установленный пакет SpaYarp позволит работать с библиотекой (рис. 2.2).

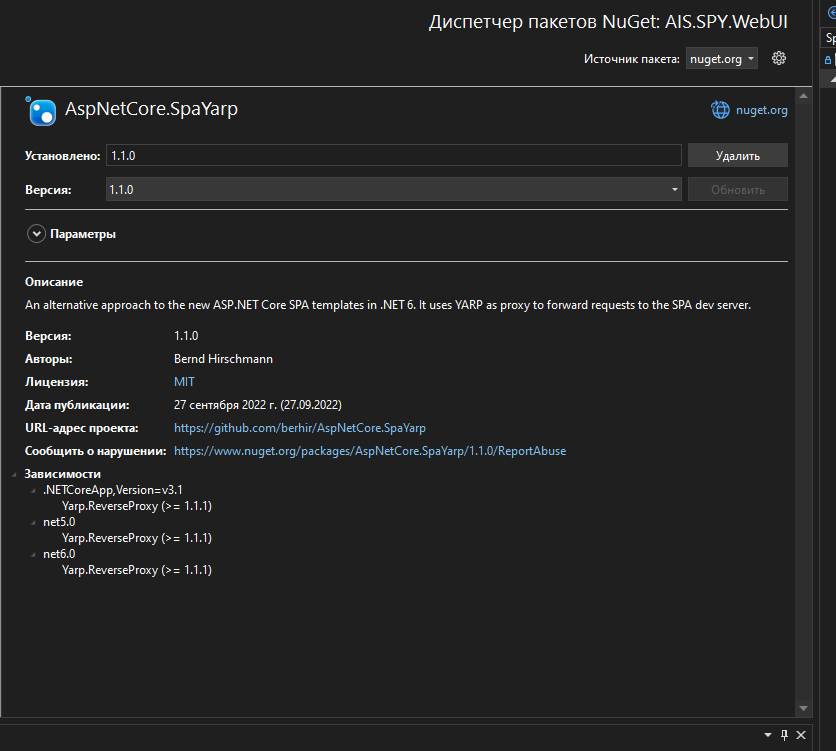


Рисунок 2.2 – Установка пакета Spa.Yarp

Для настройки работы Spa.Yarp Proxy был добавлен в файл Program.cs следующий код:

builder.Services.AddAuthentication(options =>

{

options.DefaultScheme = "Cookies";

options.DefaultChallengeScheme = "oidc";

})

.AddCookie("Cookies")

.AddOpenIdConnect("oidc", options =>

{

options.Authority = builder.Configuration.GetValue<string>("SSO:Authority");

options.RequireHttpsMetadata = false;

options.ClientId = builder.Configuration.GetValue<string>("SSO:ClientId");

options.ClientSecret = builder.Configuration.GetValue<string>("SSO:ClientSecret");

options.ResponseType = "code";

options.Prompt = "select\_account"; // насильно заставляем показать страницу входа

options.SaveTokens = true;

options.GetClaimsFromUserInfoEndpoint = true;

options.ClaimActions.MapAll();

options.Scope.Clear();

options.Scope.Add("openid");

options.Scope.Add("profile");

options.Scope.Add("email");

options.Scope.Add("offline\_access");

options.Scope.Add("phone");

options.Scope.Add("marketplace");

options.TokenValidationParameters.NameClaimType = "name";

options.TokenValidationParameters.RoleClaimType = "role";

options.AccessDeniedPath = "/ExternalError";

});

app.UseSpaYarpMiddleware();

builder.Services.AddProxy((httpClientBuilder) =>

{

httpClientBuilder.AddUserAccessTokenHandler();

});

app.Map(new PathString("/api"), app =>

{

app.RunProxy(context => context

.ForwardTo(builder.Configuration.GetValue<string>("API"))

.AddXForwardedHeaders()

.Send());

});

endpoints.MapSpaYarp();

Данный код прописывается в файле Program.cs для того чтобы при запуске приложения, запускался и прокси, который и позволит обмениваться данными между Angular приложением и C# приложением.

2.3 Добавление авторизации с помощью ЖКХ ID

Для добавления авторизации в серверную часть приложения, нужно в файле appsettings.json добавить конфигурацию для подключения к ЖКХ ID. Конфигурация для подключения к ЖКХ ID представлена в следующем фрагменте кода:

"SSO": {

"Authority": "https://sso-test.aisgorod.ru",

"ClientId": "4928fbcd-fd29-404f-a515-cdbe587dab80",

"ClientSecret": "f07d6fe0-c11d-41ae-96ba-ff8e173d640e"

},

Authority отвечает за ссылку, на которую будет переход для авторизации. ClientId и ClientSecret являются значениями, без которых будет невозможн переход на страницу авторизации.

Данный фрагмент кода добавляет переменные, которые получит сервер, после авториззации с помощью ЖКХ ID. А также происходит добавление в проект Cookie. Которые будут храниться на серверной части и потом передаваться в клиентскую часть приложения.

Далее нужно создать AuthorizationController, в котором будет метод Logout, который будет ждать пока клиентская часть приложения вызовет его. Он будет подчищать все данные Cookie и делать выход из аккаунта. Содержимое файла AuthorizationController приведено ниже:

namespace AIS.SPY.WebUI.Controllers

{

[Route("[controller]")]

[Controller]

public class AuthorizationController : ControllerBase

{

/// <summary>

/// Выход из системы.

/// </summary>

/// <returns>Перенаправление на главную</returns>

[Authorize]

[HttpGet("Logout")]

public async Task<ActionResult> Logout()

{

await HttpContext.SignOutAsync();

return Ok();

}

}

}

Важным этапом работы авторизации и аутентификации проекта является получение всех нужных данных пользователя. Для этого был создан файл IdentityController.cs. Конструктор файла приведен ниже:

public class IdentityController : ControllerBase

{

private readonly IConfiguration \_configuration;

private readonly UsersRepository \_usersRepository;

private readonly UserRolesRepository \_userRolesRepository;

public IdentityController(IConfiguration configuration, UsersRepository usersRepository, UserRolesRepository userRolesRepository)

{

\_configuration = configuration;

\_usersRepository = usersRepository;

\_userRolesRepository = userRolesRepository;

}

Также в этом файле есть функция, которая обращается к базе данных и получает оттуда все нужные данные. На следующем фрагменте кода приведен метод GetUserInformation, который получает данные текущего пользователя из базы данных:

public async Task<IActionResult> GetUserInformation()

{

try

{

var client = new HttpClient();

var disco = await client.GetDiscoveryDocumentAsync(\_configuration.GetValue<string>("Authority"));

string accessToken = Request.Headers.First(x => x.Key == "Authorization").Value;

UserInfoResponse responseIdentity = await client.GetUserInfoAsync(new UserInfoRequest

{

Address = disco.UserInfoEndpoint,

Token = accessToken.Substring("Bearer ".Length)

});

var ssoId = User.Claims.Where(x => x.Type == "urn:sso:user\_uuid").Count() > 0 ? User.Claims.SingleOrDefault(x => x.Type == "urn:sso:user\_uuid").Value : "";

var userInformation = new UserInformation()

{

SSOId = Guid.Parse(ssoId),

Login = responseIdentity.Claims.Where(x => x.Type == "preferred\_username").Count() > 0 ? responseIdentity.Claims.SingleOrDefault(x => x.Type == "preferred\_username").Value : "",

Name = responseIdentity.Claims.Where(x => x.Type == "name").Count() > 0 ? responseIdentity.Claims.SingleOrDefault(x => x.Type == "name").Value : "",

Email = responseIdentity.Claims.Where(x => x.Type == "email").Count() > 0 ? responseIdentity.Claims.FirstOrDefault(x => x.Type == "email").Value : "",

};

var userInSystemSPY = await \_usersRepository.GetBySSOId(ssoId);

if (userInSystemSPY == null)

{

await \_usersRepository.InsertAsync(new User()

{

SSOId = userInformation.SSOId.ToString(),

Name = userInformation.Login,

});

}

else

{

if (!userInSystemSPY.Name.Equals(userInformation.Login))

{

userInSystemSPY.Name = userInformation.Login;

await \_usersRepository.UpdateAsync(userInSystemSPY);

}

userInformation.IsBlocked = userInSystemSPY.IsDeleted;

}

var userRoles = await \_userRolesRepository.GetRolesBySSOId(ssoId);

userInformation.Roles = userRoles?.ToList();

return Ok(userInformation);

}

catch (Exception ex)

{

return BadRequest();

}

Данный метод достает из базы данных все данные о пользователе, такие как: SSOId, login, Name, Email. После чего проверяет зарегистрирован ли пользовтель в системе SPY, если пользователь зарегистрирован, то идет изменение данных о пользователе. Если пользователя нету в системе, то происходит его добавление.

В клиентской части приложения был добавлен сервис, позволяющий обращаться в серверную часть для получения данных о пользователе. Код сервиса представлен во фрагменте ниже:

import {Inject, Injectable} from '@angular/core';

import {HttpClient} from "@angular/common/http";

export interface UserInfo {

ssoId: string;

login: string;

name: string;

email: string;

isDefault: boolean;

isAdmin: boolean;

isTasksEdit: boolean;

isSettingsEdit: boolean;

isBlocked: boolean;

roles: Array<any>;

tokenLifeTime: number;

}

@Injectable({

providedIn: 'root'

})

export class AuthService {

private modules: any = null;

userInfo: any = null;

// Политики

isDefault: boolean = true;

isAdmin: boolean = false;

isTasksEdit: boolean = false;

isSettingsEdit: boolean = false;

isBlocked: boolean = false;

constructor(private http: HttpClient, @Inject('BASE\_URL') private baseUrl: string) {

}

isAuthenticated(): boolean {

if (this.userInfo) {

return true;

}

return false;

}

async getUserInformation() {

const userInfo = await this.http.get<UserInfo>(`/api/identity/getUserInformation`).toPromise();

if (userInfo) {

this.userInfo = userInfo;

}

}

async logout() { this.modules = null; this.userInfo = null;

await this.http.get(this.baseUrl + 'Authorization/Logout').toPromise();

window.location.reload();}}

В данном сервисе имеются три метода:

isAuthenticated;

getUserInformation;

logout.

Первый метод проверяет, заполнены ли поля у класса UserInfo,которые заполняются при успешной авторизации. Если поля заполнены, то пользователь аутентифицирован в системе и может пользоваться ей. Если нет, то пользователя переадресовывает на страницу с авторизацией ЖКХ ID.

Метод getUserInformation получает информацию о пользователе после авторизации и заполняет поля из класса UserInfo этими данными о пользователе.

Метод logout вызывает метод в серверной части, который переадресовывает пользователя на страницу с авторизацией, а также данный метод очищает класс UserInfo.

Также в клиентской части приложения был написан еще один сервис AuthGuard, который позволяет проверять через какое то время, аутентифицирован ли пользователь в системе или нет. Ниже приведен фрагмент кода файла authGuard.ts:

import {Injectable} from '@angular/core';

import {ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot, UrlTree} from '@angular/router';

import {Observable} from 'rxjs';

import {AuthService} from "../services/auth.service";

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

constructor(private auth: AuthService,

private router: Router) {

}

canActivate(

route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

return !this.auth.isAuthenticated() ? this.router.navigate(['/access-denied']) : true;

}

}

2.4 Добавление политик и прав доступа

Для корректой работы аутентификации в проекте были добавлены политики и права доступа. Политики и права доступа - это важные аспекты информационной безопасности в веб-приложениях, позволяющие управлять доступом к конфиденциальной информации и ресурсам в системе. Они определяют, какие пользователи могут получить доступ к различным функциям и ресурсам в приложении, а также как эти пользователи могут использовать эти ресурсы.

Политики и права доступа могут быть описаны в виде набора правил и настроек безопасности, которые определяют, кто может получить доступ к системе, какие действия могут быть выполнены в системе, и какие ресурсы могут использоваться.

Преимущества использования политик и прав доступа в веб-приложениях:

* + - 1. Защита от несанкционированного доступа - Политики и права доступа защищают ресурсы и данные от потенциальных угроз, таких как злоумышленники или случайный доступ пользователей, что в свою очередь делает работу приложения более безопасной.
      2. Ограничение доступа к конфиденциальной информации - Политики и права доступа могут ограничивать доступ к конфиденциальной информации, такой как информация о клиентах, платежные данные и другие данные, которые нужно защищать.
      3. Управление доступом - Политики и права доступа управляют доступом пользователей в системе, определяют, кто имеет доступ к каким данных и какие операции могут выполняться с этими данными.
      4. Безопасность данных - Политики и права доступа улучшают безопасность данных путем защиты их от случайных и несанкционированных изменений или удаления.
      5. Улучшение производительности - Политики и права доступа повышают производительность системы, ограничивая доступ к ресурсам и операциям.

Правильное применение политик и прав доступа в веб-приложениях является критически важным для обеспечения безопасности данных и управления доступом. Они позволяют контролировать доступ пользователей к ресурсам и эффективно управлять системой, надежно защищая ее от возможных угроз.

Для создания прав доступа был добавлен класс в серверной части приложения написанный на языке C#. Код классаа UserRights представлен ниже:

public class UserRights

{

/// <summary>

/// Доступ к вкладке главная.

/// </summary>

[Required]

public bool Default { get; set; }

/// <summary>

/// Управление вкладкой "Администрирование".

/// </summary>

[Required]

public bool Admin { get; set; }

/// <summary>

/// Просмотр вкладки "Главная".

/// </summary>

[Required]

public bool Home { get; set; }

/// <summary>

/// Просмотр вкладки "Задачи".

/// </summary>

[Required]

public bool Tasks { get; set; }

/// <summary>

/// Изменение вкладки "Задачи".

/// </summary>

[Required]

public bool TasksEdit { get; set; }

/// <summary>

/// Просмотр вкладки "Ошибки".

/// </summary>

[Required]

public bool Errors { get; set; }

/// <summary>

/// Изменение вкладки "Ошибки".

/// </summary>

[Required]

public bool ErrorsEdit { get; set; }

/// <summary>

/// Просмотр вкладки "Статистики".

/// </summary>

[Required]

public bool Statistics { get; set; }

/// <summary>

/// Изменение вкладки "Статистики".

/// </summary>

[Required]

public bool StatisticsEdit { get; set; }

/// <summary>

/// Просмотр вкладки "Настройки".

/// </summary>

[Required]

public bool Settings { get; set; }

/// <summary>

/// Изменение вкладки "Настройки".

/// </summary>

[Required]

public bool SettingsEdit { get; set; }

В данном классе содержаться поля с типом bool которые определяют есть ли у пользователя доступ к вкладке или нет, а ткаже определяют может ли пользователь изменять данные в этих вкладках. Далее для удобства был написан класс UserRightRequirement который позволяет применять несколько прав на одну политику. Ниже приведен фрагмент кода этого класса:

public class UserRightRequirement: IAuthorizationRequirement

{

public string[] RightNames { get; set; }

public UserRightRequirement(string rightName)

{

RightNames = new string[1] { rightName };

}

public UserRightRequirement(string[] rightNames)

{

RightNames = rightNames;

}

}

В данном классе содержится два конструктора, которые в свою очередь работают почти одинаковым образом, преобразуя получаемые права в массив строк.

Следующим шагом было добавление класса с политиками. Фрагмент кода приведен ниже:

public static class Policies

{

/// <summary>

/// Политика доступа по умолчанию.

/// </summary>

//[Obsolete("Используйте другие права...")]

public const string DEFAULT = "spy:default";

/// <summary>

/// Политика доступа для доступа и изменению вкладки "Администрирование".

/// </summary>

public const string ADMIN = "spy:admin";

/// <summary>

/// Политика доступа для доступа к вкладке "Главная".

/// </summary>

public const string HOME = "spy:home";

/// <summary>

/// Политика доступа для доступа к вкладке "Задачи".

/// </summary>

public const string TASKS = "spy:tasks";

/// <summary>

/// Политика доступа для доступа и изменению вкладки "Задачи".

/// </summary>

public const string TASKS\_EDIT = "spy:tasks:edit";

/// <summary>

/// Политика доступа для доступа к вкладке "Ошибки".

/// </summary>

public const string ERRORS = "spy:errors";

/// <summary>

/// Политика доступа для доступа и изменению вкладки "Ошибки".

/// </summary>

public const string ERRORS\_EDIT = "spy:errors:edit";

/// <summary>

/// Политика доступа для доступа к вкладке "Статистики".

/// </summary>

public const string STATISTICS = "spy:statistics";

/// <summary>

/// Политика доступа для доступа и изменению вкладки "Статистики".

/// </summary>

public const string STATISTICS\_EDIT = "spy:statistics:edit";

/// <summary>

/// Политика доступа для доступа к вкладке "Настройки".

/// </summary>

public const string SETTINGS = "spy:settings";

/// <summary>

/// Политика доступа для доступа и изменению вкладки "Настройки".

/// </summary>

public const string SETTINGS\_EDIT = "spy:settings:edit";

В данном классе содержутся наименования политик для того, чтобы их можно было добавить и в будущем сравнить с теми политиками, которые есть в базе данных (рис. 2.3.).

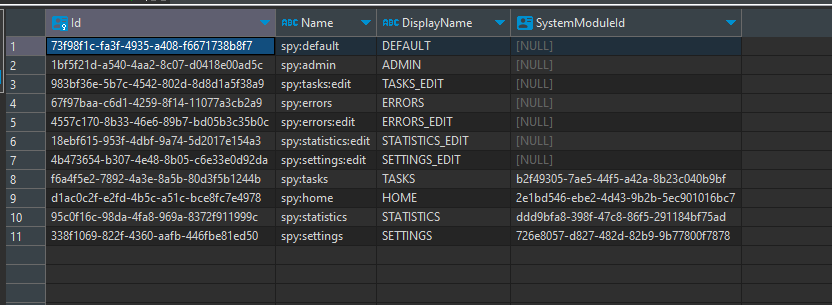


Рисунок 2.3 – Таблица Policies

Данная таблица содержит в себе наименование политики как в классе и отображаемое ей имя. Также содержит поле SystemModuleId, которое в свою очередь отвечает за привязку к модулю системы в проекте. Если у записи есть SystemModuleId значит данная запись дает доступ к просмотру этого модуля в приложении. Если SystemModuleId равен значении Null, то это значит что данная запись либо не дает доступа к этому модулю (такая как Default), либо она дает доступ к изменению данных в модуле(TASKS\_EDIT). Политика ADMIN даёт доступ ко всем модулям и изменению данных в них.

Следующим шагом было регистрирование политик в файле Program.cs. Ниже представлен код регистрирования политик:

var defaultReq = new ApiRightRequirement("spy").AllowAll();

options.AddPolicy(JwtConstants.ApiBasePolicy, policy => policy

.AddAuthenticationSchemes(JwtConstants.SsoJwtAuthScheme)

.RequireAuthenticatedUser()

);

options.AddPolicy(Policies.ADMIN, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Admin)))

);

options.AddPolicy(Policies.HOME, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Home)))

);

options.AddPolicy(Policies.ERRORS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Errors)))

);

options.AddPolicy(Policies.ERRORS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.ErrorsEdit)))

);

options.AddPolicy(Policies.DEFAULT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Default)))

);

options.AddPolicy(Policies.SETTINGS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Settings)))

);

options.AddPolicy(Policies.SETTINGS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.SettingsEdit)))

);

options.AddPolicy(Policies.STATISTICS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Statistics)))

);

options.AddPolicy(Policies.STATISTICS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.StatisticsEdit)))

);

options.AddPolicy(Policies.TASKS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Tasks)))

);

options.AddPolicy(Policies.TASKS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.TasksEdit)))

);

options.AddPolicy(JwtConstants.ApiUserPolicy, policy => policy

.AddAuthenticationSchemes(JwtConstants.SsoJwtAuthScheme)

//.RequireClaim("urn:sn:user\_id")

.RequireAuthenticatedUser()

В регистрировании политик выбирается конкретная политика, и регистрируются на неё права доступа, куда сможет или не сможет пользователь зайти. Для корректной работы политик доступа нужно у каждого метода или контроллера, которые используются для взаимодействия с вкладками, добавить аттрибут со значением подходящей политики. Ниже представлен фрагмент кода добавления аттрибута у метода и у контроллера:

Authorize(JwtConstants.ApiUserPolicy)]

[Authorize(Policy = Policies.ADMIN)]

[Route("[controller]/[action]")]

[ApiController]

public class UsersViewController : ViewBaseController<UsersView, UserView, Guid>

{

private readonly UsersView \_usersView;

public UsersViewController(UsersView usersView) : base(usersView)

{

\_usersView = usersView;

}

}

[Authorize(Policy = Policies.TASKS\_EDIT)]

[HttpPatch("{id}")]

public async Task<ActionResult> Cancel(Guid id)

{

try

{

await \_tasksRepository.Cancel(id);

return Ok();

}

catch (Exception ex)

{

ex.WriteExceptionLog("Error TaskRepository Cancel");

return BadRequest();

}

}

Данные аттрибуты не позволяют пользователям не имеющим политику пользоваться вкладками и видеть или изменять данные на них. Соответствующие аттрибуты были добавлены в каждый контроллер, для разграничения прав пользователей.

Для отслеживания прав пользователя был написан отдельный класс ApiRightHanler наследующийся от AuthorizationHandler<ApiRightRequirement>. Данный класс позволяет проверять есть ли у пользователя права доступа и связывает эти права доступа с политиками на уровне сервера. Ниже приведен фрагмент кода этого класса:

public class ApiRightHandler : AuthorizationHandler<ApiRightRequirement>

{

protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, ApiRightRequirement requirement)

{

context.Succeed(requirement);

return Task.CompletedTask;

}

}

Для успешного выполнения политик и прав доступа, в Angular компонентах были добавлены следующие проверки, позволяющие понять, есть ли у пользователя доступ к определенному модулю. В компоненте slidenav.component.ts была написана проверка, реализованная в конструкторе. При создании данного компонента, фронтенд получает от сервера данные, со списком доступных политик у пользователя. После чего в конструкторе происходит проверка, и если определенный модуль не доступен, то пользователю он виден не будет. Ниже приведен фрагмент кода проверки политик на стороне фронтенда:

ngOnInit(): void {

this.http.get('/api/GroupImplementations/GetInfo').subscribe((routes: any) => {

routes.forEach((groupImplementation: INavbarData) => {

this.navData.push(groupImplementation);

})

});

this.screenWidth = window.innerWidth;

let adminModule = this.navData.find(s => s.label == 'Администрирование');

if (adminModule) {

adminModule.isHidden = !this.authService.userInfo?.userRights.admin;

}

this.navData = this.navData.filter(s => !s.isHidden);

}

При первом входе в систему, с помощью ЖКХ ID данные пользователя записываются в базу данных проекта «Система управления решениями», а конкретно записываются:

* Имя пользователя
* SSOId

Это означает что пользователь связывается через сервис ЖКХ ID с помощью SSOId. В дальнейшем SSOId используется для получения списка доступных ролей и политик пользователя, а также остальных данных о нем.

2.5 Структура базы данных.

Для корректой работы аутентификации в базе данных были добавлены следующие таблицы:

* RoleModules;
* SystemModules;
* RolePolicies;
* UserRoles;
* Users;
* Roles;
* Policies.

Таблица Users хранит в себе данные о пользователях, их логин, SsoId и IsDeleted. SsoId это Id учетной записи, которая при первой авторизации заполняется. Если у пользователя нету SsoId он не сможет зайти в проект и получить к нему доступ. IsDeleted отвечает за то удален ли пользователь или нет. Это позволяет хранить информацию даже об удаленных из системы пользователях (рис. 2.4).

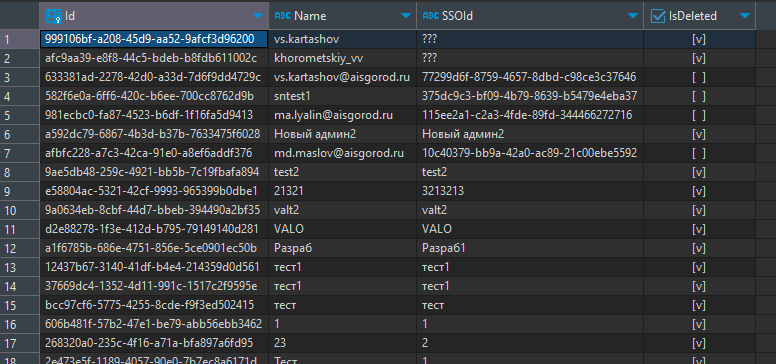


Рисунок 2.4 – Таблица Users

Таблица Roles хранит в себе наименования ролей. Она содержит столбец Id и наименование роли (рис. 2.5.).

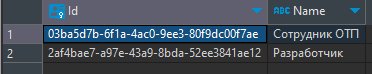


Рисунок 2.5 – Таблица Roles

Таблица UserRoles соединяет две таблицы Users и Roles посредством связи один ко многим. Это позволяет на одного пользователя добавить несколько ролей (рис. 2.6.).

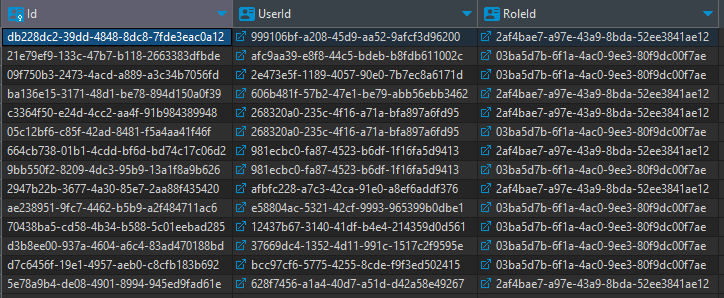


Рисунок 2.6 – Таблица UserRoles

Таблица SystemModules хранит в себе информацию о системных модулях доступных в системе на данный момент. Также эта табличка хранит в себе информацию о контроллере, который отвечает за этот модуль в серверной части приложения (рис. 2.7.).

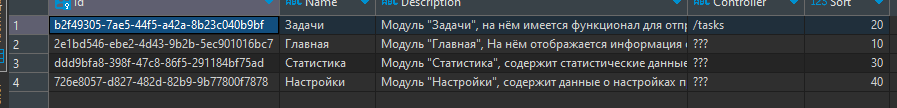


Рисунок 2.7 – Таблица Policies

Таблица RolePolicies связывает в себе таблицы с ролями и политиками таким образом что к определенной роли привязываются определенные политики связанные с ней. Это позволяет на одну роль привязать несколько политик чтобы пользователь мог пользоваться приложением (рис. 2.8.).

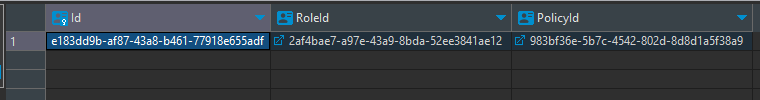


Рисунок 2.8 – Таблица RolePolicies

Таблица RoleModules отвечает за связь роли и доступными модулями. Это нужно для того что не всем пользователям будут доступны все модули (рис 2.9.).

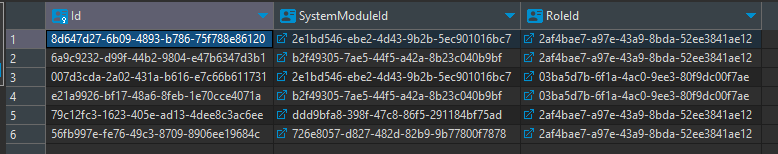


Рисунок 2.9 – Таблица RoleModules

С помощью данных таблиц приложение осуществаляет авторизацию пользователя в проекте, а также позволяет аутентифицировать пользователя и наделить определенными правами и политиками доступа для работы в проекте.

Выводы по разделу 2

В данном разделе был подробно описан алгоритм создания службы аутентификации. Была реализована функциональная схема с подробным её описанием. Подробно описана реализация приложения с представленными фрагментами кода. Описана установка и подключение дополнительной библиотеки для работы аутентификации. Описан алгоритм аутентификации и авторизации в проекте. Описан алгоритм работы клиентской части приложения. Была добавлена аутентификация на основе политик и прав доступа. Описана схема базы данных. Описано взаимодействие таблиц

3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

3.1 Описание видов и методов тестирования

Методы тестирования программного продукта являются одним из важных этапов разработки дипломного проекта. Они позволяют выявить ошибки и дефекты продукта и максимально снизить риски его неправильной работы. Вот некоторые методы, которые можно использовать при тестировании программного продукта для диплома:

1. Метод «черного ящика». Этот метод тестирования заключается в том, чтобы проверять функциональность программы без знания внутренней логики его работы. Тестирующий работает только с пользовательским интерфейсом, проверяя входные и выходные данные, а также корректность работы при различных условиях.
2. Метод «белого ящика». Этот метод тестирования заключается в том, чтобы тестировать продукт с знанием внутренней логики работы программы. Тестирующий проводит тестирование отдельных модулей, проверяя их работу на корректность, а также взаимодействие с другими модулями.
3. Метод «регрессионного тестирования». Этот метод тестирования направлен на выявление ошибок или дефектов в добавленных или измененных функциях путем повторного тестирования более ранних версий программы.
4. Метод «интерактивного тестирования». Этот метод тестирования предполагает составление списка сценариев использования продукта и проверку его работы на каждом из этих сценариев.
5. Метод «стресс-тестирования». Этот метод направлен на проверку работоспособности продукта при максимально возможной нагрузке. Тестирующий создает искусственно большую нагрузку на систему, чтобы выявить возможные проблемы при высоком использовании ресурсов.
6. Метод «методия тестирования ошибок и исключений». Этот метод направлен на проверку правильной обработки исключений и ошибок в программе. Тестирующий создает ситуации, которые могут привести к возникновению ошибок в программе и проверяет правильность обработки этих ошибок [11].

Все вышеперечисленные методы могут быть использованы в комбинации друг с другом при тестировании программного продукта для диплома. Важно выбрать подходящий метод для конкретного продукта и профессионально провести тестирование, чтобы продукт был максимально качественным и надежным.

3.2 Достоинства и недостатки какждого метода:

1. Метод «черного ящика»

Плюсы [12]:

* Эффективен для тестирования функциональности приложения с точки зрения пользователя;
* Использование реальных входных и выходных данных повышает достоверность результатов;
* Снижает риск повторного тестирования.

Минусы [12]:

* Нельзя проверить внутреннюю структуру системы;
* Невозможно тестирование основанные на сложных бизнес-логике;
* Этот метод может привести к непокрытию кода.

1. Метод «белого ящика»

Плюсы [13]:

* Эффективен для проверки внутренней структуры системы;
* Позволяет проверить функциональность каждого модуля приложения;
* Сокращает риск пропуска дефектов приложения.

Минусы [13]:

* Требуется профессиональный тестировщик с знанием внутренней структуры приложения;
* Тестирование занимает много времени и требует дополнительной структуры;
* Возможна утечка информации.

1. Метод «регрессионного тестирования»

Плюсы:

* Обнаружение дефектов уже имеющихся функций;
* Проверка изменений на баги.

Минусы:

* Может быть трудно сопровождаем;
* Требует большого количества времени и ресурсов;
* Сложно автоматизировать.

1. Метод «Интерактивного тестирования»

Плюсы:

* Полная проверка функциональности приложения;
* Определение ошибок в пользовательском интерфейсе и бизнес-логике;
* Эффективное тестирование новых функций.

Минусы:

* Требует большого количества времени и ресурсов;
* Одни и те же тесты могут проводиться несколько раз.

1. Метод «стресс тестирования»

Плюсы [14]:

* Выявление ошибок в системе, которые могут возникнуть при большой нагрузке;
* Помогает определить, как система реагирует при высокой нагрузке;
* Проверка масштабируемости приложения.

Минусы [14]:

* Требуется профессиональный тестировщик;
* Время занятий устройств.

1. Метод «тестирования ошибок и исключений»

Плюсы [15]:

* Обнаружение проблем в обработке исключений и ошибок;
* Помощь в определении возможных вредных последствий в случае ошибок приложения;
* Часто помогает в улучшении функциональности приложения.

Минусы [15]:

* Дорогой способ тестирования;
* Требует большого количесва времени.

3.3 Обоснование выбора метода тестирования

Исходя из вышеописанных видов и методов тестирования, мною был выбран метод тестирования черного ящика.

Достоинства метода:

* Тестирование методом «черного ящика» позволяет найти ошибки, которые невозможно обнаружить методом «белого ящика». Простейший пример: разработчик забыл добавить какую-то функциональность. С точки зрения кода все работает идеально, но с точки зрения спецификации это – сверхкритичный баг.
* «Черный ящик» позволяет быстро выявить ошибки в функциональных спецификациях (в них описаны не только входные значения, но и то, что мы должны в итоге получить). Если полученный при тестировании результат отличается от заявленного в спецификации, то у нас появляется повод для общения с аналитиком для уточнения конечного результата.
* Тестировщику не нужна дополнительная квалификация. Часто мы пользуемся различными сервисами и приложениями, не очень в них разбираясь. Для того, чтобы открыть инстаграм и обработать свою фотографию, нам совсем не нужно знать способ реализации фильтров. Мы хотим открыть фотографию, выбрать фильтр и получить красивую картинку на выходе. Задача тестировщика, который тестирует эту функцию в инстаграм, – убедиться, что пользователь получит эту самую красивую картинку в соответствии с выбранным фильтром. При этом нам совсем не обязательно иметь какую-либо специализацию – нужны лишь телефон и инстаграм.
* Тестирование проходит «с позиции пользователя». Пользователь всегда прав, он конечный потребитель практически любого ПО, а значит, ему должно быть удобно, комфортно и понятно.

Недостатки метода:

* Основным недостатком метода «черного ящика» является возможность пропуска границ и переходов, которые не очевидны из спецификации, но есть в реализации кода (собственно, это и заставляет тестировщиков использовать метод «белого ящика»).
* Можно протестировать только небольшое количество возможных вводных (входящих) значений; многие варианты остаются без проверки.
* Тесты могут быть избыточными, если разработчик уже проверил данную функциональность.
* При отсутствии четкой и полной спецификации проектировать тесты и тест-сценарии оказывается затруднительно.

3.4 Описание тестовых наборов данных

Набор тестовых данных с результатами тестирование данного программного продукта представлен в табл. 3.1.

Таблица 3.1−Тестирование

|  |  |  |
| --- | --- | --- |
| Осуществляемое взаимодействие | Ожидаемый результат | Фактический результат |
| Переход на страницу с проектом | Открытие страницы с авторизацией | Открытие страницы с авторизацией |
| Неправильный ввод логина и пароля пользователя | Появление сообщения об ошибке | Появление сообщения об ошибке |
| Правильный ввод логина и пароля | Открытие главной строницы проекта | Открытие главной строницы проекта |
| Вход под ролью «Разработчик» | Доступ ко всем модулям проекта | Доступ ко всем модулям проекта |
| Вход под ролью «Сотрудник ОТП» | Модуль «Администрация» не отображается у пользователя | Модуль «Администрация» не отображается у пользователя |
| Вход под пользователем, не имеющим ролей | Вкладки показаны, но данные в них не загружаются | Вкладки показаны, но данные в них не загружаются |
| Показ модуля «Администрирование» для пользователя с ролью «Разработчик» | Модуль «Администрирование» виден пользователю | Модуль «Администрирование» виден пользователю |
| Сокрытие модуля «Администрирование» для пользователя без роли «Разработчик» | Модуль «Администрирование» отсутствует | Модуль «Администрирование» отсутствует |

Выводы по разделу 3

В данном разделе была протестирована авторизация и аутентификация в проекте «система управления решениями». Ожидаемый результат полностью соответствует фактическому результату. Во время тестирования работоспособности приложения ошибок работы не было обнаружено.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было разработано приложение для службы аутентификации в проекте «Система Управления Решениями».

Были выполнены следующие задачи:

* анализ материалов по теме выпускной квалификационной работы;
* разработан сервис аутентификации;
* добавлены политики и права доступа;
* выполнено тестирование службы аутентификации.

Изучены и освоены принципы построения приложения в инструментальной среде разработки Microsoft Visual Studio 2022 и Microsot Visual Studio Code. С помощью данных программ была разработана служба аутентификации в проекте система управления решениями.

Исследованы методы тестирования и их классификация в соответствии с видом программного продукта. Разработанная служба аутентификации была протестирована с использованием метода «черного ящика». Результаты тестирования показали корректную и устойчивую работу.

Разработанная служба аутентификации позволяет разграничивать права пользователей в проекте «Система Управления Решениями», что позволяет ей работать корректно.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ООО «АИС Город» [Электронный ресурс]. Режим доступа: http://www.aisgorod.ru/ (дата обращения: 05.06.2023).
2. АИС Город, ООО [Электронный ресурс]: Тензор, CБИС. Режим доступа: https://sbis.ru/contragents/7302026625/730201001 (дата обращения: 05.06.2023).
3. Изучаем Angular 2 / пер. с англ. Р. Н. Рагимова; под науч. ред. А. Н. Киселева. - М.: ДМК Пресс, 2017. - 354 с.: ил.
4. Разработка веб-приложений с использованием AngularJS. – Пер. c англ. Киселева А. Н., М.: ДМК Пресс, 2014. – 394с.: ил.
5. Angular для профессионалов. — СПб.: Питер, 2018. — 800 с.: ил. — (Серия «Для профессионалов»).
6. AspNetCore.SpaYarp. – электронный ресурс, режим доступа: <https://github.com/berhir/AspNetCore.SpaYarp> (дата обращения: 20.06.2023).
7. Code.visualstudio.com. (n.d.). Visual Studio Code Documentation. – электронный ресурс, режим доступа:<https://code.visualstudio.com/docs> (дата обращения: 29.05.2023).
8. Марченко, А. Л. Основы программирования на С# 2.0 [Текст] / А.Л. Марченко. − М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, **2011**. − 552 c.
9. Подбельский, В. В. Язык С#. Базовый курс [Текст] / В.В. Подбельский. − М.: Финансы и статистика, Инфра-М, 2011. − 384 c.
10. Прайс, Джейсон Visual C# 2.0. Полное руководство [Текст] / Джейсон Прайс, Майк Гандэрлой. − М.: Век +, Корона-Век, Энтроп, **2010**. − 736 c.
11. Вичугова, А. А. Автоматизация процесса разработки программного обеспечения: методы и средства / А.А. Вичугова. - М.: Синергия, 2016. - 890 c.
12. Котляров, В.П. Основы тестирования программного обеспечения / В.П. Котляров. - М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2015. - 726 c.
13. Луиза, Тамре Введение в тестирование программного обеспечения. Руководство / Тамре Луиза. - М.: Диалектика / Вильямс, 2018. - 146 c.
14. Рекс Блек. Ключевые процессы тестирования - М.: Издательство Лори, 2014. - 544 с.
15. Арсеновски, Даниэль Рефакторинг в C# и ASP.NET для профессионалов / Даниэль Арсеновски. - М.: Вильямс, 2010. - 528 c

ПРИЛОЖЕНИЕ А

ФРАГМЕНТ КОДА ФАЙЛА PROGRAM.CS

builder.Services.AddAuthentication(options =>

{

options.DefaultScheme = "Cookies";

options.DefaultChallengeScheme = "oidc";

})

.AddCookie("Cookies")

.AddOpenIdConnect("oidc", options =>

{

options.Authority = builder.Configuration.GetValue<string>("SSO:Authority");

options.RequireHttpsMetadata = false;

options.ClientId = builder.Configuration.GetValue<string>("SSO:ClientId");

options.ClientSecret = builder.Configuration.GetValue<string>("SSO:ClientSecret");

options.ResponseType = "code";

options.Prompt = "select\_account"; // насильно заставляем показать страницу входа

options.SaveTokens = true;

options.GetClaimsFromUserInfoEndpoint = true;

options.ClaimActions.MapAll();

options.Scope.Clear();

options.Scope.Add("openid");

options.Scope.Add("profile");

options.Scope.Add("email");

options.Scope.Add("offline\_access");

options.Scope.Add("phone");

options.Scope.Add("marketplace");

options.TokenValidationParameters.NameClaimType = "name";

options.TokenValidationParameters.RoleClaimType = "role";

options.AccessDeniedPath = "/ExternalError";

});

ПРИЛОЖЕНИЕ Б

ФРАГМЕНТ КОДА AUTHORIZATIONCONTROLLER

namespace AIS.SPY.WebUI.Controllers

{

[Route("[controller]")]

[Controller]

public class AuthorizationController : ControllerBase

{

/// <summary>

/// Выход из системы.

/// </summary>

/// <returns>Перенаправление на главную</returns>

[Authorize]

[HttpGet("Logout")]

public async Task<ActionResult> Logout()

{

await HttpContext.SignOutAsync();

return Ok();

}

}

}

ПРИЛОЖЕНИЕ В

ФРАГМЕНТ КОДА PROGRAM.CS

builder.Services.AddAuthorization(options =>

{

var defaultReq = new ApiRightRequirement("spy").AllowUser();

options.AddPolicy(JwtConstants.ApiBasePolicy, policy => policy

.AddAuthenticationSchemes(JwtConstants.SsoJwtAuthScheme)

.RequireAuthenticatedUser()

);

options.AddPolicy(Policies.ADMIN, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Admin)))

);

options.AddPolicy(Policies.HOME, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Home)))

);

options.AddPolicy(Policies.ERRORS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Errors)))

);

options.AddPolicy(Policies.ERRORS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.ErrorsEdit)))

);

options.AddPolicy(Policies.DEFAULT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Default)))

);

options.AddPolicy(Policies.SETTINGS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Settings)))

);

options.AddPolicy(Policies.SETTINGS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.SettingsEdit)))

);

options.AddPolicy(Policies.STATISTICS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Statistics)))

);

options.AddPolicy(Policies.STATISTICS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.StatisticsEdit)))

);

options.AddPolicy(Policies.TASKS, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.Tasks)))

);

options.AddPolicy(Policies.TASKS\_EDIT, policy =>

policy.RequireAuthenticatedUser().AddRequirements(defaultReq, new UserRightRequirement(nameof(UserRights.TasksEdit)))

);

options.AddPolicy(JwtConstants.ApiUserPolicy, policy => policy

.AddAuthenticationSchemes(JwtConstants.SsoJwtAuthScheme)

.RequireAuthenticatedUser()

ПРИЛОЖЕНИЕ Г

ФРАГМЕНТ КОДА USERHASRIGHTHANDLER

namespace AIS.SPY.WebAPI.Services

{

public class UserHasRightHandler : AuthorizationHandler<UserRightRequirement>

{

protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, UserRightRequirement requirement)

{

var userRights = context.User.GetUserRights() ?? new UserRights();

foreach (var requirementRight in requirement.RightNames)

{

var propValue = userRights.GetType().GetProperty(requirementRight) ?? throw new ArgumentException(nameof(requirementRight));

bool data = (bool)(propValue.GetValue(userRights) ?? throw new ArgumentException(requirementRight));

if (data)

{

context.Succeed(requirement);

}

}

return Task.CompletedTask;

}

}

}

ПРИЛОЖЕНИЕ Д

ФРАГМЕНТ КОДА APIJWTBEAREREVENTS

public class ApiJwtBearerEvents : JwtBearerEvents

{

private readonly UsersRepository \_usersRepository;

private readonly UserRolesRepository \_userRolesRepository;

private readonly PoliciesRepository \_policiesRepository;

public ApiJwtBearerEvents(UsersRepository usersRepository, UserRolesRepository userRolesRepository, PoliciesRepository policiesRepository)

{

\_usersRepository = usersRepository;

\_userRolesRepository = userRolesRepository;

\_policiesRepository = policiesRepository;

}

/// <summary>

/// После валидации токена авторизации.

/// </summary>

/// <param name="context">Контекст</param>

public override async System.Threading.Tasks.Task TokenValidated(TokenValidatedContext context)

{

var nameIdentifier = context.Principal.FindFirstValue(ClaimTypes.NameIdentifier);

var ssoId = context.Principal.FindFirstValue("urn:sso:user\_uuid");

var userInSystemSPY = await \_usersRepository.GetBySSOId(ssoId);

var identity = new ClaimsIdentity();

if (userInSystemSPY != null)

{

var policies = await \_policiesRepository.GetPoliciesBySSOId(ssoId);

var userRights = new UserRights();

foreach (var policy in policies)

{

PropertyInfo propertyInfo = userRights.GetType().GetProperty(policy.displayName);

propertyInfo?.SetValue(userRights, true, null);

}

identity.AddClaim(new Claim("urn:spy:id", userInSystemSPY.Id.ToString(), ClaimValueTypes.Integer));

identity.AddClaim(new Claim("urn:spy:user\_rights", JsonSerializer.Serialize(userRights), ClaimValueTypes.String));

}

context.Principal?.AddIdentity(identity);

context.Success();

return/\* System.Threading.Tasks.Task.CompletedTask\*/;

}

}

Выпускная квалификационная работа выполнена мной самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«\_\_» \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (Ф.И.О.)