**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Кафедра телекоммуникаций и информационных технологий**

**РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ WEB ПЛАТФОРМЫ ДЛЯ КОНФЕРЕНЦИИ «КВАНТОВАЯ ЭЛЕКТРОНИКА 2025»**

Курсовая работа

Шкурной Марии Дмитриевны

обучающейся 3 курса специальности

«Компьютерная безопасность»

Научный руководитель:

ассистент кафедры

А.А. Афоненко

Минск, 2025

**ОГЛАВЛЕНИЕ**

|  |  |
| --- | --- |
| ВВЕДЕНИЕ ............................................................................................................ | 3 |
| ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ...................................................... | 4 |
| 1.1 Техническое задание ............................................................................... | 4 |
| 1.2 Реализация ............................................................................................... | 5 |
| 1.2.1 Архитектура ....................................................................................... | 5 |
| 1.2.2 Фреймворк и язык программирования ............................................ | 6 |
| 1.2.3 MVC .................................................................................................... | 8 |
| 1.2.4 Чистая архитектура: что это такое, преимущества и применение в проекте…………………………………………………………………………. | 10 |
| ГЛАВА 2. ВЗАИМОДЕЙСТВИЕ С ВНЕШНИМ МИРОМ .................... | 13 |
| 2.1 Взаимодействие с пользователями ...................................................... | 13 |
| 2.2 Интеграция со сторонними сервисами ............................................... | 14 |
| 2.2.1 SMTP-сервер ................................................................................... | 14 |
| 2.2.2 Файловое хранилище ..................................................................... | 14 |
| 2.2.3 PostgreSQL ...................................................................................... | 15 |
| ГЛАВА 3. БЕЗОПАСНОСТЬ ............................................................................... | 16 |
| 3.1 Аутентификация и авторизация пользователей. ASP.NET Core Identity………………………….................………………………………………. | 16 |
| 3.2 Ролевая модель доступа и авторизация на основе cookie ……......... | 18 |
| 3.3 Законодательные требования к системе ............................................. | 19 |
| 3.4 Архитектура безопасного приложения ............................................... | 20 |
| 3.5 Конвейер CI/CD обеспечения безопасности приложения .................  ГЛАВА 4. РАЗРАБОТКА……………………………………………………….. | 23  25 |
| ЗАКЛЮЧЕНИЕ ..................................................................................................... | 27 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ............................................. | 28 |

C:\Users\ladys\OneDrive\Изображения\Снимок.PNG

**ВВЕДЕНИЕ**

В современном мире веб-приложения стали неотъемлемой частью научной деятельности, облегчая сбор, обработку и обмен информацией между учёными, организаторами и техническими службами. Переход от традиционных, локально установленных систем к облачным и веб-ориентированным платформам позволяет значительно повысить эффективность работы: обеспечить круглосуточный доступ к данным, автоматизировать рутинные операции и гибко масштабировать инфраструктуру по мере роста нагрузки.

В условиях ужесточения требований законодательства об обработке персональных данных и роста числа целенаправленных атак на прикладные сервисы становится очевидным, что успешное внедрение веб-решений в научный процесс невозможно без всесторонней проработки вопросов безопасности. Необходимо не только реализовать требуемый функционал – регистрацию пользователей, управление их ролью и правами, приём платёжных документов, обработку заявок – но и обеспечить комплексную защиту от несанкционированного доступа, утечек и потери данных.

Целью курсовой работы является разработка безопасного веб-приложения для сайта конференции «Квантовая электроника 2025». Для достижения этой цели были поставлены следующие задачи:

* разработать серверную часть приложения в соответствии с техническим заданием и реализовать основную бизнес-логику, создав минимально жизнеспособный продукт;
* оценить текущий уровень безопасности приложения и проверить его соответствие законодательным требованиям;
* разработать многоуровневую схему безопасности, охватывающую все уровни – от физической инфраструктуры до приложений.

Практическая значимость работы заключается в создании функционального и безопасного веб-приложения, которое может быть использовано для проведения конференции «Квантовая электроника 2025». Разработанные меры безопасности, включая ролевую авторизацию, двухфакторную аутентификацию и защиту от веб-уязвимостей, могут быть адаптированы для других аналогичных проектов. Кроме того, исследование способствует углублению знаний о методах обеспечения безопасности в процессе разработки программного обеспечения, что имеет важное значение в условиях роста киберугроз и цифровизации.

**ГЛАВА 1.**

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

* 1. **Техническое задание**

В рамках курсового проекта была поставлена задача разработки серверной части веб-приложения, предназначенного для сопровождения научной конференции «Квантовая электроника 2025». Платформа должна обеспечить удобную и безопасную среду для взаимодействия между участниками конференции и администрацией сайта.

Согласно техническому заданию, основной функционал системы охватывает все этапы работы с докладами – от подачи заявки до подтверждения участия, а также включает средства администрирования и автоматизации ключевых процессов. Предполагалось, что платформа будет использоваться как зарегистрированными пользователями (в том числе потенциальными докладчиками), так и организаторами конференции, которым необходимо эффективно управлять потоком заявок и данных.

Система должна поддерживать регистрацию пользователей с последующим созданием персонального профиля, где можно указать основную информацию об участнике. Зарегистрированные пользователи получают возможность подать доклад и прикрепить текст тезисов. В случае одобрения основного тезиса организаторами, пользователю предоставляется возможность дополнительно загрузить расширенные материалы. Кроме того, участники, планирующие выступление, обязаны прикрепить квитанцию об оплате взноса, после чего администратор вручную подтверждает получение средств.

Одной из ключевых задач разработки стало создание удобного личного кабинета, через который участник может отслеживать текущий статус своей заявки, редактировать информацию, а при необходимости – передать доклад другому зарегистрированному участнику, который будет его представлять.

Административная часть платформы предусматривает инструменты для ручной и полуавтоматической модерации. Через специальный интерфейс администратор должен иметь возможность просматривать список поданных докладов, подтверждать или отклонять их, формировать состав оргкомитета, а также составлять расписание мероприятия. Отдельным направлением стала автоматизация части процессов, таких как обновление статусов участников при изменении условий или подтверждении оплаты.

Серверная часть должна была быть реализована с учётом расширяемости и возможного повторного использования для других конференций в будущем. Требовалось обеспечить логичную структуру кода, строгую разделённость слоёв и корректную обработку типичных пользовательских сценариев.

## **Реализация**

В рамках курсового проекта по разработке серверной части веб-приложения для научной конференции «Квантовая электроника 2025» необходимо обосновать выбор технологий и архитектуры, а также подробно описать их теоретическую реализацию. Техническое задание предусматривает создание удобной и безопасной платформы для взаимодействия участников и организаторов конференции, включающей регистрацию пользователей, подачу и модерацию докладов, управление расписанием и автоматизацию процессов. Платформа должна быть простой в разработке, поддержке и потенциально адаптируемой для будущих конференций. В данном разделе описаны выбранные архитектура, язык программирования, фреймворк, шаблон MVC и принципы чистой архитектуры, с аргументацией их соответствия задачам проекта и ссылками на авторитетные источники.

## 1.2.1 Архитектура

Для реализации проекта выбрана монолитная архитектура, которая объединяет все компоненты приложения – бизнес-логику, пользовательский интерфейс и взаимодействие с базой данных – в единый кодовый блок. Этот выбор обусловлен масштабом и задачами платформы, предназначенной для ограниченного числа пользователей, таких как участники и организаторы конференции. Монолитная архитектура упрощает разработку, тестирование и развертывание, что критично для небольшого проекта с ограниченными ресурсами и временными рамками, указанными в ТЗ [1].

В отличие от микросервисной архитектуры, которая требует значительных усилий для координации независимых сервисов, монолитное приложение минимизирует накладные расходы на межсервисное взаимодействие и снижает сложность поддержки.

Исследования подтверждают, что монолитная архитектура является оптимальной для приложений с ограниченной функциональностью, таких как системы управления мероприятиями, где сложное масштабирование не требуется [2].

Монолитная архитектура обеспечивает высокую производительность за счет отсутствия задержек, связанных с сетевым взаимодействием между сервисами, что важно для сценариев, таких как одновременная подача заявок или просмотр расписания [1]. Например, при массовой регистрации участников или загрузке тезисов докладов монолитное приложение может обрабатывать запросы быстрее, чем распределенная система, где требуется координация между сервисами. Это особенно актуально для проекта, где ожидается умеренная нагрузка, а высокая производительность является приоритетом для обеспечения удобства пользователей.

Кроме того, монолитная архитектура упрощает тестирование и отладку. Поскольку весь код находится в одном месте, разработчики могут легко отслеживать поток запросов и выявлять проблемы, что сокращает время на устранение ошибок. Это критично для обеспечения надежности приложения, особенно при обработке важных функций, таких как подача заявок, загрузка квитанций об оплате и модерация докладов. Например, если возникает ошибка при сохранении тезисов, разработчик может быстро проследить путь запроса от пользовательского интерфейса до базы данных, не разбираясь с сетевыми вызовами между сервисами.

Монолитная архитектура также поддерживает требование ТЗ о возможности повторного использования кода для других конференций. Единый кодовый блок легче адаптировать или клонировать для последующих мероприятий, чем распределенную систему, где требуется настройка множества сервисов. Например, функциональность личного кабинета, подачи докладов и администрирования может быть перенесена на другую конференцию с минимальными изменениями, что экономит время и ресурсы.

Еще одним преимуществом является экономичность. Монолитная архитектура требует меньше инфраструктурных ресурсов и затрат на обслуживание по сравнению с микросервисами, которые предполагают сложную настройку серверов, балансировщиков нагрузки и систем мониторинга. Для студенческого проекта, где бюджет ограничен, это делает монолитную архитектуру предпочтительным выбором.

Таким образом, монолитная архитектура предпочтительнее в реализации курсового проекта, так как минимизирует сложность и затраты, обеспечивая достаточную производительность для задач конференции. Микросервисная архитектура, хотя и предлагает гибкость для крупных систем, была бы избыточной и усложнила бы разработку без явных преимуществ в данном контексте.

## 1.2.2 Фреймворк и язык программирования

Для разработки серверной части выбран язык программирования C#, который сочетает производительность, читаемость и гибкость, делая его подходящим для веб-приложений. C# поддерживает объектно-ориентированное программирование, что упрощает создание структурированных классов для таких сущностей, как участники, доклады или квитанции об оплате. Его высокая степень безопасности типов помогает выявлять ошибки на этапе компиляции, что снижает вероятность багов в продакшене. Также C# позволяет строго типизировать входные данные, предотвращая ошибки, связанные с некорректными форматами.

C# тесно интегрирован с экосистемой .NET, предоставляя доступ к обширным библиотекам и инструментам, необходимым для реализации функций, таких как работа с базой данных, обработка файлов и обеспечение безопасности. C# зарекомендовал себя в веб-разработке, благодаря высокой производительности и поддержки кросс-платформенной разработки, что позволяет развертывать приложение на Windows, Linux или macOS. Это дает гибкость при выборе серверной инфраструктуры, что может быть полезно при дальнейшем масштабировании платформы.

C# также отличается читаемым синтаксисом, что упрощает написание и поддержку кода, особенно для разработчиков, знакомых с объектно-ориентированным программированием. Активное сообщество и обширная документация обеспечивают доступ к ресурсам, что упрощает решение задач, таких как автоматизация статусов заявок или интеграция с внешними сервисами. Высокая популярность C#, подтвержденная его рейтингом в индексах, таких как GitHub, делает его надежным выбором для вашего проекта.

Для реализации веб-приложения выбран фреймворк ASP.NET Core, актуальный для версии 9.0 на май 2025 года. ASP.NET Core отличается модульной архитектурой, позволяющей включать только необходимые компоненты, что снижает нагрузку на сервер и повышает эффективность приложения [3]. Это особенно полезно для проекта, где не требуется полный набор функций традиционного фреймворка, а приоритет отдается легкости и скорости.

Высокая производительность ASP.NET Core, обусловленная оптимизацией обработки HTTP-запросов, обеспечивает быструю работу платформы даже при пиковых нагрузках, например, во время массовой подачи тезисов. Фреймворк часто занимает лидирующие позиции в независимых бенчмарках, таких как TechEmpower, что подтверждает его способность эффективно обрабатывать запросы. Это критично для обеспечения удобства пользователей, особенно при выполнении операций, требующих быстрого отклика, таких как просмотр статуса заявки или загрузка файлов.

Безопасность – еще одно ключевое преимущество ASP.NET Core. Фреймворк включает встроенные механизмы аутентификации и авторизации, а также защиту от распространенных веб-угроз, таких как межсайтовый скриптинг (XSS) и межсайтовая подделка запросов (CSRF). Для сайта конференции, где обрабатываются конфиденциальные данные, такие как персональная информация участников, тезисы докладов и квитанции об оплате, эти функции безопасности неоценимы. Например, ASP.NET Core позволяет настроить OAuth или JWT для безопасной аутентификации пользователей, а также использовать политики авторизации для ограничения доступа администраторов к определенным функциям.

ASP.NET Core поддерживает кросс-платформенность, что соответствует возможностям C#, и предоставляет гибкие инструменты для структурирования приложения, включая шаблон MVC и Razor Pages, а также Razor Views [4].

## 1.2.3 MVC

Архитектура приложения основана на шаблоне MVC (Model-View-Controller), который разделяет приложение на три взаимосвязанных компонента, каждый из которых выполняет четко определенную функцию. Это разделение делает код более организованным, легким для понимания и поддержки, что особенно важно для проекта с множеством функциональных требований, таких как регистрация, подача докладов, модерация и управление расписанием [5].

*Модель* в приложении отвечает за представление и управление данными, а также за реализацию бизнес-логики. В контексте конференционного сайта это значит, что модель хранит и обрабатывает информацию об участниках, их заявках на доклады, статусах подтверждений, квитанциях об оплате и расписании мероприятий. Она обеспечивает валидацию вводимых данных и взаимодействует с базой данных для сохранения или извлечения нужной информации, гарантируя целостность и корректность бизнес-правил.

*Представление* формирует пользовательский интерфейс и отвечает за визуализацию данных, полученных от модели. В ASP .NET Core для этого используются Razor-шаблоны, которые позволяют динамически генерировать HTML-страницы на основе переданных контроллерами данных. Благодаря такому подходу интерфейс остаётся гибким и легко настраиваемым без прямого вмешательства в логику работы приложения.

*Контроллеры* выступают связующим звеном между моделью и представлением: они обрабатывают входящие HTTP-запросы, обращаются к методам модели для получения или обновления данных и затем выбирают соответствующее представление для отображения результата пользователю. Таким образом, контроллеры обеспечивают маршрутизацию запросов, управление потоком данных и подготовку модели к рендерингу, сохраняя чёткое разделение ответственности между компонентами MVC.

Процесс обработки запроса в MVC включает в себя несколько этапов:

1. *Получение запроса:* Пользователь отправляет HTTP-запрос, например, открывает страницу для выгрузки доклада через URL (например, */Reports/Add*) или отправляет форму с данными.
2. *Маршрутизация:* Система маршрутизации ASP.NET Core анализирует URL и направляет запрос к соответствующему контроллеру и методу действия. Например, запрос */Reports/Add* может быть направлен к методу *Add* в *ReportController*.
3. *Обработка в контроллере:* Контроллер выполняет необходимую логику, взаимодействуя с моделью. Например, при подаче доклада контроллер проверяет валидность данных (формата файла, размера), вызывает модель для сохранения тезисов в базе данных и обновляет статус заявки, обычно это логика реализуется через сервисы.
4. *Передача данных в представление:* Контроллер передает данные (например, сообщение об успешной подаче) в представление, которое рендерит HTML-страницу. Представление использует Razor-синтаксис для динамического отображения данных, таких как подтверждение подачи или список текущих заявок.
5. *Возврат ответа:* Сформированная HTML-страница возвращается пользователю через HTTP-ответ, завершая цикл обработки запроса.

MVC подходит для проекта, так как четко разделяет модель, представление и контроллер, упрощая разработку и поддержку сложных функций, таких как управление заявками и модерация. Это позволяет повторно использовать модели и контроллеры, например, применять модель User для регистрации и профиля, а также тестировать компоненты независимо, повышая качество кода. Гибкость MVC обеспечивает разные интерфейсы для администраторов и участников, а модульная структура облегчает исправление ошибок и добавление новых возможностей, вроде передачи докладов, без изменений в представлениях. Интеграция с ASP.NET Core и его инструментами, такими как маршрутизация и Razor, делает разработку удобнее. Эти преимущества делают MVC отличным выбором для структурированного и адаптируемого приложения.

1.2.4 Чистая архитектура: что это такое, преимущества и применение в проекте

Чистая архитектура – это философия проектирования программного обеспечения, предложенная Робертом Мартином, которая подчеркивает разделение системы на слои, с бизнес-логикой в центре, независимой от внешних факторов, таких как базы данных, пользовательские интерфейсы или фреймворки.

Основная цель чистой архитектуры – создать систему, которая является гибкой, тестируемой и легко поддерживаемой, что достигается за счет строгого соблюдения правила зависимостей: зависимости должны быть направлены внутрь, к ядру системы, а не наружу [7].

Согласно Мартину, приложение организуется в виде нескольких концентрических кругов, что показано на рисунке 1, где зависимости всегда направлены внутрь, к самому устойчивому и независимому слою.



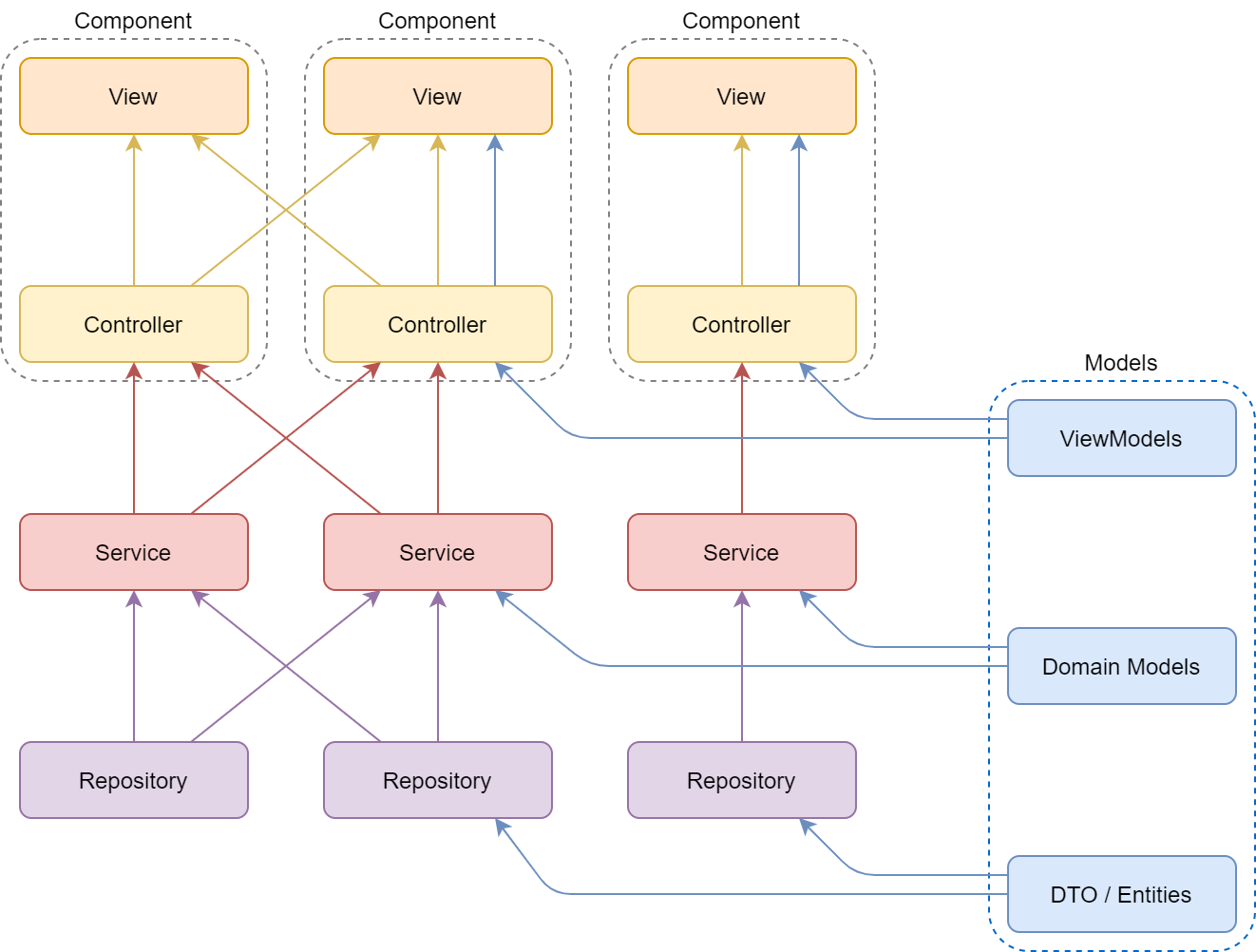
**Рисунок 1 – Диаграмма зависимостей**

В центре находятся сущности – доменные модели и бизнес‑правила, лишённые каких‑либо упоминаний о внешних технологиях. Вокруг них располагаются сценарии – классы, реализующие конкретные сценарии работы системы (регистрация участника, подача доклада, подтверждение оплаты, передача доклада другому пользователю и т. д.). Эти сервисы оперируют сущностями домена и определяют логику взаимодействия между ними, но не знают ничего ни о базе данных, ни о веб‑фреймворке. На следующем уровне располагаются интерфейс адаптеры или Контроллеры, они преобразуют входящие HTTP‑запросы в вызовы сервисов и наоборот формируют ответы для представлений. Внешний, последний круг составляют конкретные технологии: файловое хранилище, фреймворки и т.д., которые подключаются к системе через абстракции адаптеров.

На схеме взаимодействия на рисунке 2 показаны четыре уровня приложения:

1. *Контроллеры* и представления: располагаются в верхнем слое и отвечают только за приём HTTP‑запросов, вызов нужных сервисов и возврат готовых моделей для отображения в представления.
2. *Сервисы* инкапсулируют всю бизнес‑логику: они не знают ни о веб‑фреймворке, ни о БД, а работают исключительно с доменными моделями и абстракциями репозиториев.
3. *Репозитории* представляют сервисам доступ к данным (операции чтения, записи и т.д.) через интерфейсы, реализация зависит от технологий.
4. *Модели*разделены на три вида:

* *DTO/Entities* – простые переносимые объекты данных, которыми оперируют репозитории;
* *Domain Models* – объекты предметной области с бизнес‑правилами;
* ViewModels – модели, специально подготовленные для отображения.

**

**Рисунок 2 – Компонентная схема взаимодействия MVC‑контроллеров, сервисов и репозиториев в «чистой» архитектуре**

Стрелки показывают направление зависимостей: контроллеры зависят от сервисов, сервисы – от репозиториев, репозитории – от DTO/Entities. ViewModels текут от контроллера в представление, Domain Models – внутри сервисов, а DTO/Entities – внутри репозиториев.

Такая структура обеспечивает независимость бизнес-логики от технологий, что делает систему более устойчивой к изменениям и легкой для тестирования.

Эта структура обеспечивает независимость бизнес-логики от технологий. Например, если потребуется заменить базу данных SQL Server на PostgreSQL, изменения затронут только слой репозиториев, а случаи использования и сущности останутся неизменными. Аналогично, если в будущем потребуется создать мобильное приложение для конференции, бизнес-логика может быть повторно использована без изменений, добавив только новый интерфейсный слой.

Кроме того, чистая архитектура поддерживает требование ТЗ о возможности повторного использования платформы для других конференций. Поскольку бизнес-логика изолирована, ее можно легко адаптировать для новых мероприятий, изменив только внешние слои, такие как пользовательский интерфейс или конфигурацию базы данных. Это делает проект более устойчивым к будущим изменениям и экономичным в долгосрочной перспективе.

**ГЛАВА 2. ВЗАИМОДЕЙСТВИЯ С ВНЕШНИМ МИРОМ**

**2.1 Взаимодействие с пользователями**

Взаимодействие пользователей с веб‑платформой организовано через последовательную цепочку интерфейсных и серверных компонентов, обеспечивающих как ввод информации, так и её обработку, хранение и обратную отдачу пользователю.

Пользовательский опыт начинается с визуального интерфейса – браузерной формы, через которую участник или администратор выполняет действия: регистрация, вход, редактирование данных, подача доклада, загрузка документов и другие. Каждое такое действие сопровождается HTTP‑запросом к серверу, который принимает данные, проверяет их корректность и обрабатывает в соответствии с внутренней логикой приложения.

Обработка запроса разделена на несколько уровней: сначала запрос принимается системой маршрутизации и направляется в соответствующий контроллер, который выступает промежуточным звеном между пользовательским интерфейсом и логикой приложения. Контроллер взаимодействует с сервисами, где сосредоточена основная бизнес‑логика – сценарии, описывающие, что именно нужно сделать в ответ на то или иное действие пользователя. Эти сервисы, в свою очередь, обращаются к репозиториям, которые обеспечивают доступ к хранимым данным (например, в базе данных или файловой системе) [7].

После выполнения операции пользователь получает ответ – либо страницу с обновлённой информацией, либо уведомление о результате (например, об ошибке ввода или об успешной операции). Таким образом, взаимодействие представляет собой двусторонний процесс: пользователь инициирует действия, а система реагирует на них, проверяя, обрабатывая и возвращая информацию в понятном и визуально оформленном виде [8].

Такая структура позволяет организовать чёткую и управляемую модель взаимодействия: данные проходят сквозь строго определённые уровни, каждый из которых отвечает за свою зону ответственности. Благодаря этому достигается как техническая надёжность (например, защита от недопустимых входных данных), так и удобство использования с точки зрения самого участника конференции.

**2.2 Интеграция со сторонними сервисами**

Веб-приложение для научной конференции «Квантовая электроника 2025» обменивается данными с внешним миром через несколько ключевых интерфейсов, которые обеспечивают его функциональность и соответствие требованиям безопасности. На этапе разработки и тестирования используются доступные решения, такие как Gmail для отправки уведомлений, однако в продакшене планируется переход на более специализированные сервисы, интегрированные с университетской инфраструктурой.

Основные интерфейсы включают SMTP-сервер для уведомлений и двухфакторной аутентификации, файловое хранилище для фотографий и чеков, а также систему управления базами данных PostgreSQL. В будущем могут быть добавлены дополнительные внешние службы, такие как платежные системы или аналитические платформы, в зависимости от потребностей проекта.

2.2.1 SMTP-сервер

Для отправки уведомлений участникам конференции и кодов двухфакторной аутентификации (2FA) в настоящее время применяется SMTP-сервер Gmail. Это решение удобно для тестирования благодаря простоте настройки и широкой доступности, что позволяет быстро проверить функциональность уведомлений и аутентификации [9]. Однако в продакшене, когда система будет развернута на сервере Белорусского государственного университета (BSU.by), планируется использовать университетский SMTP-сервер. Такой переход обусловлен необходимостью соответствия локальным стандартам безопасности и интеграции с существующей почтовой инфраструктурой университета. Это обеспечит централизованное управление учетными записями и минимизирует риски утечки данных, что особенно важно для системы, обрабатывающей персональные данные класса 3-ин.

2.2.2 Файловое хранилище

Фотографии участников и чеки об оплате хранятся непосредственно на сервере, а пути к этим файлам записываются в базу данных. Такой подход выбран для упрощения управления данными и обеспечения их доступности в рамках приложения. Хранение файлов на сервере позволяет контролировать доступ на уровне файловой системы и выполнять резервное копирование в соответствии с требованиями безопасности, тогда как пути в базе данных упрощают их привязку к профилям участников или платежам. Это решение исключает зависимость от внешних облачных сервисов, что повышает автономность системы и соответствие университетским стандартам.

2.2.3 PostgreSQL

В качестве системы управления базами данных используется PostgreSQL, что было обусловлено тщательным анализом её возможностей по сравнению с альтернативными решениями, такими как MySQL. PostgreSQL выделяется своей поддержкой сложных запросов и транзакций, что важно для обработки данных о множестве участников, их научных докладах и финансовых операциях, записанных в чеках. Согласно официальной документации PostgreSQL, эта система обеспечивает высокую производительность при работе с большими объемами данных и сложной логикой, что делает её подходящей для высоконагруженных веб-приложений [10].

Кроме того, PostgreSQL поддерживает расширенные типы данных, такие как JSON, что позволяет эффективно хранить структурированные данные докладов, а также предлагает мощные механизмы масштабирования и партиционирования, что критично для системы, подключенной к интернету. Также в литературе пдчеркивается, что PostgreSQL превосходит MySQL в управлении параллельными соединениями и обеспечении целостности данных, что важно для академических проектов с высокой нагрузкой [11]. По сравнению с MySQL, которая проще в начальной настройке и быстрее для базовых операций, PostgreSQL предлагает более высокую устойчивость и гибкость, особенно в условиях роста данных и требований безопасности, что делает её предпочтительной для развертывания на серверe.

**ГЛАВА 3. БЕЗОПАСНОСТЬ**

В качестве средства для реализации безопасности пользовательской аутентификации и авторизации был выбран фреймворк ASP.NET Core Identity. Этот выбор обусловлен его способностью обеспечивать надежные механизмы защиты данных пользователей, простотой интеграции с cookie-based аутентификацией и поддержкой стандартных сценариев управления доступом. В данной главе мы подробно рассмотрим, как Identity реализует аутентификацию и авторизацию, с акцентом на использование cookie, а также объясним причины выбора этого инструмента.

**3.1 Аутентификация и авторизация пользователей. ASP.NET Core Identity**

ASP.NET Core Identity представляет собой встроенную систему управления идентификацией, разработанную Microsoft для платформы ASP.NET Core. Этот фреймворк предоставляет комплексное решение для реализации механизмов аутентификации и авторизации в веб-приложениях. Система ASP.NET Core Identity предлагает разработчикам широкий набор функциональных возможностей. Согласно официальной документации Microsoft, фреймворк поддерживает все основные сценарии работы с пользовательскими учетными записями: регистрацию новых пользователей, вход в систему, восстановление паролей, управление профилями. Особое внимание уделено вопросам безопасности - реализована двухфакторная аутентифиция (2FA) с поддержкой различных методов подтверждения личности, включая SMS-сообщения, электронную почту и специализированные приложения-аутентификаторы [12].

Главным преимуществом ASP.NET Core Identity является его глубокая интеграция с экосистемой ASP.NET Core. Как отмечается в документации, это обеспечивает согласованную работу всех компонентов фреймворка и значительно сокращает время настройки системы аутентификации. При создании нового проекта с опцией Individual User Accounts все необходимые компоненты Identity автоматически добавляются в решение вместе с предварительно подготовленными миграциями базы данных.

Вопросы безопасности в ASP.NET Core Identity решены на высоком уровне. Система использует современные криптографические алгоритмы для хранения паролей - PBKDF2 с HMAC-SHA256, что обеспечивает надежную защиту от атак методом перебора. Встроенные механизмы защиты от межсайтовой подделки запросов (CSRF) и межсайтового скриптинга (XSS) соответствуют последним рекомендациям OWASP. Дополнительный уровень безопасности обеспечивает гибкая система политик паролей, позволяющая настраивать требования к сложности (минимальная длина, использование специальных символов, цифр и т.д.) через класс IdentityOptions [13].

Важной особенностью ASP.NET Core Identity является его высокая степень настраиваемости. Разработчики могут расширять стандартную модель пользователя, добавляя новые свойства через наследование от базового класса IdentityUser. Система ролей и утверждений (claims) предоставляет мощный инструментарий для реализации сложных сценариев авторизации. Внешний вид стандартных форм аутентификации может быть полностью переработан в соответствии с дизайном конкретного приложения [13].

Для работы с внешними системами аутентификации Identity поддерживает протоколы OAuth 2.0 и OpenID Connect. Это позволяет реализовать вход через популярные платформы (Google, Microsoft, Facebook и другие) без необходимости разработки собственных интеграций. Поддержка Azure Active Directory делает фреймворк пригодным для корпоративных решений.

ASP.NET Core Identity демонстрирует высокую производительность и способность масштабироваться для работы с большими нагрузками. Архитектура фреймворка позволяет использовать распределенные системы и балансировку нагрузки. Интеграция с облачными сервисами Microsoft (в частности, с Azure Active Directory) открывает возможности для построения масштабируемых корпоративных решений.

Еще одним преимуществом Identity является регулярные обновления и долгосрочная совместимость с новыми версиями .NET. Фреймворк, как официальный продукт Microsoft обладает подробной документацией, доступной на нескольких языках, включая русский. А большое сообщество разработчиков предоставляет множество примеров и готовых решений распространенных проблем на таких площадках, как Stack Overflow и GitHub.

ASP.NET Core Identity представляет собой мощный и гибкий фреймворк для управления идентификацией в веб-приложениях. Его ключевые преимущества - глубокая интеграция с экосистемой ASP.NET Core, надежные механизмы безопасности, простота настройки и расширения. Широкая поддержка различных сценариев аутентификации, от базового ввода логина/пароля до сложных корпоративных решений, делает этот инструмент универсальным выбором для разработчиков.

Система особенно хорошо подходит для проектов, где важны безопасность и масштабируемость. Богатая документация и активное сообщество значительно упрощают процесс освоения технологии и решения возникающих проблем. Регулярные обновления и тесная интеграциввя с другими продуктами Microsoft обеспечивают долгосрочную перспективу использования ASP.NET Core Identity в профессиональной разработке.

**3.2 Ролевая модель доступа и авторизация на основе cookie**

Хотя Identity позволяет делать авторизацию на cookie, JWT-токенах и смешанный способ, для проекта была выбрана на основе cookie как стандартный механизм для веб-приложений. Этот подход предполагает хранение информации об аутентификации в зашифрованном cookie, который сохраняется в браузере пользователя и проверяется сервером при каждом запросе. Настройки cookie, такие как атрибуты HttpOnly, Secure и SameSite, обеспечивают защиту от распространенных веб-уязвимостей, таких как межсайтовый скриптинг (XSS) и подделка межсайтовых запросов (CSRF). Для приложения конференции, где пользователи, включая участников и администраторов, должны поддерживать сессии для выполнения таких задач, как управление профилем, подача научных работ и административные функции, авторизация на основе cookie необходима. Она гарантирует безопасное управление сессиями, позволяя пользователям беспрепятственно взаимодействовать с приложением в рамках одной сессии.

Также встроенный фреймворк поддерживает несколько моделей доступа: политики, утверждения, роли. Для учебного проекта была выбрана модель ролей. Ролевой контроль доступа дополняет аутентификацию, предоставляя возможность назначать пользователям определенные роли, такие как «Участник», «Докладчик» или «Администратор», каждая из которых имеет свои права доступа. Например, только администраторы могут утверждать поданные работы, тогда как участники имеют доступ только к своим профилям и материалам. Такой детализированный контроль критически важен для системы управления конференцией, где различные группы пользователей требуют разного уровня доступа к конфиденциальным данным [14].

Выбор ASP.NET Core Identity для данного проекта обоснован несколькими ключевыми факторами. Во-первых, система предоставляет надежные функции безопасности, включая безопасное хранение паролей, поддержку двухфакторной аутентификации и защиту от распространенных веб-атак, что крайне важно для защиты персональных и финансовых данных. Во-вторых, тесная интеграция с экосистемой ASP.NET Core снижает сложность разработки и упрощает сопровождение приложения. В-третьих, гибкость фреймворка позволяет настраивать модели пользователей и управление ролями в соответствии с конкретными потребностями приложения конференции.

ASP.NET Core Identity упрощает реализацию RBAC благодаря встроенным классам, таким как RoleManager и UserManager, которые позволяют создавать, назначать и проверять роли. Разработчики могут использовать атрибут [Authorize(Roles = "Administrator")] для ограничения доступа к определенным контр оллерам или действиям, а также определять пользовательские политики для более сложных сценариев авторизации. Эта гибкость позволяет адаптировать систему к специфическим требованиям конференции, обеспечивая безопасность и удобство управления доступом.

Интеграция ролевого контроля доступа и авторизации на основе cookie. Identity представляет собой безопасное, эффективное и соответствующее нормативным требованиям решение. Авторизация на основе cookie обеспечивает удобное управление сессиями, позволяя пользователям оставаться аутентифицированными в течение определенного времени без необходимости повторного входа. Ролевой контроль доступа упрощает взаимодействие с приложением, предоставляя пользователям доступ только к разрешенным функциям, что снижает вероятность ошибок и повышает удобство использования.

**3.3 Законодательные требования к системе**

В соответствии с приказом Оперативно-аналитического центра при Президенте Республики Беларусь от 12 ноября 2021 г. № 195, система, используемая для веб-приложения конференции «Квантовая электроника 2025», классифицируется как 3-ин, поскольку она обрабатывает персональные данные, включая фамилии, имена, отчества, адреса электронной почты и чеки об оплате, а также подключена к открытым каналам передачи данных через интернет [15].

Данный класс информационных систем требует строгого соблюдения нормативных требований по обеспечению безопасности информации, установленных законодательством Республики Беларусь, в частности Законом № 99-З «О защите персональных данных» и указанным приказом.

Система должна подвергаться регулярному аудиту безопасности для проверки соответствия установленным стандартам, при этом необходимо организовать сбор и хранение данных о событиях безопасности, включая информацию о функционировании вычислительной техники, сетевого оборудования и средств защиты, с сохранением таких данных не менее одного года. При необходимости может быть внедрен централизованный механизм сбора и хранения этих данных, а также мониторинг работы оборудования и средств защит [16].

Важным аспектом является защита данных, что включает регулирование использования съемных носителей информации и мобильных устройств с контролем их применения, а также обеспечение работоспособности оборудования, программного обеспечения и средств защиты через регулярные проверки. Резервные копии и настройки системы должны быть защищены от несанкционированного доступа, что подразумевает их шифрование и изоляцию.

Для управления доступом требуется разграничение прав пользователей к вычислительной технике, сетевому оборудованию, системному программному обеспечению и средствам защиты, а также идентификация и аутентификация пользователей с применением скрытия обратной связи при вводе аутентификационных данных, таких как пароли. Учетные записи должны находиться под полномочным контролем с возможностью их создания, активации, блокировки и удаления, при этом должны соблюдаться правила генерации и смены паролей, а также может быть предусмотрено централизованное управление учетными записями и паролями. Автоматическая блокировка доступа после периода неактивности или по запросу пользователя также является обязательной мерой.

Система защиты информации должна включать изменение стандартных настроек сетевого оборудования, системного программного обеспечения и средств защиты, постоянное наблюдение за состоянием объектов системы, контроль физического доступа к помещениям, где размещены компоненты, а также синхронизацию времени для согласованности временных меток.

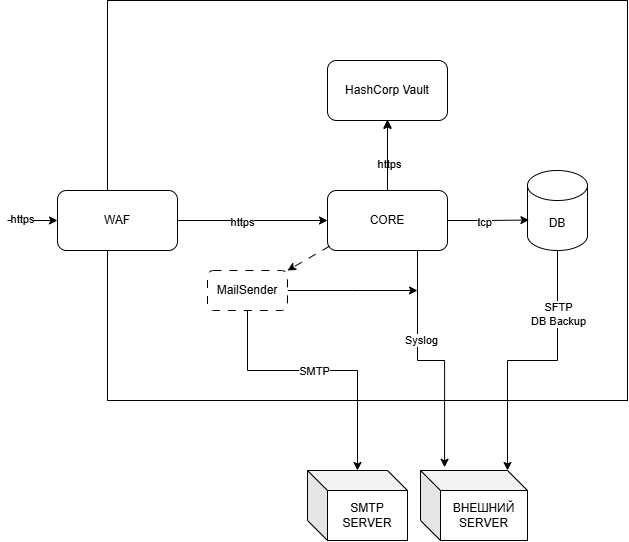
Передача данных по открытым каналам должна быть защищена с обеспечением конфиденциальности, целостности и недоступности информации, включая аутентификацию субъектов доступа и защиту от атак «отказ в обслуживании» и несанкционированного доступа.

Криптографическая защита является ключевым элементом, что требует использования сертифицированных средств шифрования для передачи, хранения и обработки данных, а также физической и логической защиты этих средств.

Дополнительные меры включают защиту от вредоносных программ, спама, фишинга, социальной инженерии и инсайдерских угроз, создание и хранение резервных копий, регулярное обновление программного обеспечения, а также, при необходимости, сегментирование сети, использование межсетевых экранов и внедрение систем обнаружения и предотвращения вторжений, включая беспроводные сети. Таким образом, соблюдение всех указанных требований обеспечивает высокий уровень безопасности системы, необходимый для защиты персональных данных участников конференции в соответствии с законодательством Республики Беларусь [8].

**3.4 Архитектура безопасного приложения**

В условиях постоянно возрастающих требований к безопасности информационных систем крайне важно проектировать архитектуру приложений с учётом потенциальных угроз и уязвимостей. Современные веб-приложения обрабатывают конфиденциальные данные, взаимодействуют с внешними сервисами и подвержены различным видам атак – от банальных сканирований до целевых вторжений. Поэтому безопасность должна быть заложена в основу ещё на этапе проектирования.



**Рисунок 3 – Схема безопасной архитектуры**

На рисунке 3 представлена безопасная архитектура приложения, включающая в себя защищённый внутренний контур, в котором функционируют основные компоненты системы, и внешний мир, из которого осуществляется доступ к приложению. Архитектура разработана с учётом принципов безопасности, надёжности и отказоустойчивости, и предусматривает разграничение доступа, шифрование данных, централизованное логирование и резервное копирование.

Входящий трафик из интернета сначала поступает на Web Application Firewall (WAF). Это может быть как аппаратный WAF, когда фильтрация трафика происходит на уровне инфраструктуры (например, через облачного провайдера), так и программный, развёрнутый непосредственно в рамках приложения. WAF выполняет первичную фильтрацию запросов, блокируя потенциально опасные действия – такие как SQL-инъекции, попытки обхода авторизации, XSS-атаки и другие угрозы. Это критически важный компонент, поскольку он является первой линией обороны, стоящей между внешним миром и внутренней логикой приложения.

После WAF трафик поступает в основной модуль приложения – компонент, обозначенный как CORE. Это ядро всей бизнес-логики, которое принимает, обрабатывает и маршрутизирует запросы пользователей. Для взаимодействия с базой данных (DB) CORE использует TCP-соединения, предполагая защищённое внутреннее сетевое окружение. Данные, передаваемые между CORE и DB, могут быть как в зашифрованном виде, так и в виде защищённых соединений (например, через TLS), в зависимости от конфигурации внутренней сети и требований безопасности.

Для хранения чувствительных данных – таких как строки подключения к базе данных, пароли, токены, API-ключи и прочие секреты – используется HashiCorp Vault. Это специализированное решение для безопасного хранения секретов, обеспечивающее централизованное управление доступом, шифрование данных и аудит. Доступ к Vault осуществляется по HTTPS, что исключает возможность перехвата информации в процессе передачи. Vault интегрирован с CORE, что позволяет динамически получать необходимые секреты в процессе выполнения, не храня их в открытом виде в конфигурационных файлах.

Компонент MailSender, обозначенный пунктиром, вероятно представляет собой дополнительный модуль, вызываемый при необходимости из CORE. Он отвечает за отправку уведомлений по электронной почте. Взаимодействие с внешним SMTP-сервером также происходит через защищённое соединение, обеспечивая конфиденциальность и целостность передаваемой информации. Подобная архитектура позволяет реализовать сценарии оповещения администраторов о критических событиях или отправку писем пользователям (например, подтверждения регистрации).

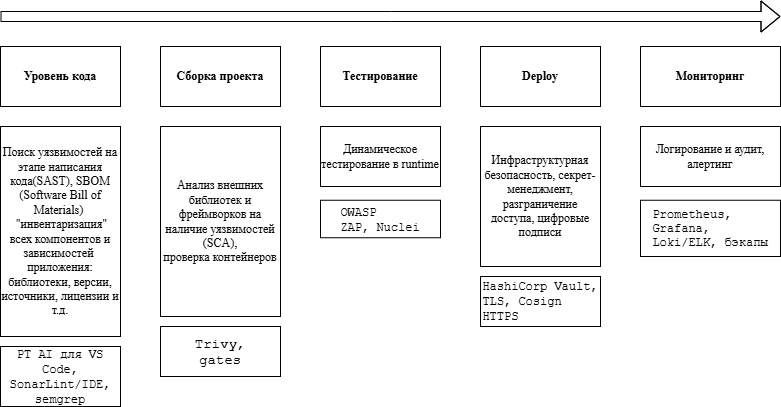
Логирование реализовано на всех этапах – как в CORE, так и в других компонентах. Логи передаются по протоколу Syslog на внешний лог-сервер. Это важная мера безопасности: если логирование происходит только внутри приложения, при компрометации системы злоумышленник может удалить или изменить логи. Внешний сервер логов позволяет сохранить достоверную информацию о действиях в системе и облегчает последующий анализ инцидентов.

Система резервного копирования базы данных реализована с использованием SFTP – защищённого протокола передачи файлов. Бэкапы сохраняются на внешнем сервере, что позволяет восстановить систему в случае потери данных, сбоев оборудования или атак (включая ransomware). Важно, чтобы бэкапы также шифровались и подписывались, чтобы исключить возможность их подделки или чтения третьими лицами.

Таким образом, представленная архитектура учитывает ключевые аспекты информационной безопасности: фильтрацию входящего трафика, разграничение доступа, защиту чувствительных данных, безопасную передачу информации, централизованное логирование и резервное копирование. Такая организация системы позволяет минимизировать риски, повысить устойчивость к атакам и обеспечить высокий уровень доверия к данным и работе приложения [17].

**3.5 Конвейер CI/CD обеспечения безопасности приложения**

В контексте разработки безопасного программного обеспечения был создан комплексный конвейер, включающий пять ключевых этапов: уровень кода, сборка проекта, тестирование, развертывание и мониторинг. Конвейер представлен на рисунке 4.

******

**Рисунок 4 – Конвейер обеспечения безопасности**

Каждый этап играет важную роль в обеспечении безопасности и надежности приложения, используя специализированные инструменты и процессы для выявления уязвимостей и соблюдения лучших практик. Конвейер особенно актуален для приложений, требующих строгих мер безопасности, где защита конфиденциальных данных и стабильность работы имеют первостепенное значение.

На этапе уровня кода основное внимание уделяется раннему обнаружению уязвимостей и управлению зависимостями в процессе написания кода. Здесь проводится статический анализ кода с использованием методов SAST для выявления недостатков безопасности до компиляции. Также формируется перечень программных компонентов (SBOM), включающий все зависимости и их версии, что упрощает своевременное обновление и отслеживание уязвимостей. Инструменты, такие как PT AI для VS Code, SonarLint и semgrep, интегрируют анализ безопасности непосредственно в среду разработки. PT AI для VS Code предоставляет данные о безопасности в реальном времени в Visual Studio Code, SonarLint повышает качество и безопасность кода через интеграцию с IDE, а semgrep помогает выявлять ошибки и уязвимости на ранних стадиях. Этот подход позволяет оперативно устранять проблемы, снижая риск утечек безопасности на последующих этапах.

Далее следует этап сборки проекта, где осуществляется компиляция и проверка зависимостей и контейнеров на наличие уязвимостей. Анализ состава программного обеспечения (SCA) используется для проверки внешних библиотек и фреймворков на известные дефекты безопасности. Также оценивается безопасность контейнеров на соответствие стандартам. Инструменты Trivy и Gates играют ключевую роль на этом этапе: Trivy сканирует контейнеры и зависимости на уязвимости, а Gates внедряет автоматические проверки качества, предотвращая продолжение сборки при несоблюдении критериев безопасности. Этот этап служит барьером, гарантируя, что в приложение интегрируются только безопасные компоненты.

На этапе тестирования акцент делается на выявлении уязвимостей, проявляющихся во время выполнения. Динамическое тестирование безопасности приложений (DAST) имитирует атаки, обнаруживая проблемы, которые статический анализ может упустить. Инструменты OWASP ZAP и Nuclei существенно помогают в этом процессе: OWASP ZAP автоматизирует тестирование веб-приложений на распространенные уязвимости, такие как SQL-инъекции и XSS, а Nuclei обеспечивает быстрое сканирование на известные уязвимости и ошибки конфигурации. Этот этап критически важен для повышения устойчивости приложения к реальным угрозам.

После успешного тестирования приложение переходит на этап развертывания, где оно переносится в производственную среду с акцентом на безопасность инфраструктуры. Основные процессы включают обеспечение безопасности инфраструктуры, управление секретами, контроль доступа и шифрование данных. Инструменты HashiCorp Vault, TLS/HTTPS и Cosign используются для этих целей: Vault безопасно управляет конфиденциальной информацией, TLS/HTTPS обеспечивает шифрование передаваемых данных, а Cosign подписывает контейнеры, подтверждая их целостность и подлинность. Этот этап минимизирует риски эксплуатации в производственной среде, соответствуя стандартам безопасного развертывания.

Завершающий этап, мониторинг, предполагает постоянное наблюдение за приложением после развертывания для поддержания стабильности и безопасности. Процессы включают ведение логов, аудит, оповещения и создание резервных копий для восстановления данных. Инструменты Prometheus, Grafana, Loki/ELK и системы резервного копирования поддерживают эти задачи: Prometheus собирает метрики производительности и данные о состоянии системы, Grafana визуализирует их в реальном времени, Loki или ELK управляют хранением и анализом логов, а резервные копии обеспечивают восстановление данных при сбоях или утечках. Этот этап необходим для долгосрочной операционной целостности и быстрого реагирования на угрозы [18].

Таким образом, описанный конвейер представляет собой структурированный подход к разработке безопасного программного обеспечения, интегрируя передовые инструменты и методы на всех этапах жизненного цикла приложения. От раннего анализа кода до мониторинга после развертывания каждый этап способствует созданию надежного и защищенного приложения, что особенно важно для проектов с чувствительной информацией.

**ГЛАВА 4. РАЗРАБОТКА**

В этой главе описаны практические аспекты реализации, включая архитектуру, технологии, интеграцию с внешними сервисами. Подробности теоретических оснований можно найти в предыдущих главах.

Приложение построено на основе монолитной архитектуры, что упрощает разработку и поддержку для проекта с ограниченным масштабом. Выбор монолитной архитектуры обусловлен необходимостью минимизировать сложность и затраты, как описано в разделе 1.2.1.

Для реализации серверной части был выбран язык C# и фреймворк ASP.NET Core 9.0. C# обеспечивает высокую производительность и безопасность типов, а ASP.NET Core предлагает модульную архитектуру и встроенные механизмы безопасности (см. пункт 1.2.2). Применен шаблон MVC, который разделяет приложение на модели, представления и контроллеры, упрощая управление кодом и его тестирование (см. пункт 1.2.3).

Проект организован в соответствии с принципами чистой архитектуры, что обеспечивает независимость бизнес-логики от внешних технологий и упрощает тестирование (см. пункт 1.2.4). Для взаимодействия с внешним миром были реализованы интеграции с несколькими сервисами, как описано в главе 2. Для отправки уведомлений и кодов двухфакторной аутентификации используется SMTP-сервер.

На этапе разработки применялся Gmail, но в продакшене планируется переход на университетский SMTP-сервер (см. пункт 2.2.1). Фотографии участников и чеки об оплате хранятся на сервере, а пути к файлам записываются в базу данных. Это решение упрощает управление данными и обеспечивает их безопасность (см. пункт 2.2.2).

В качестве системы управления базами данных выбрана PostgreSQL, интегрированная через Entity Framework Core с подходом Code First. Это обеспечивает гибкость и надежность хранения данных (см. пункт 2.2.3). Безопасность приложения реализована с учетом законодательных требований и лучших практик, как описано в главе 3. Использован фреймворк ASP.NET Core Identity для управления пользователями, ролями и доступом. Реализована аутентификация на основе cookie и ролевая модель доступа (см. разделы 3.1 и 3.2).

Для повышения безопасности внедрена двухэтапная аутентификация с использованием пароля и одноразового кода, отправляемого на электронную почту. Пароли пользователей хранятся в зашифрованном виде с использованием алгоритма HMAC-256. Применены механизмы защиты от атак CSRF и XSS, встроенные в ASP.NET Core. Разработанная система почти полностью соответствует требованиям приказа №195 от 12 ноября 2021 года для информационных систем класса 3-ин, не реализовано логирование и аудит, а их внедрение запланировано до выхода приложения в продакшен.

## **Трудности и проблемы, возникшие в процессе разработки**

Несмотря на продуманную архитектуру и выбор стабильных технологий, в ходе реализации проекта возникли ряд сложностей, которые потребовали дополнительных усилий для их преодоления.

### *1. Проблемы с конфигурацией SMTP и отправкой писем*

Одной из первых трудностей стала настройка почтового сервера для отправки уведомлений и кодов двухфакторной аутентификации. На этапе разработки использовался SMTP-сервер Gmail, однако даже при тестовом использовании потребовалось создать отдельный почтовый аккаунт, включить небезопасный доступ и сгенерировать пароль приложения. Кроме того, письма периодически попадали в спам, что создавало неудобства при тестировании. В будущем планируется переход на корпоративный SMTP-сервер, но его использование требует дополнительной настройки TLS и авторизации по сертификатам, что затрудняет отладку и требует согласования с IT-инфраструктурой университета.

### *2. Ошибки при миграции базы данных*

Поскольку в проекте применялся подход Code First с использованием Entity Framework Core, периодически возникали конфликты при создании миграций. Особенно это касалось изменений в навигационных свойствах и ограничениях внешних ключей. В некоторых случаях приходилось вручную редактировать сгенерированные миграции, что требует осторожности и понимания внутренней структуры базы данных. Одна из частых проблем – несогласованность миграций между ветками Git, что также приводило к необходимости их отката и пересоздания.

### *3. Трудности с разграничением доступа*

Хотя ASP.NET Core Identity предоставляет готовые механизмы аутентификации и авторизации, их расширение под нестандартные требования оказалось нетривиальной задачей. Необходимо было реализовать как простое разграничение ролей (администратор, участник, модератор), так и более сложные сценарии, например, передачу доклада от одного пользователя другому с сохранением авторства. Это потребовало внесения изменений в бизнес-логику и реализацию дополнительных проверок в контроллерах и сервисах. Были случаи, когда пользователи могли получить доступ к представлениям, не соответствующим их роли, из-за недостающих атрибутов или пропущенных проверок.

**ЗАКЛЮЧЕНИЕ**

Курсовая работа была посвящена разработке серверной части веб-приложения, предназначенного для сопровождения научной конференции «Квантовая электроника 2025». В процессе выполнения проекта была достигнута основная цель – создание минимально жизнеспособного программного продукта, обеспечивающего реализацию ключевых функций, необходимых для организации и проведения конференции. К ним относятся регистрация участников, подача и модерация докладов, управление расписанием мероприятия, подтверждение оплаты участия, а также обновление статусов заявок. Все эти процессы были объединены в единую систему, упрощающую взаимодействие участников и организаторов, минимизируя ручной труд и риски человеческого фактора.

Разработка велась с учётом современных требований к качеству программного обеспечения, включая удобство использования, отказоустойчивость и безопасность. В условиях растущей цифровизации всех сфер деятельности и увеличения числа онлайн-мероприятий особое внимание было уделено обеспечению защищённости пользовательских данных и соответствию актуальным нормативным требованиям. Принципы проектирования, заложенные в архитектуру системы, позволяют масштабировать и адаптировать её под нужды других мероприятий, сохраняя при этом целостность и устойчивость решения.

Практическая значимость выполненной работы заключается в создании функционального инструмента, пригодного для реального использования в рамках подготовки и проведения научной конференции. Реализованное веб-приложение способно заменить собой традиционные механизмы подачи заявок и взаимодействия участников, существенно упрощая организационные процессы. В результате значительно снижается административная нагрузка, ускоряется обработка информации, повышается прозрачность всех этапов работы с участниками, а также улучшается качество взаимодействия между ними и организационным комитетом.

Особую ценность представляет системный подход, реализованный в ходе работы над проектом. Он предполагает рассмотрение всех аспектов функционирования системы в комплексе: от пользовательского взаимодействия до организационно-правовых требований и потенциальных рисков. Такой подход позволил не только реализовать заданный функционал, но и заложить основы для безопасной и устойчивой эксплуатации приложения в реальных условиях. Проведённый анализ рисков, особенностей обработки персональных данных и требований регуляторов позволил сформулировать и внедрить обоснованные меры защиты, направленные на предотвращение несанкционированного доступа, утечек информации и других угроз.

Перспективы дальнейшего развития проекта включают не только расширение функциональности системы, но и её интеграцию с современными инструментами автоматизации разработки и поддержки. Планируется внедрение конвейера непрерывной интеграции и доставки, системы мониторинга и сбора метрик, а также более глубокая интеграция с существующей IT-инфраструктурой университета. Эти шаги направлены на повышение надёжности, масштабируемости и устойчивости системы, а также на упрощение процессов её сопровождения в долгосрочной перспективе. Благодаря заложенным архитектурным решениям, возможна постепенная модернизация и адаптация системы без кардинального переписывания кода или переработки бизнес-логики.

Таким образом, представленная курсовая работа демонстрирует успешное применение теоретических знаний на практике и служит примером комплексного подхода к разработке прикладного программного обеспечения. Созданное веб-приложение отвечает современным требованиям, обладает высоким уровнем функциональности и безопасности, а также открывает широкие возможности для использования и масштабирования в рамках научной и образовательной деятельности. Полученные результаты подтверждают целесообразность выбранных методов и подходов, а также подчеркивают важность включения вопросов информационной безопасности на всех этапах жизненного цикла программного обеспечения. В конечном итоге, данный проект вносит вклад в развитие цифровых технологий в научной сфере и может служить моделью для создания аналогичных решений в других областях.

**СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ**

1. Фаулер, М. Шаблоны корпоративных приложений (Patterns of Enterprise Application Architecture) / М. Фаулер. — М.: Addison-Wesley, 2002.
2. Ньюман, С. Переход от монолита к микросервисам (Building Microservices) / С. Ньюман. – М.: O’Reilly, 2015.
3. Microsoft Learn. Введение в .NET (Introduction to .NET) [Электронный ресурс]. URL: https://learn.microsoft.com/dotnet/introduction (дата обращения: 10.04.2025).
4. Эспозито, Д. Архитектура корпоративных приложений на платформе .NET (Microsoft .NET – Architecting Applications for the Enterprise) / Д. Эспозито, А. Салтарелло. – М.: Microsoft Press, 2014.
5. Фриман, А. Pro ASP.NET Core MVC / А. Фриман, М. Шарп. – М.: Apress, 2017–2023.
6. Мартин, Р. C. Чистая архитектура: Руководство ремесленника по структуре и проектированию ПО (Clean Architecture: A Craftsman’s Guide to Software Structure and Design) / Р. C. Мартин. – М.: Prentice Hall, 2017.
7. Microsoft Learn. Обработка запросов MVC в ASP.NET Core (Model-view-controller (MVC) request processing in ASP.NET Core) [Электронный ресурс]. URL: https://learn.microsoft.com/aspnet/core/mvc/controllers/actions?view=aspnetcore-9.0 (дата обращения: 07.05.2025).
8. Anderson, R. Архитектура и конвейер запросов ASP.NET Core MVC (ASP.NET Core MVC architecture and request pipeline) / R. Anderson // Microsoft Docs. 2021.
9. Microsoft Learn. Отправка электронной почты в ASP.NET Core (Send email in ASP.NET Core) [Электронный ресурс]. URL: https://learn.microsoft.com/aspnet/core/fundamentals/mime-mapping?view=aspnetcore-9.0 (дата обращения: 10.05.2025).
10. The PostgreSQL Global Development Group. Документация PostgreSQL 16.3 (PostgreSQL 16.3 Documentation) [Электронный ресурс]. URL: https://www.postgresql.org/docs/16/index.html (дата обращения: 10.05.2025).
11. Элмасри, Р. Системы баз данных: Полный курс (Database Systems: The Complete Book) / Р. Элмасри, С. Б. Навазе. – 2-е изд. – М.: Pearson, 2003.
12. Microsoft Learn. Введение в Identity на ASP.NET Core (Introduction to Identity on ASP.NET Core) [Электронный ресурс]. URL: https://learn.microsoft.com/aspnet/core/security/authentication/identity?view=aspnetcore-9.0 (дата обращения: 11.04.2025).
13. Граймс, Р. Разработка защищённых приложений на .NET (Developing Secure Applications with .NET) / Р. Граймс. – М.: Microsoft Press, 2020.
14. Microsoft Learn. Ролевая авторизация в ASP.NET Core (Role-based authorization in ASP.NET Core) [Электронный ресурс]. URL: https://learn.microsoft.com/aspnet/core/security/authorization/roles?view=aspnetcore-9.0 (дата обращения: 10.05.2025).
15. Приказ № 195 от 12 ноября 2021 г. «Об утверждении требований к обеспечению безопасности информационных систем третьего класса информационной надежности» / Оперативно-аналитический центр при Президенте Республики Беларусь. [Нормативный акт].
16. Республика Беларусь. Закон № 99-З от 7 ноября 2021 г. «О защите персональных данных» // Законодательство Республики Беларусь. – 2021.
17. Андресс, Дж. Основы информационной безопасности (The Basics of Information Security) / Дж. Андресс. – М.: Syngress, 2019.
18. Гилберт, Д. DevSecOps: Руководство для руководителей по безопасной разработке ПО (DevSecOps: A leader’s guide to producing secure software without compromising flow, feedback and learning) / Д. Гилберт, Дж. Хант. – М.: IT Revolution, 2020.