

Министерство образования Республики Беларусь

Учреждение образования  
«Белорусский государственный университет информатики и  
радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: ЖЦРПО

ОТЧЕТ  
по лабораторной работе № 3  
на тему  
ИССЛЕДОВАНИЕ АРХИТЕКТУРНОГО РЕШЕНИЯ

Студенты:

Ю.Л. Спасёнов  
И.К. Манько  
П.А. Щербо  
В.С. Вильчинский

Преподаватель:

Д.А. Жалейко

МИНСК 2025

# Часть 1. Проектирование архитектуры

## 1. Определение типа приложения

Разрабатываемая система представляет собой **веб-платформу для онлайн-обучения**. Она позволяет пользователям:

- Регистрироваться и проходить аутентификацию;
- Просматривать доступные курсы и записываться на них;
- Проходить уроки, выполнять тесты и отслеживать прогресс;
- Взаимодействовать с преподавателями и другими учениками.

Система реализована как **клиент-серверное веб-приложение**, доступное через современные веб-браузеры, что обеспечивает кроссплатформенность и широкий охват аудитории.

## 2. Выбор стратегии развёртывания

Наиболее подходящей стратегией развёртывания является модель **клиент-сервер**. При таком подходе:

- **Серверная часть** (на базе Django) отвечает за хранение и обработку данных, выполнение бизнес-логики и управление доступом к информации.
- **Клиентская часть** (реализованная с помощью HTML, CSS, JavaScript) обеспечивает интуитивно понятный графический интерфейс для пользователей, работающих через веб-браузер.

### Основные компоненты системы:







- **Клиентская часть:**
  - *Веб-браузер*: для доступа к платформе.
  - *Стабильное интернет-соединение*: необходимо для корректного взаимодействия с сервером.
- **Приложение:**
  - *Фронтенд*: технологии HTML, CSS и JavaScript используются для создания визуального интерфейса и интерактивных элементов.
  - *Бэкенд*: фреймворк Django для реализации серверной логики, обработки данных и управления бизнес-процессами.
- **Среда разработки:**
  - *PyCharm*: для разработки на Python.
  - *WebStorm*: для разработки клиентских частей на HTML, CSS и JavaScript.
  -

- **База данных:**

- *PostgreSQL*: реляционная СУБД, отвечающая за хранение и управление данными платформы.

Такой подход обеспечивает масштабируемость, централизованное управление данными и возможность быстрого обновления системы.

### 3. Обоснование выбора технологии

	<p><b>PyCharm</b> — это высокоэффективная интегрированная среда разработки (IDE) для Python, обеспечивающая полную поддержку языка и его фреймворков. Интуитивный интерфейс упрощает процесс разработки и повышает производительность.</p>
	<p><b>Python</b> — высокоуровневый современный язык программирования с динамической типизацией, известный своим чистым синтаксисом, облегчающим чтение и написание кода.</p>
	<p><b>Django</b> — это популярный фреймворк для разработки веб-приложений на языке Python. Он предоставляет набор инструментов и функций, упрощающих создание высокопроизводительных и масштабируемых веб-приложений.</p>
	<p><b>PostgreSQL</b> — это открытая система управления базами данных объектно-реляционного типа. PostgreSQL предлагает надежность, масштабируемость, гибкость и поддержку продвинутых SQL-стандартов. Эффективно работает с различными объемами данных и подходит для сложных запросов</p>
	<p><b>JavaScript</b> — это мультипарадигменный язык программирования, используемый для создания интерактивных web-страниц и элементов на них.</p>
	<p><b>CSS</b> — это формальный язык декорирования и описания внешнего вида web-страниц и документов</p>
	<p><b>HTML</b> — это стандартизированный язык гипертекстовой разметки документов для просмотра web-страниц в браузере</p>

#### **4. Указание показателей качества (методология FURPS)**

##### **Functionality (Функциональные требования):**

- Регистрация и аутентификация пользователей.
- Просмотр курсов и запись на выбранные курсы.
- Прохождение уроков, выполнение тестов, отслеживание прогресса.
- Выдача сертификатов по завершении курса.
- Поддержка мультимедийных материалов (видео, аудио, изображения).
- Возможность взаимодействия с преподавателями и другими учащимися.

##### **Usability (Удобство использования):**

- Интуитивно понятный и единообразный интерфейс.
- Быстрая и удобная навигация.
- Многоязычная поддержка (например, русский и английский).
- Минимальное количество кликов для доступа к ключевому функционалу.

##### **Reliability (Надёжность):**

- Высокая готовность системы (90–99% времени безотказной работы).
- Возможность восстановления данных в случае сбоев.
- Регулярное резервное копирование (не реже одного раза в сутки).
- Обеспечение отказоустойчивости: при отказе отдельных компонентов система продолжает работать с ограниченным функционалом.

##### **Performance (Производительность):**

- Средняя задержка отклика не превышает 2 секунд.
- Эффективное использование системных ресурсов.
- Поддержка одновременного доступа большого количества пользователей.
- Масштабируемость системы для роста нагрузки.

##### **Supportability (Сопровождаемость):**

- Возможность добавления нового функционала без значительных изменений существующего кода.
- Централизованное логирование и обработка ошибок.
- Автоматизированное резервное копирование.
- Адаптивный дизайн для корректного отображения на различных устройствах.

## 5. Обозначение путей реализации сквозной функциональности

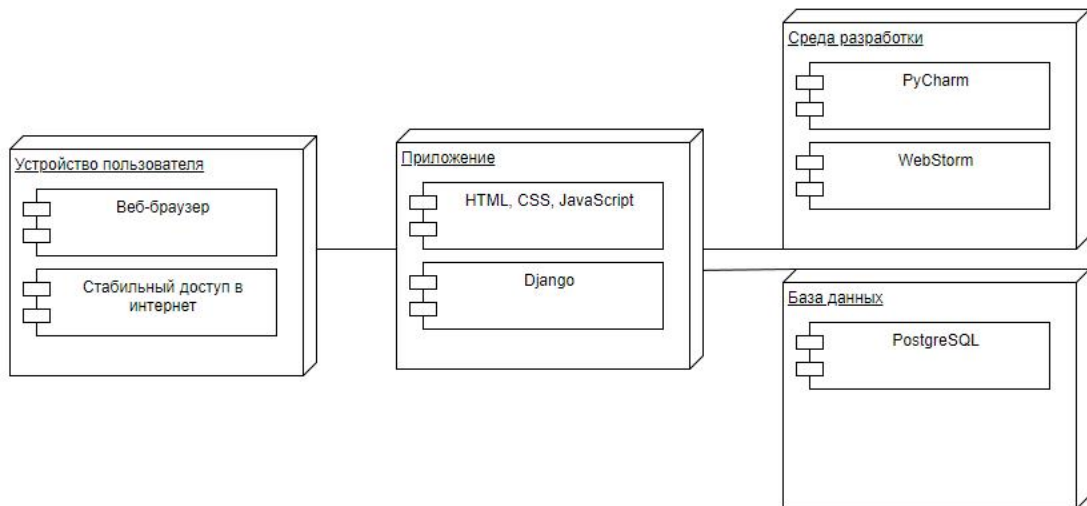
Сквозная функциональность охватывает аспекты, затрагивающие несколько модулей системы. Для их реализации будут использованы следующие подходы:

- **Аутентификация и авторизация:**  
Реализуются с помощью встроенных возможностей Django (например, использование JWT-токенов для REST API). Это обеспечивает централизованное управление доступом и безопасность.
- **Логирование и аудит:**  
Внедрение middleware для перехвата запросов и ошибок, использование специализированных сервисов (например, Sentry) для мониторинга и анализа логов.
- **Обработка исключений:**  
Единая система обработки ошибок, позволяющая централизованно регистрировать и обрабатывать исключительные ситуации, предотвращая утечки информации.
- **Кэширование:**  
Применение кэширования (с использованием Redis или Memcached) для снижения времени отклика и повышения производительности при работе с часто запрашиваемыми данными.
- **Безопасность:**  
Реализация мер защиты от CSRF, XSS, SQL-инъекций посредством встроенных механизмов Django и дополнительных модулей.
- **Международализация (i18n):**  
Использование стандартных средств Django для поддержки многоязычных интерфейсов, что позволяет легко адаптировать систему под различные языковые версии.

## 6. Изображение структурной схемы приложения

Ниже представлена структурная схема «Архитектуры To Ве», отображающая основные функциональные блоки и слои системы:

**Рис. 1. Структурная схема приложения (To Ве):**

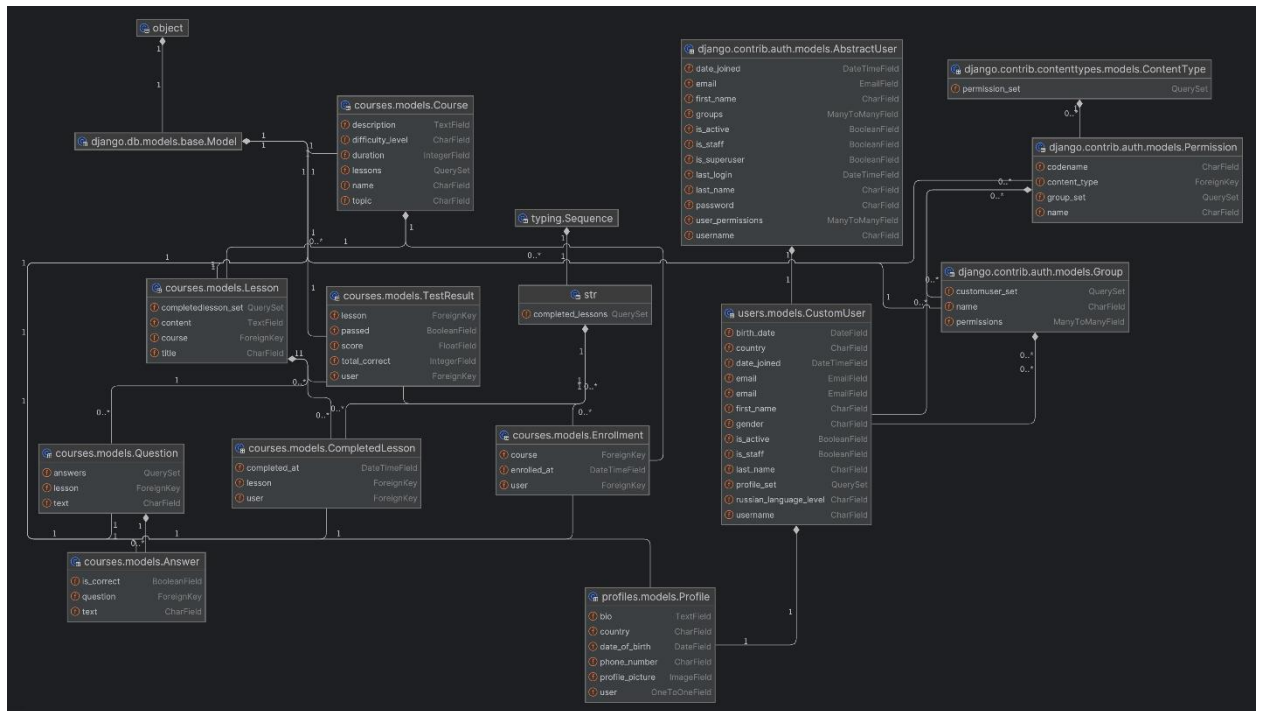


*Примечание:* Данная схема иллюстрирует основные слои приложения – от взаимодействия с пользователем до хранения данных.

## Часть 2. Анализ архитектуры («As is»)

На данном этапе проведён анализ существующей архитектуры системы, сформированной в ходе первого Sprint Review. С использованием инструмента обратной инженерии (например, IBM Rational Rose) была сгенерирована диаграмма классов, отображающая структуру ключевых компонентов.

Рис. 2. Диаграмма классов (As is):



*Примечание:* Диаграмма демонстрирует основные модели системы:

- **User** – отвечает за данные пользователей.
- **UserProfile** – хранит дополнительную информацию о пользователях.
- **Course** – представляет курсы, доступные на платформе.
- **Lesson** – содержит информацию об уроках, входящих в курсы.
- **Test и Question** – обеспечивают функциональность проведения тестирования.

Связи между классами отражают их отношения, выявленные в исходном коде.

### Часть 3. Сравнение и рефакторинг

Ниже представлены две диаграммы, отражающие текущее («As Is») и проектируемое («To Be») состояния системы обучения. Первая иллюстрирует целевой вариант архитектуры и процессов, вторая – текущее (фактическое) положение дел в уже разработанных модулях.

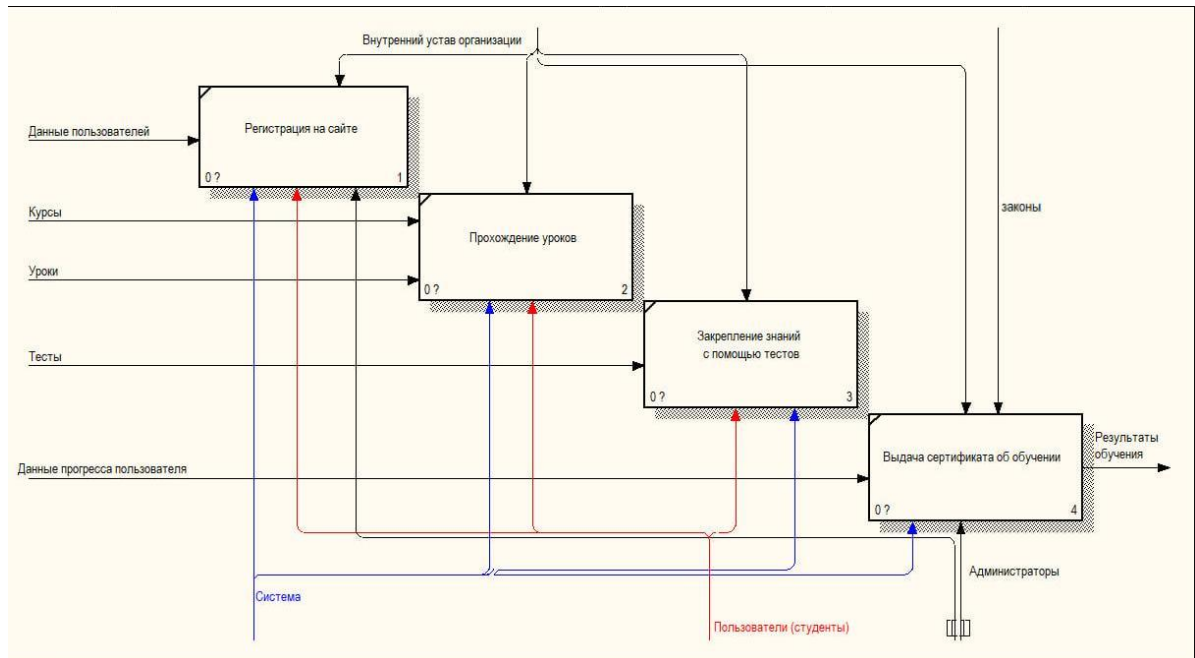


Диаграмма IDEF0 «As Is»

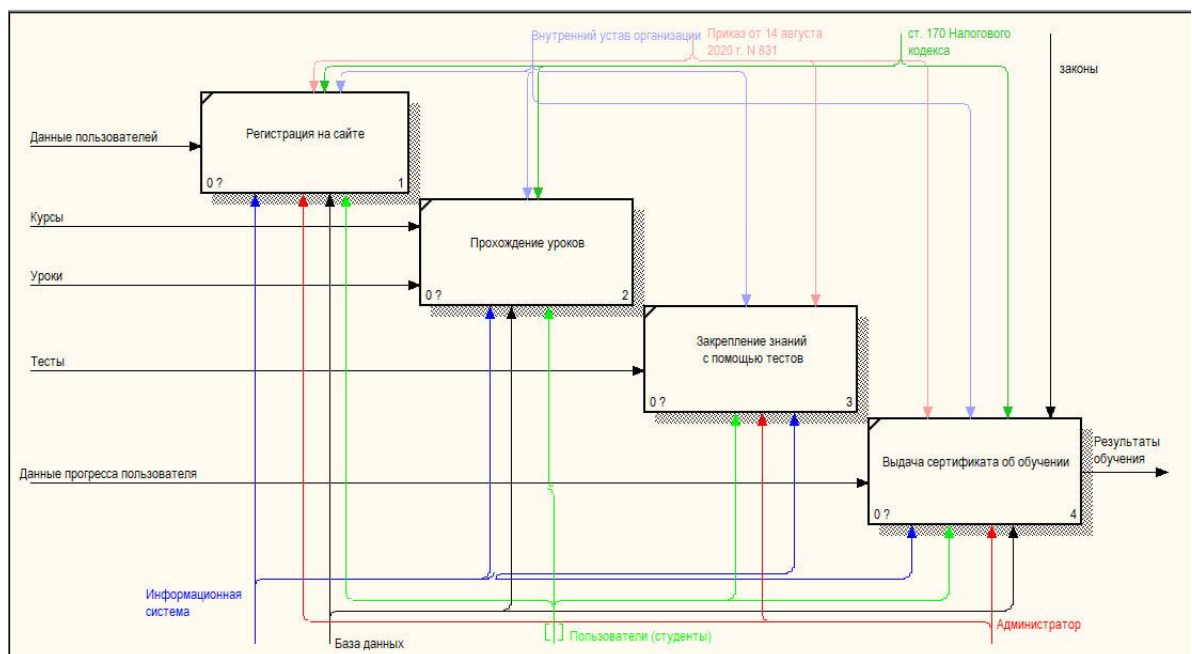


Диаграмма IDEF0 «To Be»



## 1. Сравнение архитектур «As is» и «To be»

- **Модульность и разделение ответственности:**

В архитектуре «To be» наблюдается чёткое разделение между слоями (представление, бизнес-логика, доступ к данным), что облегчает масштабирование и сопровождение. В существующей архитектуре («As is») обнаружены случаи чрезмерной связности между модулями, что затрудняет внесение изменений.

- **Сквозная функциональность:**

Проектируемая архитектура предусматривает централизованное управление аутентификацией, логированием, обработкой ошибок и кэшированием через middleware и специальные сервисы. В «As is» данные аспекты реализованы фрагментарно, что приводит к дублированию кода и усложнению поддержки.

- **Применение шаблонов проектирования:**

Архитектура «To be» опирается на проверенные шаблоны (например, Model-View-Template, Repository, Service), что способствует улучшенной тестируемости и расширяемости. В существующей реализации подобные подходы используются не везде, что затрудняет масштабирование системы.

## 2. Анализ причин отличий

- **Эволюция системы:**

Архитектура «As is» формировалась в ходе быстрого прототипирования, где приоритетом была скорость разработки, а не соблюдение принципов модульности и сквозной функциональности.

- **Ограниченные сроки:**

На ранних этапах разработки не уделялось достаточное внимание внедрению единых решений для сквозных аспектов, что привело к фрагментарной реализации таких функций, как аутентификация и логирование.

- **Отсутствие чётких стандартов:**

Недостаток использования единых архитектурных стандартов и шаблонов проектирования способствовал появлению рассогласованностей в структуре кода.

## 3. Пути улучшения архитектуры (Рефакторинг)

Для повышения качества архитектурного решения рекомендуется:

- **Улучшить модульность и разделение ответственности:**  
Провести рефакторинг, выделив отдельные слои для представления, бизнес-логики и доступа к данным. Применить паттерн Model-View-Template (MVT) в рамках Django для чёткого разделения функционала.
- **Централизовать сквозные функции:**  
Внедрить единый middleware для аутентификации, логирования, обработки ошибок и кэширования, чтобы избежать дублирования кода и упростить сопровождение.
- **Использовать шаблоны проектирования:**  
Применять такие шаблоны, как Repository, Service и Dependency Injection для повышения тестируемости, снижения связности между компонентами и облегчения масштабирования системы.
- **Оптимизировать производительность:**  
Реализовать кэширование на уровне запросов к базе данных, оптимизировать запросы ORM и проводить регулярный аудит производительности для своевременного выявления и устранения узких мест.
- **Стандартизировать код и документировать архитектуру:**  
Разработать и внедрить стандарты кодирования, а также вести подробную документацию по архитектурным решениям, что позволит новым участникам проекта быстрее включаться в работу и обеспечит единообразие разработки.

## **Вывод**

В результате проведённого исследования были выявлены основные различия между существующей («As is») и проектируемой («To be») архитектурами. В проектируемой архитектуре достигнуто лучшее разделение ответственности, централизовано реализована сквозная функциональность и заложены основы для масштабируемости и поддержки системы. Анализ существующего решения выявил слабые места, связанные с высокой связностью компонентов и фрагментарной реализацией критически важных функций. Предложенные пути рефакторинга, включая применение шаблонов проектирования и централизованное управление сквозными аспектами, позволят повысить надёжность, производительность и сопровождаемость системы.