**Автономная некоммерческая организация высшего образования**
**«Волгоградский институт бизнеса»**

# Рабочая программа учебной дисциплины

**Разработка клиент-серверных приложений**
(Наименование дисциплины)

**54.03.01 Дизайн, направленность (профиль) «Цифровой дизайн»**
(Направление подготовки / Профиль)

**Бакалавр**
(Квалификация)

**Кафедра разработчик** Экономики и управления
**Год набора** **2026**

| Вид учебной деятельности | Трудоемкость (объем) дисциплины | |
|---|---|---|
| | Очная форма | Очно-заочная форма |
| | д | в |
| Зачетные единицы | **6** | **6** |
| Общее количество часов | 216 | 216 |
| Аудиторные часы контактной работы обучающегося с преподавателями: | 64 | 40 |
| – Лекционные (Л) | 32 | 20 |
| – Практические (ПЗ) | | |
| – Лабораторные (ЛЗ) | 32 | 20 |
| – Семинарские (СЗ) | | |
| Самостоятельная работа обучающихся (СРО) | 116 | 140 |
| К (Р-Г) Р (П) (+;-) | | |
| Тестирование (+;-) | | |
| ДКР (+;-) | | |
| Зачет (+;-) | + | + |
| Зачет с оценкой (+;- (Кол-во часов)) | | |
| Экзамен (+;- (Кол-во часов)) | +(36) | +(36) |

**Волгоград 2026**

**Содержание**

## Раздел 1. Организационно-методический раздел

### 1.1. Цели освоения дисциплины

Дисциплина **«Разработка клиент-серверных приложений»** входит в Часть дисциплин, формируемую участниками образовательных отношений для подготовки обучающихся по направлению подготовки 54.03.01 Дизайн, направленность (профиль) «Цифровой дизайн».

Целью дисциплины является формирование **компетенций** (*в соответствии с ФГОС ВО и требованиями к результатам освоения основной профессиональной образовательной программы высшего образования (ОПОП ВО)*):

**ПК-4. Способен к художественно-технической разработке дизайн-проектов объектов визуальной информации, идентификации и коммуникации**

ПК-4.1. Способен использовать специализированное ПО для проектирования.

ПК-4.2. Способен подготовить дизайн-проект проект с учетом современных технологий реализации.

Перечисленные компетенции формируются в процессе достижения **индикаторов компетенций:**

| **Обобщенная трудовая функция/ трудовая функция** | **Код и наименование дескриптора компетенций** | **Код и наименование индикатора достижения компетенций (из ПС)** |
|---|---|---|
| **ПК-4** Способен к художественно-технической разработке дизайн-проектов объектов визуальной информации, идентификации и коммуникации (ПС 11.013 Графический дизайнер код B/02.6) | ПК-4.1 Способен использовать специализированное ПО для проектирования<br><br>ПК-4.2 Способен подготовить дизайн-проект проект с учетом современных технологий реализации | **Знание***:*<br>ИД-1 ПК-4.1 Компьютерное программное обеспечение, используемое в дизайне объектов визуальной информации, идентификации и коммуникации B/02.6<br>ИД-2 ПК-4.2 Технологические процессы производства в области полиграфии, упаковки, кино и телевидения B/02.6<br>**Умения***:*<br>ИД-3 ПК-4.1 Использовать специальные компьютерные программы для проектирования объектов визуальной информации, идентификации и коммуникации B/02.6<br>ИД-4 ПК-4.2 Учитывать при проектировании объектов визуальной информации, идентификации и коммуникации свойства используемых материалов и технологии реализации дизайн-проектов B/02.6<br>**Навыки и (или)опыт деятельности:**<br>ИД-5 ПК-4.1 Разработка дизайн-макета объекта визуальной информации, идентификации и коммуникации B/02.6<br>ИД-6 ПК-4.2 Подготовка графических материалов для передачи в производство B/02.6 |

### 1.2. Место дисциплины в структуре ОПОП ВО направления подготовки 54.03.01 Дизайн, направленность (профиль) «Цифровой дизайн»

| **№** | **Предшествующие дисциплины (дисциплины, изучаемые параллельно)** | **Последующие дисциплины** |
|---|---|---|
| *1* | *2* | *3* |
| 1 | Компьютерная графика | Современные архитектуры нейронных сетей для цифрового дизайна |
| 2 | Графический дизайн | Анимация интерфейсов |

| | | |
|---|---|---|
| 3 | Программирование для дизайна | Моушн-дизайн |
| 4 | | Программная инженерия |

*Последовательность формирования компетенций в указанных дисциплинах может быть изменена в зависимости от формы и срока обучения, а также преподавания с использованием дистанционных технологий обучения.*

### 1.3. Нормативная документация

Рабочая программа учебной дисциплины составлена на основе:

- Федерального государственного образовательного стандарта высшего образования по направлению подготовки **54.03.01 Дизайн**;
- Учебного плана направления подготовки **54.03.01 Дизайн, направленность (профиль) «Цифровой дизайн»** 2026 года набора;
- Образца рабочей программы учебной дисциплины (приказ № 113-О от 01.09.2021 г.).

## Раздел 2. Тематический план

### Очная форма обучения (полный срок)

| № | Тема дисциплины | Трудоемкость | | | | |
|---|---|---|---|---|---|---|
| | | Всего | Аудиторные занятия | | СРО | Код индикатора и дескриптора достижения компетенций |
| | | | Л | ПЗ (ЛЗ, СЗ) | | |
| *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | Архитектура клиент-серверных систем и модель распределённых вычислений | 14 | 2 | 2 | 10 | ИД-1 ПК-4.1<br>ИД-2 ПК-4.2 |
| 2 | React и TypeScript: компонентный подход и управление состоянием | 14 | 2 | 2 | 10 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 3 | Роутинг и навигация в современных веб-приложениях | 14 | 2 | 2 | 10 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 4 | Асинхронные данные и стратегии кэширования на клиенте | 14 | 2 | 2 | 10 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 5 | Среда выполнения Node.js и событийно-ориентированная модель | 14 | 2 | 2 | 10 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 6 | Фреймворк Express.js: маршрутизация и промежуточная обработка | 14 | 2 | 2 | 10 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 7 | Валидация входящих данных и защита серверных эндпоинтов | 14 | 2 | 2 | 10 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 8 | Интеграция клиентской и серверной частей в единый цикл разработки | 10 | 2 | 2 | 6 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| **Вид промежуточной аттестации (Зачет)** | | **0** | | | | |
| | | **108** | **16** | **16** | **76** | |
| 9 | Реляционные базы данных и проектирование схемы хранения | 9 | 2 | 2 | 5 | ИД-1 ПК-4.1<br>ИД-2 ПК-4.2 |
| 10 | ORM и миграции: управление базой данных через код | 9 | 2 | 2 | 5 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 11 | Аутентификация и авторизация веб-приложений | 9 | 2 | 2 | 5 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 12 | Безопасность веб-приложений и защита от типовых атак | 9 | 2 | 2 | 5 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 13 | Бессерверные функции и оптимизация под Vercel | 9 | 2 | 2 | 5 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 14 | Деплой, предварительные среды и управление релизами | 9 | 2 | 2 | 5 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 15 | Автоматизация сборки и контроль качества кода | 9 | 2 | 2 | 5 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 16 | Мониторинг, аналитика и поддержка промышленного окружения | 9 | 2 | 2 | 5 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| **Вид промежуточной аттестации (Экзамен)** | | **36** | **16** | **16** | **40** | |
| **Итого** | | **216** | **32** | **32** | **116** | |

### Очно-заочная форма обучения (полный срок)

| № | Тема дисциплины | Трудоемкость | | | |
|---|---|---|---|---|---|
| | | Всего | Аудиторные | СРО | Код |

| | | | занятия | | | индикатора и дескриптора достижения компетенций |
|---|---|---|---|---|---|---|
| | | | Л | ПЗ (ЛЗ, СЗ) | | |
| *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | Архитектура клиент-серверных систем и модель распределённых вычислений | 14 | 1 | 1 | 12 | ИД-1 ПК-4.1<br>ИД-2 ПК-4.2 |
| 2 | React и TypeScript: компонентный подход и управление состоянием | 14 | 1 | 1 | 12 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 3 | Роутинг и навигация в современных веб-приложениях | 14 | 1 | 1 | 12 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 4 | Асинхронные данные и стратегии кэширования на клиенте | 14 | 1 | 1 | 12 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 5 | Среда выполнения Node.js и событийно-ориентированная модель | 14 | 1 | 1 | 12 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 6 | Фреймворк Express.js: маршрутизация и промежуточная обработка | 14 | 1 | 1 | 12 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 7 | Валидация входящих данных и защита серверных эндпоинтов | 14 | 1 | 1 | 12 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 8 | Интеграция клиентской и серверной частей в единый цикл разработки | 10 | 1 | 1 | 8 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| **Вид промежуточной аттестации (Зачет)** | | **0** | | | | |
| | | **108** | **8** | **8** | **92** | |
| 9 | Реляционные базы данных и проектирование схемы хранения | 9 | 2 | 2 | 5 | ИД-1 ПК-4.1<br>ИД-2 ПК-4.2 |
| 10 | ORM и миграции: управление базой данных через код | 9 | 2 | 2 | 5 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 11 | Аутентификация и авторизация веб-приложений | 9 | 1 | 1 | 7 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 12 | Безопасность веб-приложений и защита от типовых атак | 9 | 1 | 1 | 7 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 13 | Бессерверные функции и оптимизация под Vercel | 9 | 2 | 2 | 5 | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 14 | Деплой, предварительные среды и управление релизами | 9 | 2 | 2 | 5 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 15 | Автоматизация сборки и контроль качества кода | 9 | 1 | 1 | 7 | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 16 | Мониторинг, аналитика и поддержка промышленного окружения | 9 | 1 | 1 | 7 | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| **Вид промежуточной аттестации (Экзамен)** | | **36** | **12** | **12** | **48** | |
| **Итого** | | **216** | **20** | **20** | **140** | |

## Раздел 3. Содержание дисциплины

### 3.1. Содержание дисциплины

**Тема 1. Архитектура клиент-серверных систем и модель распределённых вычислений.**

Принципы разделения ответственности между клиентской (интерфейсной, реализуемой на JavaScript/TypeScript) и серверной (вычислительной, на Node.js) частями приложения. Жизненный цикл протокола передачи гипертекста (HTTP): методы запроса (получение, создание, обновление, частичное изменение, удаление), коды состояний ответа (успешные, перенаправления, ошибки клиента, ошибки сервера), заголовки запроса и ответа. Сравнение архитектурных подходов: одностраничные приложения (SPA, реализуемые на React/Vue/Angular), многостраничные приложения (MPA) и серверный рендеринг (SSR через Next.js/Remix). Бессессионная архитектура и её влияние на масштабируемость. Понятие холодного старта в бессерверных средах. Диаграммы потока данных и модель C4 (Context, Containers, Components, Code). Обоснование выбора архитектуры под задачи коммуникации и производительности.

**Тема 2. React и TypeScript: компонентный подход и управление состоянием.**

Основы функциональных компонентов и расширения синтаксиса JavaScript (JSX), позволяющего размещать разметку HTML внутри программного кода TypeScript. Строгая типизация свойств компонентов и событийных обработчиков через интерфейсы TypeScript. Хуки управления состоянием: отслеживание изменений, выполнение побочных эффектов, мемоизация вычислений и функций. Правила вызова хуков и предотвращение скрытых побочных эффектов. Принцип однонаправленного потока данных и неизменяемость состояний. Настройка линтеров и форматировщиков (ESLint, Prettier) для обеспечения единого стиля кода. Интеграция дизайн-токенов через каскадные таблицы стилей (CSS) и пользовательские свойства. Критерии чистоты, переиспользуемости и тестируемости компонентов.

**Тема 3. Роутинг и навигация в современных веб-приложениях.**

Принципы клиентской маршрутизации на примере библиотеки React Router (или файлового роутера Next.js App Router). Динамические и вложенные маршруты, позволяющие строить сложные структуры страниц. Управление историей браузера, параметрами адресной строки и поисковыми запросами. Защищённые маршруты и автоматические перенаправления на основе ролей пользователя. Обработка отсутствующих страниц и состояний промежуточной загрузки. Паттерн макетов для переиспользования шапки, навигации и подвала между страницами. Оптимизация навигации с учётом пользовательских сценариев и сохранения локального состояния при переходах без полной перезагрузки страницы.

**Тема 4. Асинхронные данные и стратегии кэширования на клиенте.**

Взаимодействие с программными интерфейсами приложений (API) через встроенный метод получения данных (Fetch API) и специализированные клиентские библиотеки (Axios): базовая конфигурация, посредники-интерцепторы для внедрения заголовков авторизации, контроль отмены запросов (AbortController). Управление состояниями интерфейса: загрузка, ошибка, пустое множество данных, повторный запрос. Библиотеки управления серверным состоянием (TanStack Query / React Query, SWR): автоматическое кэширование, фоновое обновление, принудительная очистка устаревшего кэша. Оптимистичные обновления для мгновенной реакции интерфейса до подтверждения сервером. Обработка гоночных условий и дублирующих запросов. Принципы плавного снижения функциональности при сетевых проблемах.

**Тема 5. Среда выполнения Node.js и событийно-ориентированная модель.**

Архитектура среды выполнения JavaScript на сервере (Node.js): однопоточность, цикл обработки событий (Event Loop), очереди микрозадач и макрозадач. Различия между синхронным и асинхронным выполнением кода. Модульные системы: стандарт CommonJS и модули ECMAScript (ESM). Управление зависимостями через файл описания проекта (package.json) и

пакетные менеджеры (npm, yarn, pnpm). Работа с файловой системой и создание базового HTTP-сервера без использования сторонних фреймворков. Отладка через встроенный инспектор и структурированное логирование. Понимание ограничений среды выполнения для последующей корректной адаптации под облачные платформы.

**Тема 6. Фреймворк Express.js: маршрутизация и промежуточная обработка.**

Структура серверного приложения: маршрутизатор (распределитель запросов), контроллеры (обработчики бизнес-логики), сервисный слой (работа с данными). Концепция промежуточного программного обеспечения (middleware): парсинг тел запросов (express.json), настройка механизма совместного использования ресурсов разными источниками (CORS), логирование (morgan, pino), сжатие ответов. Организация линейной цепочки обработки запросов. Централизованная обработка ошибок и возврат структурированных ответов в формате JSON. Разделение конфигураций для сред разработки и промышленной эксплуатации. Принципы создания предсказуемого, документируемого и тестируемого интерфейса для операций создания, чтения, обновления и удаления данных.

**Тема 7. Валидация входящих данных и защита серверных эндпоинтов.**

Принципы строгой проверки запросов на уровне схемы перед запуском бизнес-логики. Использование библиотек описания схем (Zod) для типизации и проверки входящих параметров. Кастомные правила проверки и трансформация данных перед обработкой. Маппинг ошибок валидации в коды статуса HTTP 400 (некорректный запрос) и 422 (необрабатываемая сущность). Очистка строк от вредоносного кода для предотвращения инъекционных атак. Безопасное хранение конфигураций через файлы переменных окружения (dotenv) и их исключение из системы контроля версий. Логирование некорректных запросов для последующего аудита. Формирование чётких контрактов взаимодействия с клиентской частью.

**Тема 8. Интеграция клиентской и серверной частей в единый цикл разработки.**

Настройка проксирования запросов в среде разработки (Vite proxy, Next.js rewrites) для обхода ограничений браузера по кросс-доменным запросам. Синхронизация интерфейсов между фронтендом и бэкендом для предотвращения рассогласования типов TypeScript. Управление состоянием после изменения данных: принудительная очистка кэша (React Query invalidateQueries), обновление списков, обработка мгновенных ответов сервера. Реализация форм с двойной проверкой: мгновенная клиентская и окончательная серверная. Обработка сетевых таймаутов и состояния отмены запросов. Тестирование полного пользовательского сценария работы с данными. Организация структуры единого или раздельного репозитория для удобства поддержки кода.

**Тема 9. Реляционные базы данных и проектирование схемы хранения.**

Основы реляционной модели: таблицы, первичные (уникальные идентификаторы записей) и внешние ключи (ссылки на другие таблицы). Нормализация данных и устранение избыточности. Типы связей: один-к-одному, один-ко-многим, многие-ко-многим. Индексы и их влияние на скорость выборки. Транзакции и требования к их выполнению: атомарность, согласованность, изолированность, долговечность (ACID). Подключение к управляемым серверам PostgreSQL (Neon, Vercel Postgres, Supabase) через строку подключения. Пулы соединений и управление лимитами одновременных подключений. Визуализация схемы через диаграммы сущность-связь (ERD) и экспорт скриптов определения данных (DDL).

**Тема 10. ORM и миграции: управление базой данных через код.**

Концепция объектно-реляционного отображения (ORM, Prisma, Drizzle): преимущества программной абстракции над прямыми запросами к базе данных и ограничения производительности. Описание моделей данных в специализированных библиотеках: типы полей, связи, уникальные индексы. Генерация типизированного клиента базы данных. Создание и применение миграций: инициализация, разработка, промышленная эксплуатация (prisma migrate, drizzle-kit). Откат версий схемы и разрешение конфликтов структуры. Заполнение базы тестовыми

данными (seed). Типобезопасные запросы без ручного написания структурированного языка запросов (SQL). Стратегии обновления схемы данных без простоя приложения.

**Тема 11. Аутентификация и авторизация веб-приложений.**

Различия между идентификацией, подтверждением личности и проверкой прав доступа. Механизмы хранения сессий: веб-токены формата JSON (JWT) против серверных сессий (express-session, lucia). Хэширование паролей с использованием криптографических алгоритмов (bcrypt, argon2) и добавление случайной строки для усложнения подбора. Поток выдачи и обновления токенов доступа (refresh token flow). Безопасное хранение токенов в защищённых cookie-файлах с атрибутами безопасной передачи и ограничения по домену (httpOnly, Secure, SameSite). Ролевая модель доступа: проверка прав на сервере и условное отображение элементов на клиенте. Реализация хука проверки авторизации и защищённых маршрутов. Корректный выход из системы и принудительная очистка сессий.

**Тема 12. Безопасность веб-приложений и защита от типовых атак.**

Обзор десяти наиболее критических рисков безопасности веб-приложений от открытого проекта (OWASP Top 10). Предотвращение межсайтового скриптинга (XSS): экранирование вывода, настройка политики безопасности контента (CSP). Защита от межсайтовой подделки запросов (CSRF): токены синхронизации, ограничения cookie-файлов. Предотвращение инъекций структурированного языка запросов (SQLi) через объектно-реляционное отображение и параметризованные запросы. Ограничение частоты запросов для защиты от автоматического перебора паролей (rate-limiter-flexible, express-rate-limit). Аудит зависимостей (npm audit, audit-ci), обновление уязвимых пакетов. Настройка защитных HTTP-заголовков через специализированные библиотеки (helmet). Документирование мер защиты и формирование чек-листа перед релизом.

**Тема 13. Бессерверные функции и оптимизация под Vercel.**

Архитектура бессерверных вычислений (Serverless): функции, запускаемые по требованию, изоляция сред, автоматическое горизонтальное масштабирование. Ограничения облачной среды выполнения: время отклика, объём оперативной памяти, временное хранилище. Адаптация кода сервера под формат бессерверных функций платформы (@vercel/node, Vercel Functions). Среда выполнения на границе сети: выполнение кода географически ближе к пользователю, минимальная задержка, ограничения доступных системных библиотек (Vercel Edge Runtime). Оптимизация холодного старта: уменьшение размера пакета, ленивая загрузка модулей, переиспользование соединений с базой данных. Управление переменными окружения через интерфейс командной строки (Vercel CLI) и веб-дашборд. Паттерны проектирования для бессессионной среды.

**Тема 14. Деплой, предварительные среды и управление релизами.**

Конфигурация развёртывания через файл настроек платформы (vercel.json): перенаправления, кастомные заголовки, настройки сборки. Интеграция с системой контроля версий (Git, GitHub/GitLab): автоматическое развёртывание при отправке кода, создание предварительных окружений для запросов на слияние изменений (Vercel Preview Deployments). Управление переменными для разных сред: разработка, предварительный просмотр, промышленная эксплуатация (Vercel Environments). Подключение кастомного домена и автоматическое получение сертификатов защищённого соединения (SSL/TLS, Let's Encrypt). Инвалидация кэша сети доставки контента (CDN) при обновлении файлов. Стратегии развёртывания без простоя приложения. Откат версии при критических ошибках. Документирование процедуры выпуска и проверка работоспособности на каждом этапе.

**Тема 15. Автоматизация сборки и контроль качества кода.**

Концепция непрерывной интеграции и непрерывной доставки (CI/CD): автоматическая проверка и поставка кода. Настройка платформы автоматизации (GitHub Actions, GitLab CI): триггеры запуска, шаги выполнения, матрица версий среды выполнения. Этапы конвейера сборки: проверка стиля кода (ESLint, Stylelint), форматирование (Prettier), запуск модульных и

интеграционных тестов (Jest, Vitest, Playwright), сборка готовых файлов. Кэширование зависимостей для ускорения повторных сборок. Хуки перед фиксацией изменений (Husky, lint-staged) для автоматической проверки кода до отправки в репозиторий. Управление секретами и токенами доступа в автоматизированной среде. Блокировка слияния веток при падении тестов (Branch Protection Rules). Интеграция конвейера с платформой развёртывания для автоматического продвижения стабильной сборки.

**Тема 16. Мониторинг, аналитика и поддержка промышленного окружения.**

Принципы наблюдаемости системы (Observability): сбор журналов событий, метрик производительности, трассировка пользовательских запросов. Встроенная аналитика платформы (Vercel Analytics, Next.js Analytics) и журналы бессерверных функций (Vercel Function Logs): анализ задержек, ошибок и потребления ресурсов. Интеграция со сторонними системами отслеживания исключений на клиенте и сервере (Sentry, Logflare, OpenTelemetry). Отслеживание ключевых метрик скорости загрузки, времени первой реакции и визуальной стабильности интерфейса (Core Web Vitals: LCP, FID/INP, CLS), оптимизация размера загружаемых файлов (Webpack/Vite bundle analyzer). Настройка уведомлений при аномалиях доступности или росте количества ошибок сервера. Ротация журналов событий и управление квотами хранения. Базовые практики управления инцидентами и послевыпускного анализа. Составление итогового отчёта о стабильности системы после релиза.

### 3.2. Содержание практического блока дисциплины

**Очная форма обучения (полный срок)**

| № | Тема практического (семинарского, лабораторного) занятия |
|---|---|
| *1* | *2* |
| ПЗ 1 | Архитектура клиент-серверных систем и модель распределённых вычислений |
| ПЗ 2 | React и TypeScript: компонентный подход и управление состоянием |
| ПЗ 3 | Роутинг и навигация в современных веб-приложениях |
| ПЗ 4 | Асинхронные данные и стратегии кэширования на клиенте |
| ПЗ 5 | Среда выполнения Node.js и событийно-ориентированная модель |
| ПЗ 6 | Фреймворк Express.js: маршрутизация и промежуточная обработка |
| ПЗ 7 | Валидация входящих данных и защита серверных эндпоинтов |
| ПЗ 8 | Интеграция клиентской и серверной частей в единый цикл разработки |
| ПЗ 9 | Реляционные базы данных и проектирование схемы хранения |
| ПЗ 10 | ORM и миграции: управление базой данных через код |
| ПЗ 11 | Аутентификация и авторизация веб-приложений |
| ПЗ 12 | Безопасность веб-приложений и защита от типовых атак |
| ПЗ 13 | Бессерверные функции и оптимизация под Vercel |
| ПЗ 14 | Деплой, предварительные среды и управление релизами |
| ПЗ 15 | Автоматизация сборки и контроль качества кода |
| ПЗ 16 | Мониторинг, аналитика и поддержка промышленного окружения |

**Очно-заочная форма обучения (полный срок)**

| № | Тема практического (семинарского, лабораторного) занятия |
|---|---|
| *1* | *2* |
| ПЗ 1 | Архитектура клиент-серверных систем и модель распределённых вычислений. React и TypeScript: компонентный подход и управление состоянием |
| ПЗ 2 | Роутинг и навигация в современных веб-приложениях. Асинхронные данные и стратегии кэширования на клиенте |
| ПЗ 3 | Среда выполнения Node.js и событийно-ориентированная модель. Фреймворк Express.js: маршрутизация и промежуточная обработка |
| ПЗ 4 | Валидация входящих данных и защита серверных эндпоинтов. Интеграция клиентской и серверной частей в единый цикл разработки |
| ПЗ 5 | Реляционные базы данных и проектирование схемы хранения |
| ПЗ 6 | ORM и миграции: управление базой данных через код |
| ПЗ 7 | Аутентификация и авторизация веб-приложений. Безопасность веб-приложений и защита от типовых атак |
| ПЗ 8 | Бессерверные функции и оптимизация под Vercel |
| ПЗ 9 | Деплой, предварительные среды и управление релизами |
| ПЗ 10 | Автоматизация сборки и контроль качества кода. Мониторинг, аналитика и поддержка промышленного окружения |

### 3.3. Образовательные технологии

**Очная форма обучения (полный срок)**

| № | Тема занятия | Вид | Форма / Методы | % |
|---|---|---|---|---|

| | | учебного занятия | интерактивного обучения | учебного времени |
|---|---|---|---|---|
| *1* | *2* | *3* | *4* | *5* |
| 1. | Архитектура клиент-серверных систем и модель распределённых вычислений | ПЗ | Метод проектов | 100 |
| 2. | React и TypeScript: компонентный подход и управление состоянием | ПЗ | Метод проектов | 100 |
| 3. | Роутинг и навигация в современных веб-приложениях | ПЗ | Метод проектов | 100 |
| 4. | Асинхронные данные и стратегии кэширования на клиенте | ПЗ | Метод проектов | 100 |
| 5. | Среда выполнения Node.js и событийно-ориентированная модель | ПЗ | Метод проектов | 100 |
| 6. | Фреймворк Express.js: маршрутизация и промежуточная обработка | ПЗ | Метод проектов | 100 |
| 7. | Валидация входящих данных и защита серверных эндпоинтов | ПЗ | Метод проектов | 100 |
| 8. | Интеграция клиентской и серверной частей в единый цикл разработки | ПЗ | Метод проектов | 100 |
| 9. | Реляционные базы данных и проектирование схемы хранения | ПЗ | Метод проектов | 100 |
| 10. | ORM и миграции: управление базой данных через код | ПЗ | Метод проектов | 100 |
| 11. | Аутентификация и авторизация веб-приложений | ПЗ | Метод проектов | 100 |
| 12. | Безопасность веб-приложений и защита от типовых атак | ПЗ | Метод проектов | 100 |
| 13. | Бессерверные функции и оптимизация под Vercel | ПЗ | Метод проектов | 100 |
| 14. | Деплой, предварительные среды и управление релизами | ПЗ | Метод проектов | 100 |
| 15. | Автоматизация сборки и контроль качества кода | ПЗ | Метод проектов | 100 |
| 16. | Мониторинг, аналитика и поддержка промышленного окружения | ПЗ | Метод проектов | 100 |
| **Итого %** | | | | **100%** |

**Очно-заочная форма обучения (полный срок)**

| № | Тема занятия | Вид учебного занятия | Форма / Методы интерактивного обучения | % учебного времени |
|---|---|---|---|---|
| *1* | *2* | *3* | *4* | *5* |
| 1. | Архитектура клиент-серверных систем и модель распределённых вычислений. React и TypeScript: компонентный подход и управление состоянием | ПЗ | Метод проектов | 100 |
| 2. | Роутинг и навигация в современных веб-приложениях. Асинхронные данные и стратегии кэширования на клиенте | ПЗ | Метод проектов | 100 |

| | | | | |
|---|---|---|---|---|
| 3. | Среда выполнения Node.js и событийно-ориентированная модель. Фреймворк Express.js: маршрутизация и промежуточная обработка | ПЗ | Метод проектов | 100 |
| 4. | Валидация входящих данных и защита серверных эндпоинтов. Интеграция клиентской и серверной частей в единый цикл разработки | ПЗ | Метод проектов | 100 |
| 5. | Реляционные базы данных и проектирование схемы хранения | ПЗ | Метод проектов | 100 |
| 6. | ORM и миграции: управление базой данных через код | ПЗ | Метод проектов | 100 |
| 7. | Аутентификация и авторизация веб-приложений. Безопасность веб-приложений и защита от типовых атак | ПЗ | Метод проектов | 100 |
| 8. | Бессерверные функции и оптимизация под Vercel | ПЗ | Метод проектов | 100 |
| 9. | Деплой, предварительные среды и управление релизами | ПЗ | Метод проектов | 100 |
| 10. | Автоматизация сборки и контроль качества кода. Мониторинг, аналитика и поддержка промышленного окружения | ПЗ | Метод проектов | 100 |
| **Итого %** | | | | **30%** |

## Раздел 4. Организация самостоятельной работы обучающихся

### 4.1. Организация самостоятельной работы обучающихся

| № | Тема дисциплины | № вопросов | № рекомендуемой литературы |
|---|---|---|---|
| *1* | *2* | *3* | *4* |
| | Архитектура клиент-серверных систем и модель распределённых вычислений | 1-3 | 1-11 |
| 2. | React и TypeScript: компонентный подход и управление состоянием | 4-6 | 1-11 |
| 3. | Роутинг и навигация в современных веб-приложениях | 7-9 | 1-11 |
| 4. | Асинхронные данные и стратегии кэширования на клиенте | 10-12 | 1-11 |
| 5. | Среда выполнения Node.js и событийно-ориентированная модель | 13-15 | 1-11 |
| 6. | Фреймворк Express.js: маршрутизация и промежуточная обработка | 16-18 | 1-11 |
| 7. | Валидация входящих данных и защита серверных эндпоинтов | 19-21 | 1-11 |
| 8. | Интеграция клиентской и серверной частей в единый цикл разработки | 22-24 | 1-11 |
| 9. | Реляционные базы данных и проектирование схемы хранения | 25-28 | 1-11 |
| 10. | ORM и миграции: управление базой данных через код | 29-30 | 1-11 |
| 11. | Аутентификация и авторизация веб-приложений | 31-33 | 1-11 |
| 12. | Безопасность веб-приложений и защита от типовых атак | 34-36 | 1-11 |
| 13. | Бессерверные функции и оптимизация под Vercel | 37-39 | 1-11 |
| 14. | Деплой, предварительные среды и управление релизами | 40-42 | 1-11 |
| 15. | Автоматизация сборки и контроль качества кода | 43-45 | 1-11 |
| 16. | Мониторинг, аналитика и поддержка промышленного окружения | 46-48 | 1-11 |

### Перечень вопросов, выносимых на самостоятельную работу обучающихся

1. Чем принципиально отличается stateless-архитектура от традиционной сессионной модели и как это влияет на горизонтальное масштабирование?
2. Как выбор между SPA и SSR влияет на время первой отрисовки (FCP) и индексацию контента поисковыми системами?
3. Что такое холодный старт в serverless-среде и какие факторы напрямую увеличивают его длительность?
4. Почему прямая мутация состояния в React приводит к ошибкам рендеринга и как правильно обновлять массивы и объекты?
5. Как строгая типизация пропсов через TypeScript предотвращает ошибки на этапе разработки, а не в production?
6. В каких случаях использование useMemo оправдано, а когда оно создаёт лишь накладные расходы на сравнение зависимостей?
7. Как клиентский роутер имитирует переход по страницам без перезагрузки браузера и изменения URL?
8. Что произойдёт с состоянием компонента при переходе на маршрут с тем же макетом, но изменёнными динамическими параметрами?
9. Как правильно организовать проверку авторизации в клиентском роутинге, чтобы избежать «мерцания» защищённых страниц?

10. Как библиотека управления серверным состоянием отличает свежий кэш от устаревшего и когда запускает фоновое обновление?
11. В чём разница между оптимистичным обновлением интерфейса и консервативным подходом с точки зрения UX и целостности данных?
12. Почему принудительная инвалидация кэша предпочтительнее ручного перезапроса данных после успешной мутации?
13. Почему блокирующий код в Node.js «замораживает» обработку всех входящих запросов, несмотря на однопоточность среды?
14. В какой очереди выполняются колбэки промисов (Promise.then) и как это связано с порядком вывода в консоль?
15. Какую роль играет фаза poll в цикле событий и какие операции обрабатываются именно на этом этапе?
16. Почему порядок подключения middleware в Express критически важен для корректной обработки запросов и ответов?
17. Как разделение на контроллеры и сервисы упрощает модульное тестирование и поддержку серверного кода?
18. Что произойдёт, если в обработчике ошибки не вызвать next(err) или не отправить финальный ответ клиенту?
19. Почему валидацию входящих данных необходимо выполнять до передачи параметров в бизнес-логику приложения?
20. Как кастомные сообщения об ошибках в схемах Zod повышают безопасность и удобство интеграции с фронтендом?
21. Какие риски возникают при хранении секретных ключей в исходном коде или их попадании в систему контроля версий?
22. Как механизм проксирования в среде разработки обходит ограничения браузера на кросс-доменные запросы?
23. Почему рассогласование типов между клиентом и сервером часто приводит к трудноуловимым ошибкам в production?
24. Как AbortController помогает предотвратить состояние гонки при быстрых переключениях между страницами или отмене запросов?
25. Как нормализация до третьей нормальной формы устраняет аномалии вставки, обновления и удаления данных?
26. Почему внешний ключ с каскадным удалением может привести к непреднамеренной потере связанных записей и как это контролировать?
27. Как анализ плана выполнения запроса (EXPLAIN ANALYZE) показывает, использует ли база данных созданные индексы?
28. Как ORM обеспечивает типобезопасность запросов и чем это лучше ручного составления SQL-строк?
29. Почему миграции необходимо версионировать и хранить в репозитории наравне с кодом приложения?
30. Что произойдёт при попытке применить миграцию к базе данных, схема которой уже изменена вручную в обход ORM?
31. В чём принципиальная разница между хранением JWT в localStorage и в httpOnly cookie с точки зрения безопасности?
32. Как refresh-токен позволяет продлевать сессию без повторного ввода пароля и какие риски несёт его утечка?
33. Почему проверка прав доступа должна выполняться на сервере, даже если интерфейс на клиенте уже скрыл запрещённые элементы?
34. Как настройка Content-Security-Policy блокирует выполнение вредоносных скриптов, загруженных не с доверенного источника?
35. Почему rate limiting на эндпоинтах авторизации эффективнее простого увеличения сложности пароля для защиты от перебора?
36. Как объектно-реляционное отображение защищает от SQL-инъекций на уровне построения и параметризации запросов?

37. Почему глобальные переменные в serverless-функциях не гарантируют сохранность состояния между разными вызовами?
38. Как вынесение инициализации соединения с базой данных за пределы обработчика функции сокращает время отклика?
39. Какие ограничения памяти и времени выполнения предъявляет платформа к бессерверным функциям и как их обходят архитектурно?
40. Как изоляция переменных окружения по средам (dev/preview/prod) предотвращает попадание тестовых данных в production?
41. Почему автоматическое развёртывание preview-окружений ускоряет процесс код-ревью и приёмочного тестирования?
42. Как сеть доставки контента (CDN) кэширует статические ресурсы и при каких условиях требуется принудительная инвалидация кэша?
43. Как кэширование директории node_modules в конвейере сборки сокращает время выполнения автоматических проверок?
44. Почему pre-commit хуки эффективнее постфактум проверок линтером после отправки кода в репозиторий?
45. Как правила защиты веток (branch protection) гарантируют, что в основную ветку попадёт только прошедший все тесты код?
46. Как source maps позволяют преобразовать минифицированный стек-трейс в читаемый исходный код при отладке исключений в Sentry?
47. Почему метрика CLS (Cumulative Layout Shift) напрямую влияет на пользовательский опыт и ранжирование в поисковых системах?
48. Как стратегия ротации логов предотвращает переполнение дискового пространства и сохраняет возможность ретроспективного аудита инцидентов?

**с**

### 4.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся

Самостоятельная работа обучающихся обеспечивается следующими учебно-методическими материалами:

1. Указаниями в рабочей программе по дисциплине (п.4.1.)
2. Лекционные материалы в составе учебно-методического комплекса по дисциплине
3. Заданиями и методическими рекомендациями по организации самостоятельной работы обучающихся в составе учебно-методического комплекса по дисциплине.
4. Глоссарием по дисциплине в составе учебно-методического комплекса по дисциплине.

## Раздел 5. Фонд оценочных средств для проведения текущего контроля и промежуточной аттестации обучающихся

*Фонд оценочных средств по дисциплине представляет собой совокупность контролирующих материалов, предназначенных для измерения уровня достижения обучающимися установленных результатов образовательной программы. ФОС по дисциплине используется при проведении оперативного контроля и промежуточной аттестации обучающихся. Требования к структуре и содержанию ФОС дисциплины регламентируются Положением о фонде оценочных материалов по программам высшего образования – программам бакалавриата, магистратуры.*

### 5.1. Паспорт фонда оценочных средств
### Очная форма обучения (полный срок)
### Очно-заочная форма обучения (полный срок)

| № | Контролируемые разделы (темы) дисциплины | Оценочные средства | | | |
|---|---|---|---|---|---|
| | | Л | ПЗ (ЛЗ, СЗ) | СРО | Код индикатора и дескриптора достижения компетенций |
| *1* | *2* | *3* | *4* | *5* | *6* |
| 1 | Архитектура клиент-серверных систем и модель распределённых вычислений | ЛС | МП,ЗЗ | ПРВ | ИД-1 ПК-4.1<br>ИД-2 ПК-4.2 |
| 2 | React и TypeScript: компонентный подход и управление состоянием | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 3 | Роутинг и навигация в современных веб-приложениях | ЛС | МП,ЗЗ | ПРВ | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 4 | Асинхронные данные и стратегии кэширования на клиенте | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 5 | Среда выполнения Node.js и событийно-ориентированная модель | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 6 | Фреймворк Express.js: маршрутизация и промежуточная обработка | ЛС | МП,ЗЗ | ПРВ | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 7 | Валидация входящих данных и защита серверных эндпоинтов | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 8 | Интеграция клиентской и серверной частей в единый цикл разработки | ЛС, УО | УО,ЗЗ | ПРВ, ЗЗ | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 9 | Реляционные базы данных и проектирование схемы хранения | ЛС | МП,ЗЗ | ПРВ | ИД-1 ПК-4.1<br>ИД-2 ПК-4.2 |
| 10 | ORM и миграции: управление базой данных через код | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 11 | Аутентификация и авторизация веб-приложений | ЛС | МП,ЗЗ | ПРВ | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 12 | Безопасность веб-приложений и защита от типовых атак | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 13 | Бессерверные функции и оптимизация под Vercel | ЛС | МП,ЗЗ | ПРВ | ИД-3 ПК-4.1<br>ИД-4 ПК-4.2 |
| 14 | Деплой, предварительные среды и управление релизами | ЛС | МП,ЗЗ | ПРВ | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |
| 15 | Автоматизация сборки и контроль качества кода | ЛС | УО,ЗЗ | ПРВ, ЗЗ | ИД-3 ПК-4.1<br>ИД-6 ПК-4.2 |
| 16 | Мониторинг, аналитика и поддержка промышленного окружения | ЛС, УО | УО,ЗЗ | ПРВ, ЗЗ | ИД-5 ПК-4.1<br>ИД-6 ПК-4.2 |

**Условные обозначения оценочных средств (Столбцы 3, 4, 5):**
***ЗЗ*** – Защита выполненных заданий (творческих, расчетных и т.д.), представление презентаций;
***Т*** – Тестирование компьютерное;
***УО*** – Устный (фронтальный, индивидуальный, комбинированный) опрос;
***КР*** – Контрольная работа (аудиторные или домашние, индивидуальные, парные или групповые контрольные, самостоятельные работы, диктанты и т.д.);
***К*** – Коллоквиум;
***ПРВ*** – Проверка рефератов, отчетов, рецензий, аннотаций, конспектов, графического материала, эссе, переводов, решений заданий, выполненных заданий в электронном виде и т.д.;
***ДИ*** – Деловая игра;
***РИ*** – Ролевая игра;
***КМ*** – Кейс-метод;
***КС*** – Круглый стол;
***КСМ*** – Компьютерная симуляция;
***МШ*** – Метод мозгового штурма;
***ЛС*** – Лекция-ситуация;
***ЛК*** – Лекция-конференция;
***ЛВ*** – Лекция-визуализация;
***ПЛ*** – Проблемная лекция;
***Д*** – Дискуссия, полемика, диспут, дебаты;
***П*** – Портфолио;
***ПВУ*** – Просмотр видеоуроков;
***МП*** – Метод проектов.

## 5.2. Оценочные средства текущего контроля

### Перечень практических (семинарских) заданий

### Тема 1. Архитектура клиент-серверных систем и модель распределённых вычислений

Цель практики: Научиться документировать потоки данных и обосновывать выбор архитектуры под конкретные ограничения.
Задачи: Построить схему взаимодействия компонентов, проанализировать HTTP-жизненный цикл, сравнить SPA/SSR для заданного кейса.
Входные данные:
• Техническое описание учебного проекта (целевая аудитория, требования к SEO, ожидаемый трафик)
• Примеры диаграмм уровня C4 (Context, Containers) из методических материалов
• Таблица сравнения архитектурных паттернов (SPA, MPA, SSR, Serverless)

Инструменты: Draw.io / Figma / Mermaid.js, текстовый процессор
Пошаговое задание:

1. Выделить основные компоненты системы (клиент, CDN, серверные функции, база данных, внешние API)
2. Построить диаграмму потока данных для сценария «загрузка главной страницы» и «отправка формы»
3. Заполнить таблицу: указать ожидаемые HTTP-методы, статус-коды и зоны ответственности каждого компонента
4. Выбрать архитектурный подход (SPA, SSR или гибридный) и обосновать выбор тремя аргументами
5. Оценить влияние холодного старта и бессессионной модели на выбранный стек
6. Оформить пояснительную записку на две–три страницы с прикреплённой диаграммй

Итоговый результат:
.pdf Диаграмма архитектуры и пояснительная записка / Фамилия_ПР1_АрхитектураСистемы
.md Заметки по анализу ограничений и выбору подхода / Фамилия_ПР1_АнализАрхитектуры

Критерии оценки (чек-лист):
• [ ] Диаграмма содержит не менее пяти компонентов и показывает направление потоков данных
• [ ] HTTP-сценарии корректно сопоставлены с зонами ответственности клиента и сервера
• [ ] Выбор архитектуры обоснован требованиями к SEO, производительности и бюджету
• [ ] Учтены ограничения серверной модели (холодный старт, stateless) в тексте записки

### Тема 2. React и TypeScript: компонентный подход и управление состоянием

Цель практики: Освоить создание типизированных компонентов и безопасное управление локальным состоянием.
Задачи: Настроить проект, реализовать переиспользуемые компоненты с хуками, подключить линтер.
Входные данные:
• Техническое задание на интерфейс (список карточек, форма фильтрации, индикатор состояния)
• Шаблон package.json с базовыми зависимостями (React, TypeScript, Vite, ESLint, Prettier)
• Гайд по типизации пропсов и событийных обработчиков

Инструменты: VS Code, Node.js, npm/pnpm, Vite, React Developer Tools
Пошаговое задание:

1. Инициализировать проект через Vite с флагом TypeScript, удалить шаблонный код
2. Создать три функциональных компонента: Card, FilterForm, StatusBadge с явно типизированными пропсами

3. Реализовать управление состоянием через useState, добавить useMemo для тяжёлых вычислений списка
4. Настроить ESLint и Prettier, исправить все предупреждения линтера
5. Добавить обработку кликов и изменение состояния без прямого DOM-манипулирования
6. Проверить корректность типов через встроенную проверку tsc --noEmit

Итоговый результат:
.zip Архив исходного кода проекта / Фамилия_ПР2_ReactTSКомпоненты
.png Скриншот работающего интерфейса в браузере / Фамилия_ПР2_Демонстрация

Критерии оценки (чек-лист):
• [ ] Все компоненты строго типизированы, отсутствуют any или @ts-ignore
• [ ] Состояние обновляется иммутабельно, используются правила вызова хуков
• [ ] Конфигурация ESLint/Prettier применена, проект собирает без ошибок
• [ ] Отсутствуют прямые манипуляции с DOM, рендеринг полностью декларативный

**Тема 3. Роутинг и навигация в современных веб-приложениях**
Цель практики: Реализовать клиентскую маршрутизацию с защищёнными и вложенными путями.
Задачи: Настроить React Router, создать макеты страниц, реализовать условный рендеринг маршрутов.
Входные данные:
• Карта маршрутов приложения (минимум 4 страницы, включая 404 и защищённую зону)
• Стартовый репозиторий с настроенными компонентами из ПР2
• Пример реализации guard-компонента для проверки авторизации
Инструменты: React Router DOM, VS Code, браузерные DevTools
Пошаговое задание:
1. Установить react-router-dom, создать конфигурацию роутера с BrowserRouter
2. Реализовать вложенные маршруты через Outlet и общий макет (Layout)
3. Настроить динамические параметры пути (/item/:id) и их получение через хук useParams
4. Создать компонент ProtectedRoute, перенаправляющий неавторизованных пользователей на страницу входа
5. Реализовать обработчик несуществующих маршрутов с компонентом Navigate и статусом 404
6. Протестировать навигацию, проверить сохранение состояния при переходах без перезагрузки

Итоговый результат:
.git Ссылка на ветку репозитория с реализованным роутингом / Фамилия_ПР3_Роутинг
.log Консольный лог переходов и проверок guard-компонента / Фамилия_ПР3_ТестНавигации

Критерии оценки (чек-лист):
• [ ] Структура маршрутов вложена корректно, общий макет не дублируется
• [ ] Динамические параметры извлекаются и используются в компонентах
• [ ] Защищённые маршруты перенаправляют при отсутствии флага авторизации
• [ ] Переходы не вызывают полной перезагрузки страницы, история браузера сохраняется

**Тема 4. Асинхронные данные и стратегии кэширования на клиенте**

Цель практики: Научиться управлять серверным состоянием, кэшировать запросы и обрабатывать сетевые ошибки.
Задачи: Подключить TanStack Query, реализовать состояния загрузки/ошибки, настроить инвалидацию кэша.
Входные данные:
• Мок-API с эндпоинтами GET /data, POST /data (Postman Mock или JSON Server)
• Шаблон интерфейсов TypeScript для ответов сервера
• Документация по хукам useQuery и useMutation

Инструменты: @tanstack/react-query, Mockoon/Postman, Network Tab браузера
Пошаговое задание:

1. Настроить QueryClient и QueryClientProvider в корне приложения
2. Реализовать useQuery для получения списка данных, отобразить состояния loading, error, success
3. Добавить обработку пустого ответа и повторный запрос через кнопку «Retry»
4. Реализовать useMutation для создания записи с оптимистичным обновлением кэша
5. Настроить invalidateQueries после успешной мутации для синхронизации списка
6. Замерить количество запросов при быстрых переходах, подтвердить работу кэша

Итоговый результат:
.tsx Код хуков и компонентов работы с данными / Фамилия_ПР4_КлиентскоеСостояние
.pdf Отчёт по сетевой активности (скриншоты Network Tab до и после настройки кэша) / Фамилия_ПР4_АнализСети

Критерии оценки (чек-лист):
• [ ] Реализованы все три UI-состояния (загрузка, ошибка, данные) без дублирования кода
• [ ] Оптимистичное обновление корректно синхронизируется с ответом сервера
• [ ] Кэш инвалидируется только при изменении связанных данных
• [ ] Количество дублирующих запросов при навигации сведено к минимуму

**Тема 5. Среда выполнения Node.js и событийно-ориентированная модель**

Цель практики: Понять механизм Event Loop и создать базовый сервер без фреймворков.
Задачи: Продемонстрировать разницу синхронного/асинхронного кода, реализовать маршрутизацию на чистом http.
Входные данные:
• Список операций разной природы (чтение файла, запрос к API, математические вычисления)
• Шаблон проекта с файлом server.js и настройками package.json
• Лог-шаблон для фиксации порядка выполнения задач
Инструменты: Node.js, встроенный модуль http/fs, терминал, отладчик VS Code
Пошаговое задание:

1. Написать скрипт с блокирующим и неблокирующим кодом, зафиксировать порядок вывода в консоль
2. Создать базовый HTTP-сервер на модуле http, обрабатывающий пути /, /api/data, /api/status
3. Реализовать парсинг URL и методов запроса без внешних библиотек
4. Добавить обработку ошибок маршрутизации (возврат 404 с JSON-телом)
5. Настроить чтение статического файла при запросе /about
6. Проверить работу сервера через curl, зафиксировать время отклика при одновременных запросах

Итоговый результат:
.js Файл серверной логики с комментариями по Event Loop / Фамилия_ПР5_БазовыйСервер
.log Вывод консоли с порядком выполнения задач и замерами времени / Фамилия_ПР5_EventLoopЛог

Критерии оценки (чек-лист):
• [ ] Демонстрация различий между микрозадачами, макрозадачами и блокирующим кодом корректна
• [ ] Сервер обрабатывает минимум три маршрута и возвращает корректные статус-коды
• [ ] Ошибки маршрутизации не приводят к падению процесса, возвращают JSON
• [ ] Одновременные запросы обрабатываются параллельно благодаря неблокирующей модели

**Тема 6. Фреймворк Express.js: маршрутизация и промежуточная обработка**
Цель практики: Построить структурированный API с разделением слоёв и логированием.

Задачи: Настроить Express, реализовать цепочку middleware, создать контроллеры и сервисы.
Входные данные:
• Спецификация API (эндпоинты CRUD для сущности Task)
• Стартовый шаблон Express-приложения с файловой структурой src/routes, src/controllers, src/middleware
• Список рекомендуемых пакетов (express, morgan, cors, pino)
Инструменты: Express.js, Postman/Insomnia, VS Code, логгер Pino/Morgan
Пошаговое задание:

1. Инициализировать Express-приложение, подключить парсер JSON и CORS
2. Создать логгер запросов, настроить формат вывода (метод, путь, статус, время)
3. Реализовать маршрутизатор с разделением на контроллеры (обработка HTTP) и сервисы (бизнес-логика)
4. Написать обработчики для GET, POST, PUT, DELETE с возвратом структурированных ответов
5. Добавить глобальный обработчик ошибок через app.use((err, req, res, next) => ...)
6. Протестировать все эндпоинты через коллекцию Postman, проверить логи

Итоговый результат:
.zip Архив серверного кода с разделением слоёв / Фамилия_ПР6_ExpressAPI
.json Экспортированная коллекция Postman / Фамилия_ПР6_КоллекцияТестов

Критерии оценки (чек-лист):
• [ ] Код разделён на маршруты, контроллеры и сервисы, отсутствует дублирование логики
• [ ] Middleware применяются в корректной последовательности (логирование → парсинг → роуты → ошибки)
• [ ] Все ответы соответствуют единому формату JSON с полями success, data, error
• [ ] Глобальный обработчик ошибок перехватывает исключения и не «роняет» сервер

**Тема 7. Валидация входящих данных и защита серверных эндпоинтов**
Цель практики: Обеспечить строгую проверку входящих запросов и безопасное управление конфигурациями.
Задачи: Настроить Zod-схемы, маппировать ошибки в HTTP-статусы, изолировать секреты.
Входные данные:
• Спецификация полей для сущности User (email, password, age, role)
• Пример Zod-схемы с кастомными правилами и сообщениями об ошибках
• Шаблон .env.example с переменными окружения
Инструменты: Zod, dotenv, Express, Postman, пакет @asteasolutions/zod-to-openapi (опционально)
Пошаговое задание:

1. Описать Zod-схему для входящего тела запроса, добавить трансформации и кастомные проверки
2. Создать middleware validate(schema), останавливающий цепочку при ошибках валидации
3. Сопоставить ошибки схемы с кодами 400 (синтаксис) и 422 (логика), вернуть массив деталей
4. Вынести конфигурации (порт, секреты, URL БД) в .env, подключить dotenv
5. Добавить проверку обязательных переменных при старте сервера, блокировать запуск при их отсутствии
6. Протестировать валидацию на корректных, неполных и вредоносных данных, зафиксировать ответы

Итоговый результат:
.ts Модуль валидации и схема Zod / Фамилия_ПР7_ВалидацияДанных
.txt Отчёт о тестировании граничных случаев и ответов сервера / Фамилия_ПР7_ТестВалидации

Критерии оценки (чек-лист):
• [ ] Схема покрывает все обязательные поля, типы и форматы
• [ ] Ошибки валидации возвращаются в читаемом формате без раскрытия внутренней логики

• [ ] Секреты и конфигурации изолированы в .env, не попадают в репозиторий
• [ ] Сервер корректно реагирует на SQL/JS-инъекции в строковых полях

**Тема 8. Интеграция клиентской и серверной частей в единый цикл разработки**
Цель практики: Связать фронтенд и бэкенд в локальном окружении, обеспечить согласованность типов и данных.
Задачи: Настроить проксирование, синхронизировать TypeScript-интерфейсы, реализовать полный CRUD-флоу.
Входные данные:
• Готовый сервер из ПР6-ПР7 и фронтенд из ПР2-ПР4
• Конфигурация vite.config.ts для настройки server.proxy
• Интерфейсы TypeScript для запросов и ответов сервера
Инструменты: Vite, React, Express, TypeScript, браузерные DevTools
Пошаговое задание:

1. Настроить Vite proxy для перенаправления запросов /api/* на локальный порт сервера
2. Синхронизировать типы данных: экспортировать интерфейсы из бэкенда или использовать общую библиотеку типов
3. Подключить сервисный слой фронтенда к реальным эндпоинтам, удалить мок-данные
4. Реализовать форму создания/редактирования с клиентской и серверной валидацией
5. Настроить обработку таймаутов и отмену запросов через AbortController
6. Провести сквозное тестирование: создание → отображение → изменение → удаление, зафиксировать логи

Итоговый результат:
.git Ссылка на рабочую ветку с интеграцией / Фамилия_ПР8_ИнтеграцияFE-BE
.pdf Скриншоты состояний приложения при успешном и ошибочном сценариях / Фамилия_ПР8_ДемонстрацияФлоу

Критерии оценки (чек-лист):
• [ ] Проксирование настроено корректно, отсутствуют ошибки CORS при локальной разработке
• [ ] Типы данных согласованы между клиентом и сервером, компиляция проходит без предупреждений
• [ ] Формы блокируют отправку при клиентских ошибках, серверные ошибки отображаются в UI
• [ ] Сквозный CRUD-сценарий выполняется без перезагрузки страницы и потери состояния

**Тема 9. Реляционные базы данных и проектирование схемы хранения**
Цель практики: Спроектировать нормализованную схему данных и обеспечить корректные связи между таблицами.
Задачи: Создать ER-диаграмму, написать DDL-скрипты, настроить индексы и транзакции.
Входные данные:
• Описание предметной области (сущности, связи 1:N и M:N, бизнес-правила целостности)
• Доступ к облачному PostgreSQL (Neon/Vercel Postgres) или локальному контейнеру
• Шаблон отчёта по анализу производительности запросов Инструменты: dbdiagram.io / Draw.io, pgAdmin / DBeaver, PSQL CLI
Пошаговое задание:

1. Выделить сущности и связи, нормализовать таблицу до третьей нормальной формы
2. Построить ER-диаграмму, указать первичные и внешние ключи, типы данных и ограничения NOT NULL
3. Написать SQL-скрипты создания таблиц, добавить индексы для часто используемых полей поиска
4. Выполнить скрипты в целевой базе, проверить целостность при вставке связанных данных
5. Проанализировать план выполнения запроса через EXPLAIN ANALYZE, подтвердить использование индексов
6. Документировать ограничения и стратегии резервного копирования

Итоговый результат:

.pdf ER-диаграмма и SQL-скрипты инициализации / Фамилия_ПР9_СхемаБД
.txt Вывод EXPLAIN ANALYZE с комментариями по оптимизации / Фамилия_ПР9_ПланЗапросов

Критерии оценки (чек-лист):
• [ ] Схема нормализована, отсутствуют избыточные данные и аномалии обновления
• [ ] Внешние ключи настроены корректно, каскадные удаления/обновления обоснованы
• [ ] Индексы созданы для полей фильтрации и сортировки, план запроса подтверждает их использование
• [ ] Скрипты воспроизводимы, транзакции защищены от частичного применения

**Тема 10. ORM и миграции: управление базой данных через код**
Цель практики: Автоматизировать управление схемой БД и генерацию типобезопасных запросов через ORM.
Задачи: Настроить Prisma/Drizzle, создать миграции, реализовать CRUD через генерированный клиент.
Входные данные:
• Схема из ПР9 (в формате SQL или описательном виде)
• Установленный пакет prisma или drizzle-kit
• Шаблон конфигурации подключения к базе
Инструменты: Prisma CLI / Drizzle ORM, PostgreSQL, VS Code, терминал
Пошаговое задание:
1. Инициализировать ORM-проект, настроить DATABASE_URL и путь к файлу схемы
2. Описать модели данных в синтаксисе ORM, указать связи, уникальные поля и индексы
3. Сгенерировать клиент и выполнить команду создания миграции dev, проверить файл миграции
4. Применить миграции к локальной и облачной среде, выполнить seed-скрипт заполнения тестовыми данными
5. Переписать эндпоинты сервера на использование ORM-клиента, заменить ручные SQL-запросы
6. Проверить типобезопасность: убедиться, что IDE подсвечивает ошибки доступа к несуществующим полям

Итоговый результат:
.prisma / .ts Файл схемы ORM и история миграций / Фамилия_ПР10_ORMМиграции
.zip Обновлённый серверный код с ORM-клиентом / Фамилия_ПР10_ИнтеграцияORM

Критерии оценки (чек-лист):
• [ ] Схема ORM полностью отражает реляционную структуру без потери связей
• [ ] Миграции применяются и откатываются без ошибок, seed заполняет базу валидными данными
• [ ] Ручные SQL-запросы заменены на типобезопасные вызовы ORM-клиента
• [ ] IDE предоставляет автодополнение и проверяет типы полей на этапе компиляции

**Тема 11. Аутентификация и авторизация веб-приложений**
Цель практики: Реализовать безопасный механизм входа, хранения сессий и ролевого доступа.
Задачи: Настроить хэширование паролей, выдачу JWT, защищённые куки и проверку прав на сервере и клиенте.
Входные данные:
• Спецификация ролей (user, admin) и защищённых маршрутов
• Пакеты bcrypt/argon2, jsonwebtoken, cookie-parser
• Пример реализации refresh-токен flow
Инструменты: Node.js, Express, React, jwt, Postman, браузерные DevTools (Application tab)
Пошаговое задание:
1. Реализовать эндпоинты /register и /login с хэшированием паролей перед сохранением
2. Настроить генерацию JWT (access + refresh), подписать токен секретным ключом с ограниченным сроком

3. Настроить передачу токенов через httpOnly, Secure, SameSite=Strict cookie
4. Создать серверный middleware verifyToken, извлекающий и проверяющий подпись токена
5. Реализовать клиентский хук useAuth для проверки статуса сессии и условного рендеринга UI
6. Протестировать сценарии: успешный вход, истечение токена, доступ без прав, корректный logout

Итоговый результат:
.ts Модули аутентификации и авторизации (сервер + клиент) / Фамилия_ПР11_AuthFlow
.json Коллекция Postman с запросами регистрации, входа и проверки доступа / Фамилия_ПР11_ТестыAuth

Критерии оценки (чек-лист):
• [ ] Пароли хэшированы с использованием криптостойкого алгоритма и соли
• [ ] Токены хранятся в защищённых куках, недоступны для клиентских скриптов
• [ ] Серверный middleware корректно проверяет подпись и срок действия токена
• [ ] Клиентский интерфейс реагирует на 401/403 статусы, redirect происходит без утечки данных

**Тема 12. Безопасность веб-приложений и защита от типовых атак**
Цель практики: Устранить уязвимости из OWASP Top 10 и настроить защитные заголовки.
Задачи: Провести аудит зависимостей, внедрить rate limiting, CSP, защиту от XSS/CSRF/SQLi.
Входные данные:
• Готовый API из ПР6-ПР11
• Отчёт npm audit или yarn audit
• Список рекомендуемых заголовков безопасности
Инструменты: helmet, express-rate-limit, npm audit, OWASP ZAP (базовый скан), Postman
Пошаговое задание:

1. Запустить аудит зависимостей, обновить критические уязвимости или зафиксировать исключения
2. Подключить helmet, настроить заголовки X-Content-Type-Options, Referrer-Policy, Strict-Transport-Security
3. Реализовать Content-Security-Policy с белым списком доверенных источников скриптов
4. Настроить rate limiting на эндпоинты аутентификации и публичные API (макс. 5 запросов/мин)
5. Проверить экранирование вывода на фронтенде, внедрить защиту от CSRF при использовании форм
6. Провести ручные тесты на XSS и инъекции, зафиксировать блокировку вредоносных запросов

Итоговый результат:
.ts Конфигурационный файл безопасности и middleware / Фамилия_ПР12_SecurityConfig
.pdf Отчёт о проведённом аудите и тестировании уязвимостей / Фамилия_ПР12_АудитБезопасности
Критерии оценки (чек-лист):
• [ ] Зависимости обновлены, критические уязвимости устранены или задокументированы
• [ ] Заголовки безопасности настроены, CSP ограничивает выполнение неавторизованных скриптов
• [ ] Rate limiting срабатывает при превышении лимита, возвращает корректный статус 429
• [ ] Ввод пользователя экранируется, инъекции блокируются на уровне ORM и валидации

**Тема 13. Бессерверные функции и оптимизация под Vercel**
Цель практики: Адаптировать серверный код под ограничения serverless-среды и Edge Runtime.
Задачи: Перенести маршруты в формат api/, оптимизировать пакет, протестировать холодный старт.
Входные данные:
• Готовый Express-сервер из предыдущих практик

• Документация @vercel/node и ограничения Vercel Functions
• Локальный эмулятор vercel dev
Инструменты: Vercel CLI, @vercel/node, Node.js, терминал, таймеры производительности
Пошаговое задание:

1. Преобразовать маршруты Express в формат файлов api/*.ts, адаптировать сигнатуры (req, res) под Vercel
2. Удалить неиспользуемые зависимости, настроить .vercelignore для исключения dev-файлов
3. Реализовать переиспользование соединений с БД вне обработчика функции для снижения нагрузки
4. Замерить время холодного старта при первом вызове функции, сравнить с последующими вызовами
5. Проверить совместимость с Edge Runtime (если применимо), вынести тяжёлые вычисления в фоновые задачи
6. Запустить локально через vercel dev, убедиться в корректной работе всех эндпоинтов

Итоговый результат:
.zip Архив серверных функций в структуре api/ / Фамилия_ПР13_ServerlessФункции
.txt Логи замеров времени отклика и рекомендации по оптимизации / Фамилия_ПР13_ТестХолодногоСтарта

Критерии оценки (чек-лист):
• [ ] Код адаптирован под stateless-среду, отсутствуют глобальные переменные состояния
• [ ] Зависимости минимизированы, .vercelignore исключает тестовые и dev-файлы
• [ ] Соединения с БД переиспользуются между вызовами функции
• [ ] Холодный старт не превышает допустимых лимитов платформы, замеры задокументированы

**Тема 14. Деплой, предварительные среды и управление релизами**
Цель практики: Настроить автоматическое развёртывание и управление окружениями через Vercel.
Задачи: Подключить Git-репозиторий, настроить vercel.json, продвинуть переменные окружения, проверить HTTPS.
Входные данные:
• Готовый фронтенд и serverless-функции, закоммиченные в Git
• Доступ к дашборду Vercel и аккаунту GitHub/GitLab
• Список переменных для development, preview и production
Инструменты: Vercel Dashboard, Vercel CLI, Git, браузер
Пошаговое задание:

1. Создать проект в Vercel, подключить репозиторий, выбрать framework preset (Vite/Next.js)
2. Настроить vercel.json: указать пути сборки, перенаправления, кастомные заголовки кэширования
3. Добавить переменные окружения для всех трёх сред через интерфейс или CLI, проверить изоляцию
4. Создать feature-ветку, запушить изменения, дождаться автоматического развёртывания preview-окружения
5. Проверить работу приложения по временному URL, убедиться в наличии HTTPS и корректном отображении контента
6. Выкатить стабильную версию в production, зафиксировать время деплоя и статус сборки

Итоговый результат:
.json Файл конфигурации деплоя и скриншоты настроек окружений / Фамилия_ПР14_VercelConfig
.log Логи сборки и ссылка на рабочий preview-URL / Фамилия_ПР14_ДеплойЛог

Критерии оценки (чек-лист):
• [ ] vercel.json корректно задаёт пути сборки и правила маршрутизации
• [ ] Переменные окружения изолированы по средам, секреты не попадают в лог сборки
• [ ] Preview-окружение разворачивается автоматически при пуше в ветку

• [ ] Production-ссылка работает по HTTPS, контент отдаётся без ошибок 4xx/5xx

**Тема 15. Автоматизация сборки и контроль качества кода**
Цель практики: Реализовать CI/CD-конвейер для проверки качества кода до деплоя.
Задачи: Написать GitHub Actions workflow, добавить Husky-хуки, кэшировать зависимости, блокировать мёрж при падениях.
Входные данные:
• Репозиторий проекта с настроенным ESLint, Prettier и тестами (Jest/Vitest)
• Шаблон .github/workflows/ci.yml • Документация по lint-staged и husky
Инструменты: GitHub Actions, Husky, lint-staged, VS Code, терминал
Пошаговое задание:

1. Создать файл ci.yml с триггером push и pull_request, задать матрицу версий Node.js
2. Добавить шаги: checkout, установка зависимостей, линтинг, форматирование, запуск тестов, сборка
3. Настроить кэширование node_modules и директории сборки для ускорения повторных запусков
4. Инициализировать Husky, настроить lint-staged для проверки только изменённых файлов перед коммитом
5. Протестировать конвейер: запушить коммит с ошибкой линтера и коммит с прошедшими тестами
6. Настроить правила защиты ветки: запретить мёрж при статусе failure в CI

Итоговый результат:
.yml Файл конвейера CI/CD и конфигурация Husky / Фамилия_ПР15_CICDКонвейер
.pdf Скриншоты успешного и упавшего запуска пайплайна / Фамилия_ПР15_СтатусыСборки
Критерии оценки (чек-лист):
• [ ] Конвейер запускается автоматически при пуше и pull-request, покрывает lint, test, build
• [ ] Кэширование зависимостей работает, время сборки стабильно сокращается
• [ ] Pre-commit хуки блокируют коммиты с ошибками линтера или форматирования
• [ ] Защита ветки активирована, мёрж невозможен при падении тестов

**Тема 16. Мониторинг, аналитика и поддержка промышленного окружения**
Цель практики: Настроить отслеживание ошибок, сбор метрик производительности и ротацию логов.
Задачи: Интегрировать Sentry и Vercel Analytics, проанализировать Core Web Vitals, составить отчёт о стабильности.
Входные данные:
• Развёрнутое production-приложение на Vercel
• Доступ к дашбордам Sentry и Vercel Analytics
• Шаблон отчёта по наблюдаемости системы
Инструменты: Sentry, Vercel Analytics, Lighthouse CI, браузерные DevTools, текстовый процессор
Пошаговое задание:

1. Создать проект в Sentry, установить SDK во фронтенд и serverless-функции, настроить source maps
2. Сгенерировать тестовую ошибку, проверить её появление в дашборде, добавить контекст пользователя
3. Подключить Vercel Analytics, активировать сбор Core Web Vitals (LCP, INP, CLS)
4. Провести аудит производительности через Lighthouse, зафиксировать метрики до и после оптимизации бандла
5. Настроить алерты на рост 5xx ошибок и аномальное время отклика функций
6. Составить итоговый отчёт: карта метрик, план реагирования на инциденты, рекомендации по ротации логов

Итоговый результат:
.pdf Отчёт по мониторингу, аналитике и плану стабильности / Фамилия_ПР16_МониторингProduction

.json Экспорт конфигурации алертов и скриншоты дашбордов / Фамилия_ПР16_КонфигАлертов

Критерии оценки (чек-лист):
• [ ] Ошибки фронтенда и сервера корректно передаются в Sentry с развёрнутым стек-трейсом
• [ ] Core Web Vitals собираются в реальном времени, метрики соответствуют рекомендациям Google
• [ ] Алерты настроены на критические пороги, уведомления приходят в указанный канал
• [ ] Отчёт содержит измеримые показатели, план действий при инцидентах и стратегию ротации логов

### 5.4. Перечень вопросов промежуточной аттестации по дисциплине

**Темы курсовых работ**

1. Платформа бронирования коворкинг-пространств
2. Система управления задачами для распределённых команд
3. Маркетплейс цифровых активов и графических ресурсов
4. Система управления обучением с трекингом прогресса
5. Агрегатор тарифов и услуг с калькулятором стоимости
6. Дашборд мониторинга данных IoT-датчиков
7. Сервис онлайн-записи к специалистам
8. Система складского учёта и инвентаризации
9. Платформа peer-to-peer обмена профессиональными навыками
10. Аналитическая панель для социальных сетей
11. Персональный финансовый трекер с визуализацией бюджетов
12. Портал организации волонтёрских мероприятий
13. Цифровой каталог объектов культурного наследия
14. Система онлайн-тестирования и оценки компетенций
15. Афиша событий с персонализированными рекомендациями
16. CRM-система для управления проектами и клиентами
17. Веб-приложение для совместного редактирования документов
18. Сервис подбора туристических маршрутов и бронирования гидов
19. Headless CMS для управления медиаконтентом
20. Трекер привычек и поведенческих метрик
21. Платформа управления хакатонами и командными проектами
22. Сервис аренды профессионального оборудования
23. Корпоративная база знаний с полнотекстовым поиском
24. Система анонимного онлайн-голосования и сбора обратной связи
25. Панель управления устройствами умного дома (симуляция API)

Для написания курсовой работы студент последовательно выполняет ПР1–ПР16, каждый раз развивая один и тот же репозиторий:
ПР1 → рисует схему Клиент → Vercel Edge → Serverless API → Neon DB, утверждает в зачёт.
ПР2–ПР4 → создаёт колонки, карточки, drag-and-drop UI, подключает кэш задач.
ПР5–ПР8 → разворачивает /boards, /tasks, /comments, настраивает валидацию, соединяет фронтенд с бэкендом.
ПР9–ПР10 → проектирует таблицы users, boards, tasks (связь M:N), применяет миграции, заполняет seed-данными.
ПР11–ПР12 → добавляет регистрацию, роли admin/member, защищает эндпоинты, настраивает CSP и rate-limit.
ПР13–ПР16 → переносит логику в api/, деплоит в Vercel, настраивает CI/CD, подключает мониторинг, защищает ветку main.

**Вопросы к экзамену:**

1. Перечислите компоненты клиент-серверной модели распределённых вычислений.
2. Определите различия между одностраничными и многостраничными приложениями.
3. Назовите факторы, влияющие на время холодного старта бессерверных функций.
4. Сформулируйте правила вызова хуков в React.
5. Перечислите преимущества однонаправленного потока данных.
6. Определите назначение интерфейсов TypeScript в функциональных компонентах.
7. Опишите принцип работы клиентской маршрутизации без перезагрузки страницы.
8. Перечислите атрибуты защищённых маршрутов в одностраничных приложениях.
9. Определите назначение паттерна макетов при построении пользовательских интерфейсов.
10. Определите разницу между оптимистичным и консервативным обновлением интерфейса.
11. Перечислите методы принудительной очистки кэша в библиотеках управления состоянием.
12. Назовите причины возникновения гоночных состояний при параллельных асинхронных запросах.
13. Опишите структуру цикла обработки событий в среде выполнения Node.js.
14. Перечислите различия между модульными системами CommonJS и ECMAScript.
15. Определите назначение файла package.json в управлении зависимостями проекта.
16. Перечислите уровни архитектуры серверного приложения на фреймворке Express.js.
17. Определите назначение промежуточного программного обеспечения в цепочке обработки запросов.
18. Назовите принципы централизованной обработки ошибок в прикладных интерфейсах.
19. Перечислите этапы валидации входящих запросов перед выполнением бизнес-логики.
20. Определите назначение санитайзинга строк в серверной разработке.
21. Назовите стандартные коды состояния протокола HTTP для ошибок валидации.
22. Опишите механизм проксирования запросов в среде разработки.
23. Перечислите риски рассогласования типов при интеграции клиентской и серверной частей.
24. Определите назначение двойной валидации данных в клиент-серверных приложениях.
25. Перечислите свойства транзакций согласно модели ACID.
26. Определите назначение первичных и внешних ключей в реляционных таблицах.
27. Назовите последствия нормализации данных для избыточности и целостности.
28. Опишите принцип объектно-реляционного отображения.
29. Перечислите этапы жизненного цикла миграций базы данных.
30. Определите назначение генерации типизированного клиента в ORM.
31. Определите различия между веб-токенами JWT и серверными сессиями.
32. Перечислите атрибуты безопасного хранения авторизационных cookie-файлов.
33. Назовите этапы потока выдачи и обновления токенов доступа.
34. Перечислите основные векторы атак из рейтинга OWASP Top 10.
35. Определите назначение политики безопасности контента.
36. Назовите механизмы защиты форм от межсайтовой подделки запросов.
37. Опишите архитектурные особенности бессерверных вычислений.
38. Перечислите технические ограничения среды выполнения на границе сети.
39. Назовите методы оптимизации времени холодного старта функций.
40. Определите назначение файла конфигурации vercel.json в процессе развёртывания.
41. Перечислите этапы автоматического создания предварительных окружений.
42. Назовите принципы инвалидации кэша сети доставки контента при обновлениях.
43. Опишите этапы конвейера непрерывной интеграции и доставки.
44. Перечислите функции хуков перед фиксацией изменений в репозитории.
45. Определите назначение кэширования зависимостей в автоматизированных сборках.
46. Перечислите компоненты принципа наблюдаемости производственной системы.
47. Определите назначение метрик Core Web Vitals в аналитике производительности.
48. Назовите методы ротации журналов событий в облачных средах.

## Раздел 6. Оценочные средства промежуточной аттестации (с ключами)

**1. Укажите один правильный ответ.** Какое архитектурное ограничение характерно для serverless-среды выполнения?
а) Постоянное хранение состояния в памяти между вызовами
б) Автоматическое горизонтальное масштабирование и холодный старт
в) Обязательное использование Docker-контейнеров
г) Синхронная обработка всех запросов в одном потоке
**Правильный ответ: б)**

**2. Установите соответствие** между хуками React и их назначением:

| **Хук** | **Назначение** |
|---|---|
| А) useState | 1) Мемоизация результата вычислений для оптимизации рендера |
| Б) useEffect | 2) Управление локальным состоянием компонента |
| В) useMemo | 3) Выполнение побочных эффектов (запросы, подписки) |
| Г) useCallback | 4) Мемоизация самой функции для предотвращения лишних ререндеров |

**Правильный ответ: А-2, Б-3, В-1, Г-4**

**3. Разместите по порядку** этапы обработки асинхронного кода в цикле событий Node.js (от начального к завершающему):
1. Выполнение колбэков из очереди макрозадач (timers, I/O callbacks)
2. Выполнение синхронного кода в основном потоке
3. Выполнение колбэков из очереди микрозадач (Promises, process.nextTick)
4. Переход к следующей итерации цикла событий

**Правильный ответ: 2, 3, 1, 4**

**4. Установите соответствие** между HTTP-статусами и ситуациями валидации:

| **Ситуация** | **Статус** |
|---|---|
| А) Ошибка синтаксиса запроса (неверный JSON) | 1) 422 Unprocessable Entity |
| Б) Данные валидны по типу, но нарушают бизнес-логику | 2) 400 Bad Request |
| В) Ресурс не найден по указанному ID | 3) 401 Unauthorized |
| Г) Отсутствует или просрочен токен авторизации | 4) 404 Not Found |

**Правильный ответ: А-2, Б-1, В-4, Г-3**

**5. Выберите два правильных ответа.** Какие преимущества предоставляет библиотека TanStack Query (React Query) по сравнению с ручным управлением useEffect?
а) Автоматическое кэширование и фоновое обновление данных
б) Встроенная поддержка CSS-стилизации компонентов
в) Упрощённая работа с состояниями загрузки, ошибок и повторных запросов
г) Замена необходимости использовать TypeScript в проекте
**Правильный ответ: а) в)**

**6. Укажите один правильный ответ.** Что произойдёт, если в Express-приложении middleware для обработки ошибок не будет размещён после всех маршрутов?
а) Сервер автоматически перезагрузится при каждом запросе
б) Ошибки в промежуточных маршрутах будут проигнорированы и вернут стандартный 500
в) Все запросы будут перенаправлены на статическую страницу
г) Приложение не сможет запускаться без указания порта

**Правильный ответ: б)**

**7. Запишите название конфигурационного поля** (на английском), которое в файле vite.config.ts позволяет перенаправлять запросы /api/* на локальный порт сервера для обхода ограничений браузера. server._____
**Правильный ответ: proxy**

**8. Установите соответствие** между свойствами транзакций (ACID) и их определением:

| **Свойство** | **Определение** |
|---|---|
| А) Атомарность | 1) Гарантирует, что результаты транзакции сохраняются после завершения |
| Б) Согласованность | 2) Транзакция выполняется целиком или не выполняется вовсе |
| В) Изолированность | 3) Параллельные транзакции не влияют друг на друга |
| Г) Долговечность | 4) База данных переходит из одного корректного состояния в другое |

**Правильный ответ: А-2, Б-4, В-3, Г-1**

**9. Разместите по порядку** шаги безопасного потока аутентификации с использованием JWT (от запроса пользователя до получения данных):

1. Сервер проверяет логин/пароль и генерирует access- и refresh-токены
2. Клиент отправляет запрос на эндпоинт /login с учётными данными
3. Клиент сохраняет токены в httpOnly cookies
4. Сервер валидирует signature токена и возвращает защищённые данные

**Правильный ответ: 2, 1, 3, 4**

**10. Выберите два правильных ответа.** Какие задачи решают миграции в ORM (Prisma/Drizzle)?
а) Автоматическая генерация клиентских интерфейсов TypeScript
б) Версионирование схемы базы данных и возможность отката изменений
в) Замена необходимости писать SQL-запросы вручную в продакшене
г) Контроль изменений структуры таблиц без ручного выполнения DDL-скриптов
**Правильный ответ: б) г)**

**11. Укажите один правильный ответ.** Какой HTTP-заголовок, настраиваемый через библиотеку helmet, ограничивает источники загрузки скриптов, стилей и изображений для защиты от внедрения кода?
а) Strict-Transport-Security
б) Content-Security-Policy
в) X-Content-Type-Options
г) Access-Control-Allow-Origin
**Правильный ответ: б)**

**12. Установите соответствие** между ограничениями serverless-среды и методами их оптимизации:

| **Ограничение** | **Метод оптимизации** |
|---|---|
| А) Холодный старт при первом вызове | 1) Переиспользование соединений с БД вне обработчика |
| Б) Лимит времени выполнения (timeout) | 2) Ленивая загрузка модулей и минимизация бандла |
| В) Отсутствие постоянного состояния | 3) Вынос тяжёлых фоновых задач в асинхронные очереди |
| Г) Ограничение памяти на инстансе | 4) Кеширование конфигураций и подключение к managed-сервисам |

**Правильный ответ: А-2, Б-3, В-1, Г-4**

**13. Разместите по порядку** этапы стандартного CI/CD-пайплайна для веб-приложения (от первого к последнему):

1. Сборка production-артефактов и публикация на хостинг
2. Запуск модульных и интеграционных тестов
3. Проверка стиля кода линтером и форматирование
4. Получение кода из репозитория и установка зависимостей

**Правильный ответ: 4, 3, 2, 1**

**14. Выберите два правильных ответа.** Какие функции выполняет файл vercel.json в процессе развёртывания?
а) Настройка перенаправлений (rewrites/redirects) и кастомных заголовков
б) Компиляция исходного кода TypeScript в JavaScript
в) Определение правил маршрутизации и путей сборки
г) Управление переменными окружения для разных сред
**Правильный ответ: а) в)**

**15. Укажите один правильный ответ.** Какая метрика из Core Web Vitals измеряет визуальную стабильность страницы и отслеживает неожиданные сдвиги макета при загрузке?
а) LCP (Largest Contentful Paint)
б) FID (First Input Delay) / INP (Interaction to Next Paint)
в) CLS (Cumulative Layout Shift)
г) TTFB (Time to First Byte)
**Правильный ответ: в)**

**16. Установите соответствие** между инструментами наблюдаемости и их основной функцией:

| **Инструмент** | **Функция** |
|---|---|
| А) Sentry | 1) Сбор метрик скорости загрузки и пользовательского поведения |
| Б) Vercel Analytics | 2) Отслеживание и классификация исключений на клиенте и сервере |
| В) Lighthouse CI | 3) Автоматический аудит производительности и SEO в пайплайне |
| Г) Logflare / CloudWatch | 4) Агрегация и поиск структурированных журналов событий |

**Правильный ответ: А-2, Б-1, В-3, Г-4**

**17. Выберите два правильных ответа.** Какие состояния интерфейса необходимо обрабатывать при асинхронном запросе данных?
а) Состояние загрузки (loading)
б) Состояние компиляции TypeScript
в) Состояние ошибки сервера или сети (error)
г) Состояние холодного старта сервера
**Правильный ответ: а) в)**

**18. Укажите один правильный ответ.** Какую задачу решает промежуточное ПО cors в Express-приложении?
а) Сжимает ответы для ускорения передачи данных
б) Позволяет браузеру выполнять запросы к API с другого домена
в) Хэширует пароли пользователей перед записью в БД
г) Автоматически генерирует документацию Swagger
**Правильный ответ: б)**

**19. Запишите термин** (с большой буквы, в именительном падеже), обозначающий уникальный идентификатор каждой записи в реляционной таблице. _____ — гарантирует уникальность строк и ускоряет поиск.
**Правильный ответ: Первичный ключ**

**20. Разместите по порядку** шаги работы с миграциями в ORM (от первого к последнему):
1. Применение миграции к целевой базе данных
2. Изменение схемы моделей в конфигурационном файле ORM
3. Генерация файла миграции на основе diff-схемы
4. Откат миграции при обнаружении критических ошибок

**Правильный ответ: 2, 3, 1, 4**

**21. Укажите один правильный ответ.** Почему хранение JWT в localStorage считается менее безопасным, чем в httpOnly cookie?
а) localStorage не поддерживает хранение строк длиннее 5 МБ
б) Скрипты на странице имеют доступ к localStorage, что упрощает кражу токена при XSS-атаке
в) localStorage автоматически очищается при перезагрузке страницы
г) Браузеры блокируют запросы с токенами из localStorage
**Правильный ответ: б)**

**22. Установите соответствие** между библиотеками и их назначением в стеке Node.js:

| **Библиотека** | **Назначение** |
|---|---|
| А) Zod | 1) Описание схем и валидация входящих данных |
| Б) dotenv | 2) Загрузка переменных окружения из .env |
| В) bcrypt | 3) Криптографическое хэширование паролей |
| Г) morgan | 4) Логирование HTTP-запросов в консоль |

**Правильный ответ: А-1, Б-2, В-3, Г-4**

**23. Выберите два правильных ответа.** Какие особенности отличают Vercel Functions от традиционного Express-сервера на выделенном хостинге?
а) Автоматическое масштабирование до нуля при отсутствии трафика
б) Возможность использования глобальных переменных для хранения постоянного состояния
в) Ограничение времени выполнения и объёма оперативной памяти на инстансе
г) Обязательное ручное управление пулами соединений к базе данных
**Правильный ответ: а) в)**

**24. Запишите термин** (на английском, с заглавной буквы), обозначающий компонент в React Router, который отображает содержимое дочерних маршрутов внутри родительского макета. _____ — точка вставки вложенных страниц.
**Правильный ответ: Outlet**

**25. Укажите один правильный ответ.** Какую проблему решает синхронизация TypeScript-интерфейсов между фронтендом и бэкендом?
а) Ускоряет загрузку статических файлов
б) Предотвращает рассогласование типов данных на этапе компиляции
в) Автоматически оптимизирует SQL-запросы к базе данных
г) Заменяет необходимость валидации данных на сервере
**Правильный ответ: б)**

## Раздел 7. Перечень учебной литературы, необходимой для освоения дисциплины

### 7.1. Обязательная литература


1. Адамс, Д. Р. Основы работы с XHTML и CSS : учебник / Д. Р. Адамс, К. С. Флойд. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2025. — 567 с. — ISBN 978-5-4497-0907-3. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/146372.html

2. Борисов, Р. С. Информатика и программирование : учебное пособие для среднего профессионального образования / Р. С. Борисов, А. С. Скотченко. — Москва : Российский государственный университет правосудия, 2023. — 334 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/133635.html

3. Боровков, В. А. Информатика и программирование. Текстовый редактор MS Word : учебное пособие для СПО / В. А. Боровков, С. М. Колмогорова. — Москва : Ай Пи Ар Медиа, 2023. — 136 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/129311.html

4. Кашевский, П. А. Типографика : учебное пособие / П. А. Кашевский. — Минск : Республиканский институт профессионального образования (РИПО), 2022. — 298 с. — ISBN 978-985-895-072-9. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/134164.html

5. Киренберг, А.Г. Основы информатики, организации ЭВМ, вычислительных и информационных систем : учебное пособие / А. Г. Киренберг, В. О. Коротин. — Кемерово : Кузбасский государственный технический университет имени Т.Ф. Горбачева, 2023. — 165 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/135106.html

6. Кучерова, А. В. Типографика: основы верстки : учебное пособие / А. В. Кучерова. — Омск : Омский государственный технический университет, 2023. — 97 с. — ISBN 978-5-8149-3649-3. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/140872.html

7. Кучерова, А. В. Типографика : учебное пособие / А. В. Кучерова. — Омск : Омский государственный технический университет, 2022. — 144 с. — ISBN 978-5-8149-3460-4. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/131234.html

8. Петракова, Н. В. Веб-программирование. Основы HTML : учебное пособие для СПО / Н. В. Петракова. — Саратов : Профобразование, 2026. — 49 с. — ISBN 978-5-4488-2823-2. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/160948.html

9. Савельев, А. О. HTML5. Основы клиентской разработки : учебное пособие / А. О. Савельев, А. А. Алексеев. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. — 270 с. — ISBN 978-5-4497-2398-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/133910.html

10. Сычев, А. В. Web-технологии : учебное пособие / А. В. Сычев. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. — 407 с. — ISBN 978-5-4497-2429-8. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/133914.html

11. Фролов, А. Б. Web-сайт. Разработка, создание, сопровождение : учебное пособие / А. Б. Фролов, И. А. Нагаева, И. А. Кузнецов ; под редакцией И. А. Нагаевой. — 2-е изд. — Саратов : Вузовское образование, 2024. — 355 с. — ISBN 978-5-4487-1025-4. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/142801.html


### 7.2. Дополнительная литература

1. Бурьков, Д. В. Информатика и программирование : учебное пособие / Д. В. Бурьков. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2022. — 215 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/131449.html

2. Кисленко, Н. П. Информатика и программирование : учебное пособие / Н. П. Кисленко, И. Н. Мухина. — Новосибирск : Новосибирский государственный архитектурно-строительный университет (Сибстрин), ЭБС АСВ, 2022. — 105 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/129325.html

3. Кузьменко, И. П. Информатика и программирование : учебник для иностранных студентов / И. П. Кузьменко, С. В. Богданова. — Ставрополь : Ставропольский государственный аграрный университет, 2022. — 184 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/129581.html

4. Моренкова, О. И. Введение в курс информатики : учебное пособие / О. И. Моренкова, Т. И. Парначева. — Новосибирск : Сибирский государственный университет телекоммуникаций и информатики, 2020. — 158 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/117092.html

### 7.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет»

1. http: //elibrary.ru/
2. https://habr.com/
3. Справочно-правовая система «Консультант Плюс».
4. «Гарант»
5. ПО для организации конференций

## Раздел 8. Материально-техническая база и информационные технологии

Материально-техническое обеспечение дисциплины «**Разработка клиент-серверных приложений**» включает в себя учебные аудитории для проведения лекционных, лабораторных занятий, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, самостоятельной работы обучающихся.

Учебные аудитории укомплектованы специализированной мебелью и техническими средствами обучения. Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой с возможностью подключения к сети Интернет.

Дисциплина может реализовываться с применением дистанционных технологий обучения. Специфика реализации дисциплины с применением дистанционных технологий обучения устанавливается дополнением к рабочей программе. В части не противоречащей специфике, изложенной в дополнении к программе, применяется настоящая рабочая программа.

Материально-техническая база, необходимая для осуществления образовательного процесса по дисциплине с применением дистанционных образовательных технологий включает в себя:
Компьютерная техника, расположенная в учебном корпусе Института (ул.Качинцев, 63, кабинет Центра дистанционного обучения):

1) Intel i3 3.4Ghz\ОЗУ 4Gb\500GB\RadeonHD5450
2) Intel PENTIUM 2.9GHz\ОЗУ 4GB\500GB
3) личные электронные устройства (компьютеры, ноутбуки, планшеты и иное), а также средства связи преподавателей и студентов.

Информационные технологии, необходимые для осуществления образовательного процесса по дисциплине с применением дистанционных образовательных технологий включают в себя:

- система дистанционного обучения (СДО) (Learning Management System) (LMS) Moodle (Modular Object-Oriented Dynamic Learning Environment);
- электронная почта;
- система компьютерного тестирования;
- Цифровой образовательный ресурс IPR SMART;
- система интернет-связи skype;
- телефонная связь;
- ПО для проведения конференций.

Обучение обучающихся инвалидов и обучающихся с ограниченными возможностями здоровья осуществляется посредством применения специальных технических средств в зависимости от вида нозологии.

При проведении учебных занятий по дисциплине используются мультимедийные комплексы, электронные учебники и учебные пособия, адаптированные к ограничениям здоровья обучающихся.

Лекционные аудитории оборудованы мультимедийными кафедрами, подключенными к звуковым колонкам, позволяющими усилить звук для категории слабослышащих обучающихся, а также проекционными экранами, которые увеличивают изображение в несколько раз и позволяют воспринимать учебную информацию обучающимся с нарушениями зрения.

При обучении лиц с нарушениями слуха используется усилитель слуха для слабослышащих людей Cyber Ear модель HAP-40, помогающий обучаемым лучше воспринимать учебную информацию.

Обучающиеся с ограниченными возможностями здоровья, обеспечены печатными и электронными образовательными ресурсами (программы, учебники, учебные пособия, материалы для самостоятельной работы и т.д.) в формах, адаптированных к ограничениям их здоровья и восприятия информации:

**для лиц с нарушениями зрения:**

- в форме электронного документа;
- в форме аудиофайла;

**для лиц с нарушениями слуха:**

- в печатной форме;
- в форме электронного документа;

**для лиц с нарушениями опорно-двигательного аппарата:**
- в печатной форме;
- в форме электронного документа;
- в форме аудиофайла.

Программное обеспечение, используемое на занятиях**:**

- Операционная система Windows,
- Архиватор 7-zip,
- Система тестирования,
- Microsoft Office 2007,
- Антивирус Касперский 6,
- Консультант+,
- Виртуальная машина VirtualBox,
- Виртуальная машина VirtualPC,
- Internet Explorer.
-

## Раздел 9. Методические указания для обучающихся по освоению дисциплины

Для успешного усвоения материала дисциплины требуются значительное время, концентрация внимания и усилия: посещение лекционных занятий и конспектирование преподаваемого материала, работа с ним дома, самостоятельная проработка материала рекомендуемых учебников и учебных пособий при самостоятельной подготовке. Особое внимание следует обратить на выполнение практических работ, практических задач по СРО, тестовых вопросов.

При самостоятельной работе с учебниками и учебными пособиями полезно иметь под рукой справочную литературу (энциклопедии) или доступ к сети Интернет, так как могут встречаться новые термины, понятия, которые раньше обучающиеся не знали.

Цель практических занятий по дисциплине - закрепление знаний по определенной теме, приобретенных в результате прослушивания лекций, получения консультаций и самостоятельного изучения различных источников литературы. При выполнении данных работ обучающиеся должны будут глубоко изучить состав и принцип работы современных информационных систем. Получить практические навыки работы с современными ИС.

Перед практическим занятием обучающийся должен детально изучить теоретические материалы вопросов практики в учебниках, конспектах лекций, периодических журналах и прочее. Если при выполнении практического задания у обучающегося остаются неясности, то ему необходимо оперативно обратиться к преподавателю за уточнением.

После выполнения практического задания обучающиеся должны выполнить самостоятельную работу. Самостоятельная работа включает в себя индивидуальное задание по пройденной теме. Таким образом, каждый обучающийся выполняет только свой вариант задания.

При дистанционном выполнении практических работ обучающийся может самостоятельно приобрести операционные системы Windows XP, Windows 7, Windows 8, Windows 10 и пакет Microsoft Office или Open Office. Ответственность за установку и настройку программного обеспечения в данном случае ложится на обучающегося. Следует воспользоваться методическими указаниями по установке данных программных систем.

Результаты выполненных заданий оцениваются с учетом теоретических знаний по соответствующим разделам дисциплины, техники выполнения работы, объективности и обоснованности принимаемых решений в процессе работы с данными, качества оформления. Переход к выполнению следующего практического задания допускается только после отчета выполненной работы.

Учебно-методическое издание

Рабочая программа учебной дисциплины

**Разработка клиент-серверных приложений**

*(Наименование дисциплины в соответствии с учебным планом)*

**Сафонова Елена Владимировна**

*(Фамилия, Имя, Отчество составителя)*