

Федеральное государственное образовательное бюджетное учреждение
высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)**

**Кафедра информационных технологий
Факультет информационных технологий и анализа больших данных**

Документ подписан усиленной неквалифицированной электронной подписью
Организация: Финансовый университет при Правительстве РФ
Утверждено: Проректор по учебной и методической работе Е.А. Каменева
Сертификат: OgDO/neEd4N67KyOXjpL1inmFHZgdVG
Дата: 26.11.2025 г.

А.Ю. Шаталова

Веб-разработка

Рабочая программа дисциплины

для студентов, обучающихся по направлению подготовки:

09.03.03 - Прикладная информатика,

Образовательная программа «Прикладные информационные системы в экономике
и финансах»

Профиль:

«Инженерия данных»

Рекомендовано

*Факультет информационных технологий и анализа больших данных
(протокол № 03 от 16.12.2025 г.)*

Одобрено

*Кафедра информационных технологий
(протокол № 12 от 03.12.2025 г.)*

© Москва 2025

СОДЕРЖАНИЕ

1.	Наименование дисциплины	3
2.	Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине	3
3.	Место дисциплины в структуре образовательной программы	7
4.	Объем дисциплины (модуля) в зачетных единицах и в академических часах с выделением объема аудиторной (лекции, семинары) и самостоятельной работы обучающихся	7
5.	Содержание дисциплины, структурированное по темам (разделам) дисциплины с указанием их объемов (в академических часах) и видов учебных занятий	8
5.1.	Содержание дисциплины	8
5.2.	Учебно-тематический план	10
5.3.	Содержание семинаров	11
6.	Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине	18
6.1.	Перечень вопросов, отводимых на самостоятельное освоение дисциплины, формы внеаудиторной самостоятельной работы	18
6.2.	Перечень вопросов, заданий, тем для подготовки к текущему контролю	27
7.	Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине	31
8.	Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины	41
9.	Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины	41
10.	Методические указания для обучающихся по освоению дисциплины	43
11.	Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень необходимого программного обеспечения и информационных справочных систем	44
12.	Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине	44

1. Наименование дисциплины

«Веб-разработка».

2. Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине

Код компетенции	Наименование компетенции	Индикаторы достижения компетенции	Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции
ПКН-2	Способность разрабатывать алгоритмы и программы с использованием современных технологий программирования	Владеет объектно-ориентированным языком программирования на уровне знания синтаксиса и семантики, основ стандартной библиотеки	знать: принципы клиент-серверного взаимодействия (REST/GraphQL), методологии проектирования UI/UX, основы информационной безопасности веб-приложений, шаблоны проектирования (MVC, MVVM и др.) уметь: анализировать техническое задание, выбирать подходящий стек технологий, создавать макеты интерфейсов (wireframes), проектировать API эндпоинты и структуру базы данных
		Использует инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки)	знать: синтаксис и парадигмы выбранных языков программирования (JavaScript/Python/Java и др.), фреймворки (React/Vue/Angular; Express/Django/Spring), системы контроля версий (Git), основы работы с базами данных (SQL/NoSQL) уметь: писать чистый, модульный и документированный код; интегрировать frontend и backend части; работать с внешними API; выполнять операции CRUD с базой данных
		Организовывает кодовую базу, ориентируется в существующем коде, демонстрирует знание общепринятых соглашений и политик в области оформления кода	знать: виды тестирования (юнит-тесты, интеграционные, e2e), инструменты тестирования (Jest, Mocha, Cypress, Selenium), принципы TDD/BDD уметь: писать юнит-тесты для функций и компонентов,

			настраивать интеграционное тестирование API, использовать инструменты отладки
		Проектирует текстовый, программный или графический интерфейс программной системы исходя из ее назначения	знать: процессы сборки (build), контейнеризация (Docker), основы работы веб-серверов (Nginx), облачные платформы (VPS, Heroku, AWS Elastic Beanstalk/Yandex Cloud), CI/CD-пайплайны уметь: настраивать переменные окружения, создавать Docker-образ приложения, выполнять деплой на удаленный сервер, читать логи сервера
ПКП-2	Способность разрабатывать, согласовывать и управлять исполнением технического задания и технического проекта с использованием технологий больших данных	Работает со стандартами, в том числе адаптирует стандарты для специфических требований больших данных.	знать: структура и требования ГОСТ 34, IEEE 830 или корпоративных стандартов к технической документации; жизненный цикл данных (Data Lifecycle Management); типовые архитектурные паттерны для Big Data (Lambda, Kappa, Data Lake, Data Mesh); критерии выбора технологий больших данных (Hadoop, Spark, Kafka, NoSQL БД) в зависимости от типа данных (структурированные, полуструктурированные, неструктурированные) и задач (обработка в реальном времени, пакетная обработка); методы сбора и формализации требований заинтересованных сторон (стейкхолдеров) уметь: проводить интервью с бизнес-пользователями для выявления и анализа потребностей в работе с данными; декомпозировать бизнес-требования на технические спецификации; описывать функциональные и нефункциональные требования (производительность, масштабируемость, отказоустойчивость) к системам больших данных; формализовать критерии приемки (Acceptance Criteria) для этапов проекта; составлять глоссарий терминов в предметной области

		Разрабатывает технические задания и технические проекты для технологий больших данных.	знать: принципы работы и области применения компонентов экосистемы Big Data (HDFS, YARN, Spark Core/Streaming/SQL, Flink, Kafka, HBase, Cassandra, Elasticsearch и др.); модели согласованности данных (CAP-теорема) и их влияние на выбор СУБД; методы проектирования Data Pipeline (конвейеров данных); подходы к обеспечению качества данных (Data Quality) на этапе проектирования; принципы безопасности данных (Data Security) и нормативные требования (152-ФЗ, GDPR) уметь: выбирать и комбинировать технологии из стека Big Data для решения конкретной задачи; проектировать схемы данных, включая форматы хранения (Parquet, Avro, ORC) и стратегии партиционирования; разрабатывать архитектурные диаграммы (Data Flow, Component Diagram) с использованием нотаций (C4, UML); рассчитывать требования к инфраструктуре (кластерная мощность, объемы хранения, сетевые bandwidth); оценивать технологические риски и предлагать митигирующие действия
		Реализует управление рабочими проектами технологической инфраструктуры больших данных.	знать: методологии управления требованиями (Requirements Management); техники фасилитации и проведения согласовательных встреч (workshops); форматы и инструменты для совместной работы над документацией (Confluence, Wiki, Notion); процедуры управления изменениями (Change Management Process), включая оценку влияния изменений (Impact Analysis); основы коммуникации и аргументации в технической среде уметь: проводить презентации технических решений для технической и нетехнической аудитории; формулировать

			аргументированные ответы на замечания по проекту; вести протоколы совещаний и регистр замечаний; оценивать стоимость и сроки при внесении изменений в требования; использовать системы контроля версий для технической документации (Git для docs-as-code)
--	--	--	---

3. Место дисциплины в структуре образовательной программы

Дисциплина «Веб-разработка» относится к «Профилю "Инженерия данных"».

4. Объем дисциплины (модуля) в зачетных единицах и в академических часах с выделением объема аудиторной (лекции, семинары) и самостоятельной работы обучающихся

Очная форма обучения

Вид учебной работы по дисциплине	Всего (в з/е и часах)	Семестр 5 (в часах)	Семестр 6 (в часах)
Общая трудоёмкость дисциплины	6/216	108	108
Контактная работа- Аудиторные занятия	100	50	50
Лекции	32	16	16
Семинары, практические занятия	68	34	34
Самостоятельная работа	116	58	58
Вид текущего контроля	Контрольная работа, Контрольная работа	Контрольная работа	Контрольная работа
Вид промежуточной аттестации	Зачет, Экзамен	Зачет	Экзамен

5. Содержание дисциплины, структурированное по темам (разделам) дисциплины с указанием их объемов (в академических часах) и видов учебных занятий

5.1. Содержание дисциплины

Тема 1. Введение и основы веб-технологий

Введение в дисциплину. Современная веб-разработка как процесс создания ИС. Жизненный цикл информационной системы. Роли в веб-команде (Frontend, Backend, DevOps, QA). Обзор современных стеков технологий. Инструменты разработчика (браузер, редактор кода, терминал). Архитектура веб-приложений и протокол HTTP(S). Клиент-серверная модель. Структура HTTP-запроса и ответа (методы, статусы, заголовки). Безопасность (HTTPS, CORS). Основы DNS и доменных имен. Основы клиентской триады: HTML5 и CSS3. Семантическая HTML-разметка. Основы каскадных таблиц стилей (CSS): селекторы, блочная модель, позиционирование. Валидация и доступность.

Тема 2. Разработка клиентской части (Frontend)

Программирование на JavaScript (ES6+). Синтаксис, типы данных, функции, управляющие конструкции. Работа с DOM и событиями. Асинхронность: Callbacks, Promises, async/await. Fetch API для взаимодействия с сервером. Современный CSS и адаптивная верстка. Flexbox и CSS Grid Layout. Медиазапросы. Методологии (БЭМ). Препроцессоры (Sass/SCSS). Фреймворки (Bootstrap/Tailwind CSS - по выбору). Компонентный подход и фреймворки (на примере React). Концепция Virtual DOM. Создание компонентов (функциональные и классовые). State & Props. Жизненный цикл компонента. Хуки (useState, useEffect). Управление состоянием и маршрутизация в SPA. Паттерн Flux и его реализации (Redux Toolkit, MobX). Клиентская маршрутизация (React Router). Сборка проекта (Webpack/Vite).

Тема 3. Разработка серверной части (Backend) и работа с данными

Введение в серверную разработку на Node.js (или Python/PHP). Среды исполнения. Встроенные модули. Основы фреймворка Express.js (маршрутизация, middleware). Проектирование и разработка RESTful API. Принципы REST. Структура endpoint'ов. Работа с параметрами запроса и телом (JSON). Валидация входных данных. Документирование API (OpenAPI/Swagger). Основы баз данных. Реляционные (SQL) vs. нереляционные (NoSQL) СУБД. Язык SQL: базовые операции (CRUD), связи между таблицами (JOIN). Нормализация данных. Введение в MongoDB. Интеграция Backend с

БД. Подключение к СУБД (драйверы). Использование ORM (например, Sequelize для Node.js или Prisma) / ODM (например, Mongoose). Миграции базы данных.

Тема 4. Интеграция, безопасность и инструменты

Аутентификация и авторизация в веб-приложениях. Сессии vs. Токены. Реализация JWT (JSON Web Tokens). OAuth 2.0 (социальные входы). Хеширование паролей (bcrypt). Безопасность веб-приложений. Обзор OWASP Top 10. Защита от атак: SQL-инъекции, XSS, CSRF. Безопасные заголовки HTTP (CSP, HSTS). Работа с чувствительными данными. Инструменты контроля версий и совместной разработки. Система Git: основные команды, ветвление (Git Flow/GitHub Flow), разрешение конфликтов. Работа с удаленными репозиториями (GitHub/GitLab). Code Review. Тестирование и отладка. Виды тестов: unit, integration, end-to-end (e2e). Написание модульных тестов с использованием Jest (для JS). Инструменты e2e-тестирования (Cypress, Puppeteer). Отладка на клиенте и сервере.

Тема 5. Деплой и мониторинг

Контейнеризация и облачный деплой. Введение в Docker (образы, контейнеры, Dockerfile, docker-compose). Развертывание приложения на облачных платформах (Heroku, Vercel, AWS EC2 / Google Cloud Run). Основы непрерывной интеграции и поставки (CI/CD). Принципы CI/CD. Настройка простого пайплайна (на примере GitHub Actions или GitLab CI) для запуска тестов и деплоя. Оптимизация и мониторинг производительности. Метрики производительности (Core Web Vitals). Аудит с помощью Lighthouse. Кэширование (браузерное, серверное). Базовый мониторинг работы приложения.

5.2. Учебно-тематический план

№ п/п	Наименование тем(разделов) дисциплины	Трудоемкость в часах				
		Всего	Контактная работа - Аудиторная работа			Самостоя тельная работа
			Общая, в т.ч.:	Лекции	Семинары, практическ ие занятия	
1	Введение и основы веб-технологий	35	15	5	10	20
2	Разработка клиентской части (Frontend)	70	30	10	20	40
3	Разработка серверной части (Backend) и работа с данными	53	23	8	15	30
4	Интеграция, безопасность и инструменты	36	20	5	15	16
5	Деплой и мониторинг	22	12	4	8	10
	Итого	216	100	32	68	116

5.3. Содержание семинаров

Наименование тем (разделов) дисциплины	Перечень вопросов для обсуждения на семинарах, практических занятиях	Формы проведения занятий
Введение и основы веб-технологий	Чем процесс разработки современного веб-приложения (SPA) принципиально отличается от создания классического многостраничного сайта (MPA) начала 2000-х годов? Какие ключевые роли (specializations) существуют в команде веб-разработки и как распределяется ответственность между фронтенд, бэкенд и DevOps-инженером при создании новой функции? Какую эволюцию прошли инструменты веб-разработчика (от простого текстового редактора до современных IDE и DevTools) и как они повлияли на производительность и качество кода? Почему сегодня нельзя рассматривать веб-разработку только как написание кода? Какие смежные области (UX, безопасность, performance, SEO) становятся ее неотъемлемой частью? Что происходит в браузере, когда пользователь вводит URL в адресную строку и нажимает Enter? Опишите последовательность шагов вплоть до отображения страницы. В чем ключевая разница между методами HTTP GET и POST с точки зрения идиоматичности, безопасности и типового использования? Почему для операции оплаты нельзя использовать GET? Как протокол HTTPS обеспечивает конфиденциальность и целостность данных между браузером и сервером? Почему он критически важен для любого сайта с формами ввода? Что такое CORS (Cross-Origin Resource Sharing) и почему этот механизм безопасности был введен в браузеры? Приведите пример сценария, когда он блокирует запрос. Каковы основные различия между статус-кодами HTTP из групп 4xx и 5xx? Кто обычно отвечает за устранение ошибок 404 и 500 — разработчик фронтенда или бэкенда? Почему семантическая HTML-разметка (<article>, <nav>, <header>) важнее, чем верстка сплошными <div>? Как она влияет на доступность, SEO и читаемость кода? Как принцип каскадности (Cascading) в CSS влияет на конечный стиль элемента при наличии конфликтующих правил? Что имеет больший вес: селектор по ID или селектор по классу? В чем	Практические занятия/Семинары; Лабораторные работы;

	закljučается фундаментальная проблема классической блочной модели CSS (content-box) и как свойство box-sizing: border-box решает ее на практике? Каковы основные принципы адаптивного веб-дизайна (RWD)? Почему медиазапросы (media queries) — это лишь часть решения, а не его основа? Почему обеспечение веб-доступности (a11y) — это обязанность разработчика, а не опция? Какие минимальные практики (например, alt для изображений, ARIA-атрибуты) должен знать каждый?	
Разработка клиентской части (Frontend)	Чем let и const принципиально отличаются от var с точки зрения области видимости (scope) и поднятия (hoisting)? В каких сценариях использование var может привести к скрытым ошибкам? В чем разница между стрелочной функцией (=>) и обычной функциональной декларацией (function) по следующим параметрам: контекст выполнения (this), наличие arguments, возможность использовать в качестве конструктора? Каков жизненный цикл Promise? Чем цепочка . then(). catch() отличается от конструкции try/catch при работе с асинхронным кодом? Когда уместно использовать async/await, а когда — цепочки Promise? Как работает механизм обработки событий (Event Handling) в браузере? Что такое фазы всплытия (bubbling) и погружения (capturing)? Как и зачем использовать методы event. stopPropagation() и event. preventDefault()? Когда следует использовать Flexbox, а когда — CSS Grid для построения макета? Можно ли сказать, что Grid «вытесняет» Flexbox, или у них разные сферы применения? В чем преимущество использования CSS-препроцессора (например, Sass) перед чистым CSS? Какие конкретные возможности (переменные, миксины, вложенность) решают какие практические проблемы поддержки кода? Что такое «Mobile First» как подход к адаптивному дизайну? Почему технически и концептуально он предпочтительнее стратегии «Desktop First»? Как этот подход отражается в структуре медиазапросов? Зачем нужны CSS-методологии вроде БЭМ? Как решает проблему конфликта стилей простое соглашение об именовании классов по принципу block__element_modifier? В чем фундаментальное отличие декларативного подхода (React) от императивного (прямые манипуляции с DOM через jQuery)? Как декларативность упрощает	Практические занятия/Семинары; Лабораторные работы;

	рассуждения о состоянии интерфейса? Что такое «состояние» (state) и «свойства» (props) компонента? Почему state считается «приватным», а props — «публичным» API компонента? Как изменение каждого из них влияет на ререндер? Зачем в функциональных компонентах React нужны хуки (Hooks)? Как useEffect заменяет методы жизненного цикла (componentDidMount, componentDidUpdate) и какие новые возможности дает? В чем заключаются ключевые проблемы «прокидывания пропсов» (props drilling) и как их решают с помощью Context API или сторонних решений для управления состоянием? Когда сложности проекта перерастают возможности useState и useContext, и пора подключать специализированную библиотеку (Redux, MobX)? Какие признаки этого? Каковы три основных принципа Redux (единый источник истины, состояние только для чтения, изменения через чистые функции-редьюсеры)? Как Redux Toolkit упрощает классическую настройку Redux? Как клиентская маршрутизация (например, в React Router) меняет URL в браузере без полной перезагрузки страницы? В чем отличие HashRouter от BrowserRouter и какие ограничения у последнего? Какие стратегии кэширования и ленивой загрузки (lazy loading) компонентов/маршрутов можно применить в SPA для повышения производительности при первом посещении и последующих переходах?	
Разработка серверной части (Backend) и работа с данными	В чем ключевые архитектурные различия между монолитным приложением (где весь backend — единая кодовая база) и микросервисным подходом? Какие преимущества и операционные сложности появляются при переходе к микросервисам? Что такое middleware («промежуточное ПО») в контексте фреймворков (Express.js, Django)? Какие типовые задачи решаются с его помощью (аутентификация, логирование, CORS, валидация)? Приведите пример последовательности выполнения middleware для одного HTTP-запроса. Как правильно организовать структуру файлов и папок (project layout) для масштабируемого backend-приложения? Почему важно отделять маршруты (routes), контроллеры (controllers), бизнес-логику (services) и модели данных (models)? Каковы ключевые принципы архитектурного стиля REST (Client-Server, Stateless, Uniform Interface и др.)?	Практические занятия/Семинары; Лабораторные работы;

	Почему следование этим принципам повышает предсказуемость, масштабируемость и независимость компонентов системы? Как правильно спроектировать ресурс-ориентированные URL (endpoints) для сущности «Статья блога», которая имеет комментарии? Приведите примеры GET, POST, PUT, DELETE запросов для статей и для комментариев к конкретной статье. Какие основные подходы к версионированию API существуют (версия в URL, в заголовке Ассерта, в домене)? Каковы их плюсы, минусы и рекомендации по применению? Почему валидация входных данных — это ответственность backend, даже если она уже есть на frontend? Какие библиотеки или встроенные механизмы для валидации вы знаете? В каких сценариях следует выбрать реляционную (SQL), а в каких — нереляционную (NoSQL) СУБД? Сравните их по критериям: схема данных, масштабируемость, поддержка транзакций (ACID), типовые use cases. Что такое SQL-инъекция? Продемонстрируйте, как злонамеренный пользовательский ввод может изменить или удалить данные, если запрос формируется простой конкатенацией строк. Какие два основных метода защиты от этой уязвимости? Объясните на примере сущностей «Пользователь» и «Заказ», что такое связи «один-ко-многим» (One-to-Many) и «многие-ко-многим» (Many-to-Many). Как они реализуются на уровне схемы SQL и в SQL-запросе с использованием JOIN? В чем заключается «проблема объектно-реляционного отображения» (Object-Relational Impedance Mismatch), которую призваны решить ORM? Какие задачи ORM (например, Sequelize, Prisma) берут на себя, упрощая жизнь разработчику? Каковы основные преимущества и риски использования ORM по сравнению с написанием «чистого» SQL? Когда использование ORM может стать «антипаттерном» и создать проблемы с производительностью? Что такое миграции базы данных (Database Migrations) и зачем они нужны? Почему прямое изменение структуры БД на продакшн-сервере через GUI-клиент считается опасной практикой? Как организовать доступ к БД из backend-приложения корректно с точки зрения управления соединениями (connection pool)? Что произойдет, если не закрывать соединения после выполнения запросов?	
--	--	--

Интеграция, безопасность и инструменты	В чем фундаментальное отличие аутентификации (authentication) от авторизации (authorization)? Какие механизмы (сессии, JWT, OAuth) решают какую задачу? Приведите пример, где пользователь аутентифицирован, но не авторизован для выполнения действия. Каков жизненный цикл JWT-токена? Почему в классической реализации JWT нельзя «отозвать» доступ до истечения срока его действия (exp), и какие стратегии (короткоживущие access-токены + refresh-токены, blacklist) решают эту проблему? Когда следует использовать сквозную аутентификацию (сессии, хранимые в БД или Redis), а когда — безсостоятельную (stateless JWT)? Сравните их с точки зрения масштабируемости серверного кластера, сложности реализации выхода (logout) и контроля доступа. Объясните механизм атаки «Межсайтовая подделка запроса» (CSRF). Как злоумышленник может заставить браузер авторизованного пользователя выполнить нежелательное действие (например, перевод денег) на другом сайте? Какие два основных метода защиты на уровне backend (CSRF-токены, SameSite cookies) и как они работают? Что такое «Межсайтовые сценарии» (XSS) и какие три основных типа (Reflected, Stored, DOM-based) существуют? Как санитизация (очистка) пользовательского ввода и контентная политика безопасности (CSP) предотвращают эту атаку? Почему хранение секретных ключей, паролей и конфиденциальных данных в коде или публичных репозиториях Git — критическая ошибка? Каковы современные практики управления секретами (environment variables, .env-файлы, облачные хранилища секретов)? В чем ключевая разница между централизованной (SVN) и распределенной (Git) системами контроля версий? Как распределенная модель Git повышает надежность и упрощает параллельную работу? Опишите рабочий процесс (workflow) на основе веток (branching model). В чем разница между популярными моделями Git Flow (с develop, release-ветками) и GitHub Flow (одна main, feature-ветки)? Какой подход более уместен для команды, практикующей CI/CD? Что такое merge conflict и в какой момент разработки он возникает? Каков алгоритм действий по его разрешению с помощью инструментов Git и IDE? Сформулируйте	Практические занятия/Семинары; Лабораторные работы;
---	---	--

	«пирамиду тестирования» (Test Pyramid). Почему рекомендуется иметь много модульных (unit) тестов, меньше интеграционных (integration) и еще меньше end-to-end (e2e) тестов? Каковы затраты на написание и поддержку каждого типа? Что такое моки (mocks), стабы (stubs) и шпионы (spies) в контексте модульного тестирования? Зачем они нужны, если вы тестируете функцию, которая зависит от сложной внешней системы (БД, API)? Как организовать эффективный процесс отладки (debugging) полноценного веб-приложения? Какие инструменты вы будете использовать для поиска ошибки на стороне клиента (DevTools, логирование), на стороне сервера (дебаггер в IDE, логи) и на уровне сетевого взаимодействия (вкладка Network, Postman)?	
Деплой и мониторинг	В чем фундаментальное отличие виртуальной машины (VM) от контейнера (например, Docker)? Как архитектура на основе контейнеров решает проблемы переносимости приложений и эффективного использования ресурсов («works on my machine»)? Что описывают ключевые инструкции в Dockerfile (FROM, COPY, RUN, CMD, EXPOSE)? Как правильно их структурировать, чтобы минимизировать размер итогового образа и ускорить процесс сборки? Зачем для многокомпонентного приложения (frontend, backend, БД) нужен docker-compose.yml? Как он упрощает локальную разработку и процесс развертывания по сравнению с ручным запуском отдельных контейнеров? Дайте определение CI (Continuous Integration) и CD (Continuous Delivery/Deployment). В чем разница между Continuous Delivery (поставка вручную) и Continuous Deployment (развертывание автоматически)? Какой подход менее рискованный для начинающей команды? Каков типовой пайплайн (pipeline) CI/CD для веб-приложения? Назовите ключевые этапы (например, сборка, тестирование, сборка Docker-образа, деплой на staging/prod) и объясните цель каждого. Какие метрики и критерии «качества сборки» (например, успешное прохождение всех тестов, отсутствие уязвимостей в зависимостях, размер бандла) должны блокировать автоматический деплой в production? Что такое Core Web Vitals и почему они стали ключевыми метриками пользовательского опыта? Как конкретные технические действия (оптимизация	Практические занятия/Семинары; Лабораторные работы;

	изображений, lazy loading, кэширование) влияют на показатели LCP, FID, CLS?Какая разница между мониторингом (monitoring) и оповещением (alerting)? Какие метрики (логи ошибок, время ответа сервера, потребление памяти, бизнес-показатели) нужно мониторить в первую очередь для небольшого веб-приложения?Каковы основные стратегии кэширования на разных уровнях: на стороне клиента (браузерные заголовки Cache-Control), на стороне сервера (in-memory кэш вроде Redis), на уровне CDN? В каких сценариях каждая из них наиболее эффективна?Как правильно декомпозировать крупную цель («сделать интернет-магазин») на управляемые задачи и подзадачи для постановки в бэклог проекта? Какие критерии у «хорошей» задачи в трекере (например, в Jira или GitHub Issues)?Почему документация — это неотъемлемая часть продукта? Какой минимальный набор документов (README, архитектурное описание, инструкция по запуску, описание API) должен сопровождать проект и для кого (разработчик, новый член команды, заказчик) предназначен каждый?	
--	--	--

6. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине

6.1. Перечень вопросов, отводимых на самостоятельное освоение дисциплины, формы внеаудиторной самостоятельной работы

Наименование тем (разделов) дисциплины	Перечень вопросов, отводимых на самостоятельное освоение	Формы внеаудиторной самостоятельной работы
Введение и основы веб-технологий	1. Что такое Интернет и Всемирная паутина (WWW)? Различие между Интернетом (сетью сетей) и Вебом (одной из служб, работающих в Интернете). История возникновения и ключевые этапы развития (от ARPANET до Web 3. 0). Базовые принципы работы: концепция клиент-серверного взаимодействия. 2. Как работает веб? Базовый цикл запрос-ответ. Роль веб-браузера (клиента) и веб-сервера. Что происходит, когда пользователь вводит URL в адресную строку и нажимает Enter (краткий обзор). Понятие DNS (системы доменных имен) и ее роль в преобразовании example. com в IP-адрес. 3. Триада фронтенда: HTML, CSS, JavaScript. HTML (HyperText Markup Language): Назначение: структура и семантика веб-страницы; Основные понятия: теги, атрибуты, элементы; Базовая структура документа (<html>, <head>, <body>). CSS (Cascading Style Sheets): Назначение: внешний вид и оформление (цвета, шрифты, расположение элементов); Основные понятия: селекторы, свойства, значения; Способы подключения CSS к HTML. JavaScript: Назначение: интерактивность и динамическое поведение страницы; Что может делать JS: реагировать на действия пользователя, изменять контент, общаться с сервером; Роль JS в современном вебе. 4. Протокол HTTP/HTTPS. Что такое протокол? Модель «запрос-ответ». Основные методы HTTP-запросов: GET, POST, PUT, DELETE. Коды состояния HTTP (Status Codes): 200 (OK), 404 (Not Found), 500 (Server Error) и др. Что такое HTTPS и почему он важен (шифрование, SSL/TLS-сертификаты). 5. URL / URI. Структура веб-адреса. Разбор компонентов URL: протокол, домен, порт, путь, параметры запроса (?ключ=значение), якорь (#). 6. Что такое	Работа с учебной литературой и ресурсами.

	веб-сервер (программное обеспечение: Nginx, Apache). Его основная функция: обрабатывать HTTP-запросы и возвращать ресурсы (HTML-страницы, изображения, CSS/JS файлы). 7. Хостинг и доменные имена. Как разместить свой сайт в интернете? Виды хостинга (общий, VPS, облачный). Регистрация и принцип работы доменных имен. 8. Введение в серверную часть (Backend). Зачем нужен бэкенд? (Хранение данных, аутентификация, сложная логика). Краткий обзор серверных языков: Python (Django/Flask), JavaScript (Node.js), PHP, Java, C#. Понятие о веб-фреймворках. 9. Базы данных (БД). Зачем нужны базы данных для веб-сайтов? Типы баз данных: реляционные (SQL, например, MySQL, PostgreSQL) и нереляционные (NoSQL, например, MongoDB). 10. Веб-браузер как среда выполнения. Что такое движок браузера (Blink, WebKit, Gecko)? Инструменты разработчика (DevTools) в браузере: для чего нужны и как ими пользоваться на базовом уровне (просмотр HTML/CSS, консоль, сетевая активность). 11. Версионирование кода (Git). Зачем нужны системы контроля версий (VCS)? Базовые понятия Git: репозиторий, коммит, ветка. Знакомство с GitHub / GitLab как платформами для хранения кода и совместной работы. 12. Введение в веб-безопасность (базовые понятия). Почему безопасность важна? Основные угрозы: XSS, SQL-инъекции. Важность валидации данных на стороне сервера. Роль HTTPS.	
Разработка клиентской части (Frontend)	1. HTML5: Современная семантика и API. Семантические теги нового поколения: <article>, <section>, <aside>, <nav>, <header>, <footer>, <main>, <figure>, <time> и их правильное использование для доступности и SEO. Мультимедийные элементы: <audio>, <video> (атрибуты, управление). Графика: <canvas> vs <svg> (базовые отличия и назначение). Формы HTML5: новые типы полей (email, tel, date, range, color), атрибуты валидации (required, pattern, placeholder). 2. CSS3: Продвинутое техники верстки. Flexbox: Основная концепция: контейнеры, оси, направления; Ключевые свойства для контейнера (display: flex, flex-direction, justify-content, align-items, flex-wrap); Свойства для элементов (flex-grow, flex-shrink, flex-basis, order). CSS Grid Layout: Двумерная система (строки и столбцы); Определение сетки:	Работа с учебной литературой и ресурсами.

	grid-template-columns/rows, grid-gap; Размещение элементов: grid-column, grid-row, grid-area; Сравнение и выбор между Flexbox и Grid. Адаптивный и отзывчивый дизайн (Responsive Web Design - RWD): Медиа-запросы (@media): синтаксис, логические операторы; Mobile-first vs Desktop-first подходы; Относительные единицы измерения: vw, vh, %, rem, em; Техники: резиновые изображения (max-width: 100%), свойства object-fit, clamp(). 3. Современный синтаксис ES6+. Ключевые нововведения: let и const (область видимости блока, отличия от var); Стрелочные функции (=>) и их особенности с this; Шаблонные строки (Template Literals) и интерполяция; Деструктуризация массивов и объектов; Операторы spread (. . .) и rest; Параметры по умолчанию. 4. Работа с DOM (Document Object Model) на практике. Методы поиска элементов: querySelector, querySelectorAll. Создание, изменение, удаление узлов (createElement, appendChild, remove). Работа с классами, атрибутами, стилями элементов. События (Events): Основная модель: addEventListener, removeEventListener; Объект события (event), его свойства и методы (preventDefault(), stopPropagation()); Делегирование событий (Event Delegation) — принцип и преимущества. 5. Асинхронность в JavaScript. AJAX и fetch API: Что такое AJAX и зачем он нужен; Работа с fetch(): отправка GET и POST запросов; Обработка ответов (. json(), . text()), обработка ошибок. Промисы (Promises): Проблема "ада обратных вызовов" (Callback Hell); Создание и потребление промисов (. then(), . catch(), . finally()). Async/Await: Синтаксический сахар над промисами, работа с асинхронным кодом как с синхронным; Обработка ошибок с try. . . catch. 6. Системы контроля версий для фронтенд-разработчика. Углубленная работа с Git: ветвление (branch), слияние (merge), разрешение конфликтов. Рабочие процессы (Workflow), например, GitHub Flow или Git Flow. Понятие о git rebase. 7. Сборка проекта: менеджеры пакетов и модули. Менеджеры пакетов: npm или yarn. Что такое package. json и package-lock. json/yarn. lock? Модули в JavaScript (ES Modules): import/export. Сборщики модулей (Module Bundlers): Базовое понимание, зачем нужен Webpack или Vite (объединение файлов, транспиляция,	
--	---	--

	минификация). 8. Фреймворки и библиотеки (введение). Зачем нужны фреймворки? Проблемы работы с "голым" JavaScript на больших проектах. Обзор основных игроков: React, Vue. js, Angular. Их философия и ключевые отличия (на концептуальном уровне). Что такое SPA (Single Page Application)? Основная идея и преимущества/недостатки перед МРА. 9. Основы доступности (Web Accessibility, a11y). Почему доступность важна? (Юридические, этические, бизнес-аспекты). Базовые принципы: семантическая HTML-разметка, ARIA-атрибуты (aria-label, aria-hidden), управление фокусом. Инструменты для проверки (Lighthouse в DevTools). 10. Оптимизация фронтенда. Оптимизация загрузки: минификация и сжатие CSS/JS, ленивая загрузка изображений (loading="lazy"). Производительность рендеринга: CSS-свойства, влияющие на перерисовку и перекомпоновку (reflow/repaint). Использование DevTools для аудита производительности (Performance, Lighthouse).	
Разработка серверной части (Backend) и работа с данными	1. Архитектура клиент-серверного приложения. Детальное понимание цикла "запрос-ответ" (HTTP Request/Response Cycle). Роль веб-сервера (Nginx, Apache) и серверного приложения (ваш код на Node. js/Python и т. д.). Модели взаимодействия: SSR (Server-Side Rendering) vs API для SPA. 2. Выбор стека технологий. Обзор популярных языков и их фреймворков: JavaScript/Node. js: Express. js, NestJS, Fastify; Python: Django (полноценный фреймворк), Flask (микрофреймворк), FastAPI; PHP: Laravel, Symfony; Java: Spring Boot; Go: Gin, Echo. Критерии выбора: производительность, скорость разработки, экосистема, личные предпочтения. 3. Создание первого серверного приложения (Hello World API). Настройка среды разработки (установка Node. js/Python интерпретатора и т. д.). Инициализация проекта, управление зависимостями (npm init, pip, composer). Создание простейшего HTTP-сервера, который обрабатывает запросы и возвращает JSON. 4. Маршрутизация (Routing). Определение эндпоинтов (endpoints) API. Работа с HTTP-методами: GET (получение), POST (создание), PUT/PATCH (обновление), DELETE (удаление). Извлечение данных из URL: параметры строки запроса (?id=1), параметры пути (/users/:id). 5.	Работа с учебной литературой и ресурсами.

	Обработка данных запроса и ответа. Request: Работа с телами запросов (body parsing) в форматах JSON, FormData. Доступ к заголовкам (headers). Response: Формирование ответов с правильными статус-кодами (200, 201, 400, 404, 500). Отправка данных в JSON, HTML, файлов. 6. Промежуточное ПО (Middleware). Концепция middleware (цепочка обработки запроса). Написание собственных middleware для логирования, аутентификации, валидации. Использование встроенных и сторонних middleware (для работы с CORS, статическими файлами и т. д.). 7. Введение в базы данных. Реляционные (SQL) vs Нереляционные (NoSQL) БД. Плюсы, минусы, сценарии использования. Основы SQL: ключевые команды (SELECT, INSERT, UPDATE, DELETE, JOIN). Нормализация данных (базовое понимание). Обзор систем: PostgreSQL (продвинутый SQL), MySQL/MariaDB, SQLite (для разработки). 8. Подключение БД к серверному приложению. Драйверы баз данных. Концепция ORM (Object-Relational Mapping) для SQL и ODM (Object-Document Mapper) для NoSQL. Примеры: Sequelize (Node.js), Prisma (Node.js), SQLAlchemy (Python), Eloquent (PHP/Laravel). Определение моделей (сущностей) и их связей (один-ко-многим, многие-ко-многим). 9. Выполнение операций CRUD через ORM/ODM. Создание, чтение, обновление и удаление записей. Использование миграций (Migrations) для безопасного изменения структуры БД. 10. Аутентификация и авторизация. Различие понятий: Authentication (кто ты?) vs Authorization (что тебе можно?). Реализация регистрации и входа (логин/пароль). Токены: JWT (JSON Web Tokens) - структура, принцип работы, хранение на клиенте (в куках или localStorage). Сессии: Работа с сессиями на сервере (session store в Redis или БД). 11. Основы безопасности веб-приложений (OWASP Top 10). SQL-инъекции (SQLi): Почему ORM и параметризованные запросы защищают от них. XSS (Межсайтовый скриптинг): Защита на уровне backend (экранирование данных). CSRF (Межсайтовая подделка запроса): Защита с помощью токенов. Хеширование паролей: Алгоритмы (bcrypt, scrypt), использование "соли" (salt). 12. RESTful API Design (Основы). Принципы REST: ресурсы, HTTP-методы, статус-коды, версионирование API. Форматы ответа и ошибок.	
--	---	--

	Документирование API (Swagger/OpenAPI). 13. Взаимодействие с внешними сервисами (API сторонних провайдеров). Отправка HTTP-запросов из backend-кода (используя axios, requests и т. д.). Работа с API популярных сервисов (например, Telegram, Google Maps, Stripe/Pay). 14. Основы контейнеризации и развёртывания. Docker: Зачем нужен? Создание простого Dockerfile для backend-приложения. Переменные окружения (Environment Variables): Хранение чувствительных данных (ключи, пароли БД). Основы облачного развёртывания: Выбор платформы (Render, Railway, Heroku, VPS). Процесс деплоя.	
Интеграция, безопасность и инструменты	1. Архитектурные паттерны интеграции. SPA + Backend API: Классическая современная схема. Роль бэкенда как чистого поставщика JSON API. Server-Side Rendering (SSR): Когда и зачем нужен? Реализация с помощью Next.js (для React), Nuxt.js (для Vue) или шаблонизаторов на бэкенде (EJS, Pug, Jinja2). Статическая генерация (SSG) и гибридные подходы (Next.js). Микросервисы vs Монолит: Базовое понимание, плюсы и минусы, проблемы интеграции (общение между сервисами). 2. Практическая настройка взаимодействия. Настройка CORS (Cross-Origin Resource Sharing) на бэкенде: зачем, как и какие политики безопасны. Правильная организация HTTP-запросов с фронтенда: использование экземпляров axios, интерсепторы для автоматической подстановки токенов, обработка ошибок глобально. Управление состоянием аутентификации на клиенте (сохранение токена, проверка срока, обновление refresh-токена). 3. Защита от OWASP Top-10 угроз (практическое применение). Инъекции (A01): Параметризованные запросы в ORM, валидация и санация всех пользовательских данных. Недостатки аутентификации (A02): Реализация сложных паролей, защита от брут-форса (rate limiting), безопасное хранение сессий, использование проверенных библиотек для JWT. Раскрытие конфиденциальных данных (A03): Запрет на логирование паролей и токенов, использование HTTPS везде, шифрование чувствительных полей в БД. Межсайтовый скриптинг (XSS, A03): Экранирование (escaping) данных на выходе, политика Content Security Policy (CSP) — что это и как настроить заголовок. Подделка межсайтовых запросов (CSRF, A07):	Работа с учебной литературой и ресурсами.

	Защита для stateful-приложений (анти-CSRF токены) и stateless-API (правильная настройка CORS, проверка заголовков). Небезопасные десериализации (A08): Понимание рисков при работе с eval() или недоверенными данными сессий. 4. Безопасность API. Авторизация на уровне объектов (ABAC/RBAC): Реализация проверок, что пользователь имеет доступ именно к этому ресурсу (например, if (post. userId !== req. user. id) return 403). Rate Limiting (ограничение частоты запросов): Защита API от DDoS и брут-форса. Использование готовых middleware. Валидация и санация входных данных: Глубинная проверка всех параметров запроса (query, body, params). Библиотеки: Joi, Zod, class-validator. 5. Работа с конфиденциальными данными и ключами. Принцип наименьших привилегий для доступа к БД и сервисам. Управление секретами: Никаких паролей в коде! Использование переменных окружения, .env файлов (с .gitignore), Secrets Manager в облаках (AWS Secrets Manager, HashiCorp Vault базовые понятия). Безопасная работа с файловыми загрузками: проверка MIME-типа, сканирование на вирусы, хранение вне корневой директории веб-сервера. 6. Контейнеризация и оркестрация (базовый уровень). Docker: Углубленное понимание. Создание эффективных многоступенчатых (multi-stage) Dockerfile. Оптимизация размера образов. Docker Compose: Локальный запуск всего стека приложения (frontend, backend, БД, Redis, Nginx) одной командой. Основы оркестрации: Зачем нужен Kubernetes (K8s)? Базовые понятия: pod, deployment, service. Умение развернуть простое приложение в Minikube или managed K8s (опционально). 7. CI/CD (Непрерывная интеграция и доставка). Полный цикл: код -> репозиторий -> автоматические тесты -> сборка образа -> развертывание. Настройка пайплайна в GitHub Actions / GitLab CI: запуск линтеров и тестов, сборка Docker-образа, деплой на staging/production. Стратегии деплоя: синий-зеленый (blue-green), канареечный (canary) — базовое понимание. 8. Мониторинг, логирование и отладка в production. Структурированные логи (Structured Logging): Использование JSON-формата для логов для их последующего анализа. Централизованный сбор логов: Знакомство с ELK-стеком (Elasticsearch, Logstash, Kibana) или	
--	---	--

	Grafana Loki. Метрики и APM (Application Performance Monitoring): Интеграция с Prometheus, отправка метрик (запросов в секунду, время ответа, ошибки). Использование инструментов типа Sentry для отслеживания ошибок на клиенте и сервере. 9. Оптимизация работы с базой данных. Индексы: Зачем, когда и как создавать. Анализ медленных запросов (EXPLAIN в SQL). Кэширование: Стратегии кэширования. Использование Redis или Memcached для кэширования результатов запросов, сессий, HTML-фрагментов. Пагинация на уровне БД, а не в приложении. 10. Оптимизация API и фронтенда для production. CDN (Content Delivery Network): Размещение статики (CSS, JS, картинки, шрифты). Сжатие (gzip, Brotli) на уровне веб-сервера (Nginx). Оптимизация загрузки фронтенда: code splitting, lazy loading, tree shaking. HTTP/2 и HTTP/3: Преимущества.	
Деплой и мониторинг	1. Жизненный цикл приложения: от разработки до продакшена. Понимание различных окружений: local -> development -> testing/staging -> production. Концепция Immutable Infrastructure (неизменяемая инфраструктура). Почему это важно и как достигается (через Docker-образы). 2. Платформы для деплоя и их выбор. PaaS (Platform as a Service): Heroku, Render, Railway. Простота, ограничения. Автоматический деплой из Git. VPS (Virtual Private Server): DigitalOcean, Linode, AWS EC2. Полный контроль, больше ответственности. Ручная или скриптовая настройка. Контейнерные платформы: Managed Kubernetes: Google GKE, Amazon EKS, Azure AKS. Мощно, сложно, дорого; Упрощённые контейнерные сервисы: Fly.io, DigitalOcean App Platform. Serverless/Edge: Vercel, Netlify (для фронтенда и serverless функций), Cloudflare Workers. Критерии выбора: стоимость, сложность, масштабируемость, требования проекта. 3. Стратегии деплоя (Deployment Strategies). Big Bang / Recreate: Остановка старой версии, развёртывание новой. Просто, но есть downtime. Rolling Update: Постепенная замена подов/инстансов. Без простоев, но две версии сосуществуют. Blue-Green: Два идентичных окружения (синее – старое, зелёное – новое). Переключение трафика мгновенно. Требуется ресурсов. Canary: Новую версию получает небольшая часть пользователей (например, 5%). Снижение рисков. 4. Инфраструктура как код	Работа с учебной литературой и ресурсами.

	(IaC) – основы. Зачем нужно описывать инфраструктуру в коде? Воспроизводимость, контроль версий. Terraform vs Pulumi vs CloudFormation: Базовое сравнение. Написание простого конфига Terraform для развёртывания VPS или managed БД. 5. Три столпа Observability: Логи, Метрики, Трейсы. Логи (Logs): Что, где и когда произошло. Централизованный сбор (Loki, ELK). Уровни логирования (debug, info, error). Метрики (Metrics): Числовые данные о работе системы за время (RPS, latency, error rate, потребление CPU/RAM). Prometheus + Grafana. Трейсы (Traces): Как запрос проходит через все микросервисы/компоненты. OpenTelemetry, Jaeger. Для распределённых систем. 6. Построение эффективных дашбордов (Grafana). Принципы хорошего дашборда: для кого он (DEV, OPS, бизнес)? Какие метрики критичны? (SLO: Service Level Objectives). Полезные метрики для мониторинга: Rate, Errors, Duration (RED метод) и Utilization, Saturation, Errors (USE метод). 7. Системы алертинга (Alerting). Когда и на что нужно оповещать? Принцип "алерт должен требовать действия". Настройка алертов в Prometheus Alertmanager или Grafana. Каналы доставки: Email, Slack, Telegram, PagerDuty. Избегание "алертной усталости": группировка, подавление шума. 8. Бэкапы и аварийное восстановление (Disaster Recovery, DR). RPO (Recovery Point Objective) и RTO (Recovery Time Objective). Стратегии бэкапирования БД: полные, инкрементальные, логические (pg_dump) vs физические. План восстановления (Runbook/Playbook). Проведение учебных тренировок по восстановлению.	
--	--	--

6.2. Перечень вопросов, заданий, тем для подготовки к текущему контролю

Примерные вопросы контрольной работы (5 семестр)

1. Опишите основные роли в современной команде веб-разработки (Frontend, Backend, DevOps, QA) и их ключевые обязанности.
2. Объясните, что такое клиент-серверная архитектура. Перечислите основные этапы жизненного цикла HTTP-запроса.
3. В чем заключается принцип семантической верстки? Приведите примеры семантических HTML-тегов и объясните их назначение.
4. Что такое CSS селектор? Перечислите основные виды селекторов (по типу, классу, id, атрибуту) и объясните их специфичность.
5. Объясните разницу между let, const и var в JavaScript. В чем особенности их области видимости (scope) и поднятия (hoisting)?
6. Что такое Promise в JavaScript? Опишите три возможных состояния Promise и приведите пример его использования с then() и catch().
7. В чем ключевое различие между Flexbox и CSS Grid? Укажите типичные сценарии использования каждого модуля.
8. Опишите концепцию Virtual DOM. Какой проблеме она призвана решить и как работает в React?
9. Что такое props и state в React? Объясните разницу между ними и правила их изменения.
10. Объясните назначение и принципы REST. Перечислите основные HTTP-методы, используемые в RESTful API, и их назначение (GET, POST, PUT, DELETE, PATCH).

Примерные задания контрольной работы (5 семестр)

1. HTML/CSS: На основе предоставленного макета (текстового описания или скриншота) создайте семантически верную HTML-разметку для карточки товара интернет-магазина и стилизуйте ее с использованием CSS (Flexbox/Grid).

Карточка должна содержать изображение, название, описание, цену и кнопку «В корзину».

2. JavaScript (DOM, события): Напишите функцию на чистом JavaScript, которая динамически добавляет новый пункт в список `<ul>` при нажатии на кнопку «Добавить». Пункт должен содержать текст из поля ввода `<input>`. Реализуйте также возможность удаления любого пункта списка по клику на нем.

3. JavaScript (Асинхронность, API): Используя `fetch()` и публичное API (например, JSONPlaceholder), напишите код, который загружает и отображает список постов (как минимум `title` и `body`) на странице. Реализуйте обработку ошибок сети и состояния загрузки (`loading`).

4. React (Компоненты, состояние): Создайте простой компонент React «Счетчик». Компонент должен отображать текущее значение счетчика и две кнопки: «+1» (увеличивает значение) и «Сброс» (обнуляет значение). Используйте хук `useState`.

5. SQL (Базы данных): Даны таблицы `users(id, name)` и `orders(id, user_id, amount, created_at)`. Напишите SQL-запрос, который выводит имя пользователя и общую сумму всех его заказов за последний месяц (предположим, текущая дата - 2024-12-16). Отсортируйте результат по убыванию суммы.

6. REST API (Проектирование): Спроектируйте REST API для управления библиотекой книг. Опишите эндпоинты, HTTP-методы и предполагаемые структуры JSON-запросов/ответов для операций:

- Получить список всех книг.
- Добавить новую книгу.
- Обновить информацию о конкретной книге (например, количество доступных экземпляров).
- Удалить книгу.

Примерные вопросы контрольной работы (6 семестр)

1. Сравните механизмы аутентификации на основе сессий (`cookies`) и JWT (токенов). Перечислите преимущества и недостатки каждого подхода.

2. Опишите механизм CSRF-атаки (Cross-Site Request Forgery). Какие способы защиты от нее существуют на стороне backend и frontend?

3. Что такое SQL-инъекция? Объясните принцип атаки и приведите пример защищенного SQL-запроса с использованием параметризованных запросов (prepared statements).
4. Объясните процесс разрешения конфликта слияния (merge conflict) в системе Git. Опишите последовательность действий разработчика.
5. Какие основные виды автоматизированного тестирования (unit, integration, e2e) применяются в веб-разработке? Укажите, что тестирует каждый вид и приведите примеры инструментов.
6. В чем заключается идея контейнеризации (Docker)? Объясните разницу между образом (Image) и контейнером (Container).
7. Опишите структуру и назначение ключевых инструкций в простом Dockerfile (FROM, WORKDIR, COPY, RUN, CMD).
8. Раскройте принципы CI (Continuous Integration) и CD (Continuous Delivery/Deployment). Какую пользу они приносят процессу разработки?
9. Перечислите ключевые метрики производительности веб-приложений (Core Web Vitals или другие). Какие инструменты используются для их измерения?
10. Что должно быть включено в финальную презентацию и документацию итогового проекта?

Примерные задания контрольной работы (6 семестр)

1. Node.js + Express + JWT: Напишите фрагмент кода для маршрута POST /api/login, который принимает email и password, проверяет пользователя в БД (условно), и в случае успеха возвращает JWT-токен с полезной нагрузкой (payload), содержащей userId и email. Используйте библиотеку jsonwebtoken.
2. Git (Ветвление и слияние): Опишите последовательность команд Git для реализации простого рабочего процесса:
 - Создать новую ветку feature/add-search от main.
 - Сделать несколько коммитов в новой ветке.
 - Переключиться обратно на main и получить последние изменения с удаленного репозитория (git pull).

- Влить изменения из feature/add-search в main через merge.

3. Тестирование (Jest): Напишите модульный тест (unit test) с использованием Jest для функции `formatDate(timestamp)`, которая принимает `timestamp` и возвращает строку в формате "DD.MM.YYYY".

4. Docker (Dockerfile): Напишите базовый Dockerfile для Node.js приложения. Предположим, точка входа — `server.js`. Включите шаги: использование официального образа Node, установку зависимостей из `package.json`, копирование исходного кода, объявление порта и команду для запуска.

5. CI/CD (GitHub Actions): Опишите конфигурацию (.yml файл) простого пайплайна GitHub Actions, который запускается при пуше в ветку `main` и выполняет две задачи: 1) установку зависимостей (`npm ci`) и 2) запуск юнит-тестов (`npm test`).

6. Оптимизация и безопасность (Ситуационная задача): Вам предоставили код эндпоинта `GET /api/users/search?q=...`, который ищет пользователей по имени. Код уязвим для SQL-инъекции и не имеет лимита на количество возвращаемых записей. Предложите и напишите исправленную, безопасную версию этого эндпоинта (псевдокод или код на знакомом языке), добавив пагинацию (лимит, смещение).

Критерии балльной оценки различных форм текущего контроля успеваемости содержатся в соответствующих методических рекомендациях Кафедры информационных технологий Факультета информационных технологий и анализа больших данных.

7. Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине

Перечень компетенций с указанием индикаторов их достижения в процессе освоения образовательной программы содержится в разделе 2. *Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине.*

Типовые контрольные задания или иные материалы, необходимые для оценки индикаторов достижения компетенций, умений и знаний

ПКН-2 Способность разрабатывать алгоритмы и программы с использованием современных технологий программирования

1) Владеет объектно-ориентированным языком программирования на уровне знания синтаксиса и семантики, основ стандартной библиотеки

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: принципы клиент-серверного взаимодействия (REST/GraphQL), методологии проектирования UI/UX, основы информационной безопасности веб-приложений, шаблоны проектирования (MVC, MVVM и др.)

Уметь: анализировать техническое задание, выбирать подходящий стек технологий, создавать макеты интерфейсов (wireframes), проектировать API эндпоинты и структуру базы данных

Типовые контрольные задания

На основе предоставленного технического задания (ТЗ) спроектируйте архитектуру веб-приложения для онлайн-голосования. Предоставьте: блок-схему взаимодействия клиента и сервера. ER-диаграмму базы данных, спецификацию ключевых API методов (метод, URL, параметры, ответ).

2) Использует инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки)

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: синтаксис и парадигмы выбранных языков программирования (JavaScript/Python/Java и др.), фреймворки (React/Vue/Angular; Express/Django/Spring), системы контроля версий (Git), основы работы с базами данных (SQL/NoSQL)

Уметь: писать чистый, модульный и документированный код; интегрировать frontend и backend части; работать с внешними API; выполнять операции CRUD с базой данных

Типовые контрольные задания

Разработайте backend на Node.js/Express с REST API для управления списком задач (добавление, чтение, обновление, удаление). Реализуйте простой frontend на React, который взаимодействует с этим API и отображает список задач с возможностью отметки выполнения.

3) Организует кодовую базу, ориентируется в существующем коде, демонстрирует знание общепринятых соглашений и политик в области оформления кода

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: виды тестирования (юнит-тесты, интеграционные, e2e), инструменты тестирования (Jest, Mocha, Cypress, Selenium), принципы TDD/BDD

Уметь: писать юнит-тесты для функций и компонентов, настраивать интеграционное тестирование API, использовать инструменты отладки

Типовые контрольные задания

Напишите набор юнит-тестов (минимум 5) для функции валидации email на стороне backend. Настройте и выполните e2e-тест для сценария. Пользователь добавляет новый элемент в список

4) Проектирует текстовый, программный или графический интерфейс программной системы исходя из ее назначения

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: процессы сборки (build), контейнеризация (Docker), основы работы веб-серверов (Nginx), облачные платформы (VPS, Heroku, AWS Elastic Beanstalk/Yandex Cloud), CI/CD-пайплайны

Уметь: настраивать переменные окружения, создавать Docker-образ приложения, выполнять деплой на удаленный сервер, читать логи сервера

Типовые контрольные задания

Запакуйте разработанное веб-приложение в Docker-контейнер. Опишите последовательность команд для развертывания этого контейнера на удаленном VPS-сервере с использованием Nginx в качестве reverse proxy. Предоставьте конфигурационные файлы (Dockerfile, nginx.conf).

**ПКП-2 Способность разрабатывать, согласовывать и управлять исполнением
технического задания и технического проекта с использованием технологий больших
данных**

1) Работает со стандартами, в том числе адаптирует стандарты для специфических требований больших данных.

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: структура и требования ГОСТ 34, IEEE 830 или корпоративных стандартов к технической документации; жизненный цикл данных (Data Lifecycle Management); типовые архитектурные паттерны для Big Data (Lambda, Kappa, Data Lake, Data Mesh); критерии выбора технологий больших данных (Hadoop, Spark, Kafka, NoSQL БД) в зависимости от типа данных (структурированные, полуструктурированные, неструктурированные) и задач (обработка в реальном времени, пакетная обработка); методы сбора и формализации требований заинтересованных сторон (стейкхолдеров)

Уметь: проводить интервью с бизнес-пользователями для выявления и анализа потребностей в работе с данными; декомпозировать бизнес-требования на технические спецификации; описывать функциональные и нефункциональные требования (производительность, масштабируемость, отказоустойчивость) к системам больших данных; формализовать критерии приемки (Acceptance Criteria) для этапов проекта; составлять глоссарий терминов в предметной области

Типовые контрольные задания

На основе описания бизнес-процесса интернет-магазина (необходимость анализировать поток кликов в реальном времени, хранить исторические данные о покупках за 5 лет, строить рекомендательные модели) разработайте раздел «Требования к системе» Технического задания. Включите: цели системы, функциональные требования к сбору, обработке и хранению данных, требования к производительности (RPS, latency), перечень стейкхолдеров и критерии приемки для MVP.

2) Разрабатывает технические задания и технические проекты для технологий больших данных.

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: принципы работы и области применения компонентов экосистемы Big Data (HDFS, YARN, Spark Core/Streaming/SQL, Flink, Kafka, HBase, Cassandra, Elasticsearch и др.); модели согласованности данных (CAP-теорема) и их влияние на выбор СУБД; методы проектирования Data Pipeline (конвейеров данных); подходы к обеспечению качества данных (Data Quality) на этапе проектирования; принципы безопасности данных (Data Security) и нормативные требования (152-ФЗ, GDPR)

Уметь: выбирать и комбинировать технологии из стека Big Data для решения конкретной задачи; проектировать схемы данных, включая форматы хранения (Parquet, Avro, ORC) и стратегии партиционирования; разрабатывать архитектурные диаграммы (Data Flow, Component Diagram) с использованием нотаций (C4, UML); рассчитывать требования к инфраструктуре (кластерная мощность, объемы хранения, сетевые bandwidth); оценивать технологические риски и предлагать митигирующие действия

Типовые контрольные задания

Для ТЗ из предыдущего задания спроектируйте архитектурное решение. Создайте диаграмму потока данных (Data Flow Diagram). Обоснуйте выбор каждого технологического компонента (например, Kafka для стриминга, Spark для ETL, Cassandra для хранения). Опишите схему партиционирования данных в хранилище. Рассчитайте ориентировочные требования к инфраструктуре на первые 3 года эксплуатации.

3) Реализует управление рабочими проектами технологической инфраструктуры больших данных.

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: методологии управления требованиями (Requirements Management); техники фасилитации и проведения согласовательных встреч (workshops); форматы и инструменты для совместной работы над документацией (Confluence, Wiki, Notion); процедуры управления изменениями (Change Management Process), включая оценку влияния изменений (Impact Analysis); основы коммуникации и аргументации в технической среде

Уметь: проводить презентации технических решений для технической и нетехнической аудитории; формулировать аргументированные ответы на замечания по проекту; вести протоколы совещаний и регистр замечаний; оценивать стоимость и сроки при внесении изменений в требования; использовать системы контроля версий для технической документации (Git для docs-as-code)

Типовые контрольные задания

Вам поступил запрос от заказчика на изменение в ТЗ: добавить возможность обработки данных из нового источника (голосовые записи кол-центра) и интеграцию с внешней ВІ-системой. Проведите анализ влияния изменений (Impact Analysis). Подготовьте презентацию для согласования с заказчиком, в которой оцените увеличение бюджета и сроков, риски и альтернативные варианты реализации.

Примеры практико-ориентированных заданий

Задание 1. Карточка товара для маркетплейса

- Контекст: Вам нужно сверстать типовой блок товара для интернет-магазина.
- Задача: Создать семантически правильный блок с:
 - Изображением товара (с тегом alt).
 - Названием (заголовок).
 - Рейтингом (звёзды из юникода ★).
 - Кратким описанием.
 - Ценой (старая и новая со скидкой).
 - Кнопкой "В корзину".
- Требования: Адаптивность. На мобильном — карточка на всю ширину, на планшете — две в ряд, на десктопе — три в ряд (использовать flex-wrap или grid). При наведении на кнопку — плавное изменение фона.
- Результат: Файлы index.html, style.css. Проверка валидатором HTML/CSS.

Задание 2. "Умный" калькулятор чаевых

- Контекст: Приложение для посетителей кафе.
- Задача: Создать форму с полями:
 1. Сумма счёта (input[type="number"]).
 2. Количество человек (input[range] или number).
 3. Кнопки для выбора процента чаевых (10%, 15%, 20%, custom).
- Логика на JS: При изменении любого поля в реальном времени пересчитывать и отображать:
 - Сумма чаевых с человека.
 - Итоговая сумма к оплате с человека.
- Требования: Валидация ввода (нельзя ввести буквы, отрицательные числа). Аккуратный UI с использованием CSS.
- Результат: Работрующий интерактивный калькулятор.

Задание 3. Виджет погоды с выбором города

- Контекст: Необходимо добавить на сайт виджет погоды.
- Задача:
 1. Получить данные о погоде с открытого API (например, OpenWeatherMap). Использовать `async/await` и `fetch`.
 2. Отобразить: город, температуру, иконку погоды, описание.
 3. Добавить `<select>` с 3-5 городами. При выборе — виджет должен обновляться без перезагрузки страницы.
 4. Реализовать обработку ошибок (нет сети, город не найден) с понятным сообщением для пользователя.
- Требования: Использовать современный JS (ES6+ модули). Элементы интерфейса создавать динамически через JS. Стилизовать виджет.
- Результат: Готовый виджет, работающий с реальным API.

Примерные вопросы для подготовки к зачету, экзамену

Примерные вопросы для подготовки к зачету

1. Современная веб-разработка как процесс. Опишите жизненный цикл типового веб-проекта и ключевые роли в команде.
2. Клиент-серверная архитектура. Объясните принцип взаимодействия по протоколу HTTP. Структура запроса и ответа.
3. Принципы семантической верстки (HTML5). Приведите примеры семантических тегов и объясните их влияние на SEO и доступность.
4. Каскад и наследование в CSS. Объясните, как вычисляется специфичность селектора.
5. Блочная модель (box model) в CSS. Из каких свойств состоит и как влияет на размер элемента?
6. Основы JavaScript: типы данных, объявление переменных (`let`, `const`, `var`), область видимости.

7. Работа с DOM в JavaScript. Методы поиска элементов, манипуляция содержимым и атрибутами.
8. События (Events) в JavaScript. Модель распространения событий (всплытие и перехват). Делегирование событий.
9. Асинхронность в JavaScript. Что такое Promise? Объясните синтаксис async/await.
10. Принципы адаптивной верстки. Что такое медиазапросы (media queries)? Подходы Mobile First vs Desktop First.
11. Модули CSS для компоновки: Flexbox и CSS Grid. Основные свойства и примеры применения.
12. Компонентный подход во фронтенд-разработке. Концепция Virtual DOM и ее преимущества.
13. React: компоненты, состояние (state) и свойства (props). Правила работы с ними.
14. Хуки в React: назначение и правила использования. Объясните работу useState и useEffect.
15. Принципы REST. Перечислите основные HTTP-методы и их назначение при проектировании API.
16. Реляционные базы данных (SQL). Основные операции CRUD. Простые SQL-запросы с SELECT, WHERE, ORDER BY.
17. Связи между таблицами в реляционной БД. Объясните типы связей (один-ко-многим, многие-ко-многим).
18. Нормализация баз данных: цели и суть первой нормальной формы (1NF).
19. Node.js и Express.js: назначение и базовые концепции (маршрутизация, middleware).

Примерные вопросы для подготовки к экзамену

1. Аутентификация и авторизация в веб-приложениях. Сравните подходы на основе сессий и JWT-токенов.
2. Механизм работы JSON Web Token (JWT). Из каких частей состоит токен и как обеспечивается его целостность?
3. OAuth 2.0: основные роли (клиент, ресурсный сервер, сервер авторизации) и суть потока Authorization Code.

4. Основные угрозы безопасности веб-приложений (OWASP Top 10). Детально опишите механизм XSS-атаки и способы защиты.
5. SQL-инъекция: принцип, пример уязвимого кода и способы защиты (параметризованные запросы, ORM).
6. CSRF-атака (межсайтовая подделка запроса). Объясните механизм и методы защиты (CSRF-токены, SameSite cookies).
7. Система контроля версий Git. Опишите процесс работы с ветками (ветвление, слияние, разрешение конфликтов).
8. Принципы тестирования программного обеспечения. Сравните модульное (unit), интеграционное (integration) и сквозное (e2e) тестирование.
9. Контейнеризация с Docker. Объясните разницу между образом, контейнером и реестром.
10. Назначение и структура Dockerfile и docker-compose.yml. Приведите пример базового Dockerfile для веб-приложения.
11. Непрерывная интеграция и поставка (CI/CD). Какие задачи решает и как выглядит типичный пайплайн?
12. Метрики производительности веб-приложений (Core Web Vitals: LCP, FID, CLS). Что они измеряют и как их улучшить?
13. Кэширование в веб-приложениях. Какие стратегии кэширования вы знаете (на стороне клиента и сервера)?
14. Межсайтовые запросы (CORS). Почему браузер блокирует такие запросы по умолчанию и как настроить сервер для их разрешения?
15. Безопасные заголовки HTTP (HSTS, CSP, X-Frame-Options). Для чего они используются?
16. Процесс деплоя веб-приложения. Опишите этапы от сборки проекта до его развертывания на облачном сервере.
17. Мониторинг приложения после деплоя. Какие ключевые метрики важно отслеживать (uptime, error rate, latency)?
18. Архитектурные решения для обеспечения отказоустойчивости и масштабируемости веб-приложения.

19. Структура и требования к итоговому проекту. Какие разделы должны быть включены в техническую документацию и презентацию?

Пример экзаменационного билета

Экзаменационный билет №

1. Теоретический вопрос. (10 баллов)

Аутентификация и авторизация. Сравните механизмы аутентификации на основе серверных сессий (с использованием cookies) и JWT (JSON Web Tokens). Опишите преимущества и недостатки каждого подхода. Объясните, как обеспечивается безопасность передачи и хранения JWT на клиенте.

2. Практико-ориентированный вопрос. (20 баллов)

Безопасность и тестирование. Дан фрагмент кода серверного маршрута на Node.js (Express), который выполняет поиск пользователей по имени:

javascript

```
app.get('/api/users/search', (req, res) => { const name =  
req.query.name; const query = `SELECT * FROM users WHERE name LIKE  
'%${name}%'`; db.query(query, (err, results) =>  
{ res.json(results); });});
```

Code

а) Выявите уязвимость (укажите тип) и объясните возможные последствия её эксплуатации.

б) Предложите и напишите безопасную версию этого кода.

в) Сформулируйте два модульных теста (unit test) для безопасной версии функции поиска. Опишите, что они должны проверять.

3. Ситуационная задача (кейс). (30 баллов)

Архитектура и DevOps. Вы завершили разработку небольшого full-stack приложения (SPA на React + REST API на Node.js + PostgreSQL). Опишите пошаговый план его промышленного развертывания (деплой), начиная с этапа подготовки кода и заканчивая мониторингом. В плане обязательно отразите: подготовку окружения (контейнеризацию); настройку CI/CD пайплайна (укажите 3-4 ключевых этапа); развертывание на облачной инфраструктуре (укажите примеры сервисов); базовые меры по обеспечению безопасности и наблюдаемости.

8. Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины

Основная литература:

1. Лехмус, М.Ю. Базовые технологии веб-программирования : Учебное пособие / М.Ю. Лехмус Электрон. дан. Москва : КноРус, 2025 86 с. Режим доступа: book.ru Internet access <https://book.ru/book/956632> ISBN 978-5-406-14103-8. [БИК ID: RU\bookru\bibl\956632]
2. Шаталова, А.Ю. Основы веб-разработки : Учебное пособие / А.Ю. Шаталова, М.В. Коротеев Электрон. дан. Москва : КноРус, 2025 282 с. Режим доступа: book.ru Internet access <https://book.ru/book/957272> ISBN 978-5-406-14458-9. [БИК ID: RU\bookru\bibl\957272]

Дополнительная литература:

1. Биткина, И. В. Методические указания по выполнению аналитического задания веб-квест «Временный творческий коллектив» [Электронный ресурс] : учебное пособие / Биткина И. В. Москва : Прометей, 2019 26 с. Книга из коллекции Прометей - Экономика и менеджмент <https://e.lanbook.com/book/116176> ISBN 978-5-907100-29-9. [БИК ID: RU-LAN-BOOK-116176]
2. Лехмус, М.Ю. Тестовая поддержка базовых технологий веб-программирования. Часть 1 : Учебное пособие / М.Ю. Лехмус Электрон. дан. Москва : Русайнс, 2026 103 с. Режим доступа: book.ru Internet access <https://book.ru/book/959020> ISBN 978-5-466-09333-9. [БИК ID: RU\bookru\bibl\959020]

9. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. <https://www.planetaexcel.ru/>
2. Электронная библиотека Финансового университета (ЭБ) <http://elib.fa.ru/> (<http://library.fa.ru/files/elibfa.pdf>)
3. Электронно-библиотечная система BOOK.RU <http://www.book.ru>
4. Электронно-библиотечная система "Университетская библиотека ОНЛАЙН" <http://biblioclub.ru/>
5. "Деловая онлайн библиотека" издательства "Альпина Паблишер" <http://lib.alpinadigital.ru/en/library>

6. Электронно-библиотечная система издательства "Лань" <https://e.lanbook.com/>

10. Методические указания для обучающихся по освоению дисциплины

Технология поэтапного освоения дисциплины

ШАГ 1: Теоретическое погружение

- Изучите презентации, лекционные материалы.
- Посмотрите 1-2 коротких видеоурока по теме.
- Составьте глоссарий из 10-15 ключевых терминов модуля.

ШАГ 2: Микропрактика

- Выполните задания на интерактивных платформах:
- HTML/CSS: [HTML Academy](#) , [CSS Grid Garden](#)
- JavaScript: [FreeCodeCamp](#) , [Codewars](#) (5-7 kyu)
- Algorithms: [LeetCode](#) (Easy задачи)
- Создайте "песочницу" (Sandbox) на CodePen или CodeSandbox для быстрых экспериментов.

ШАГ 3: Интеграционное задание

- Выполните основное практическое задание модуля (лабораторную работу).
- Важно: Сначала напишите код самостоятельно, только затем сравнивайте с эталоном или ищите решение.
- При возникновении ошибок:
 1. Прочитайте сообщение об ошибке.
 2. Изучите с помощью поисковика текст ошибки (на английском).
 3. Проверьте код через отладчик (debugger) или console.log().
 4. Обратитесь к документации (MDN Web Docs, официальная док. фреймворка).

11. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень

необходимого программного обеспечения и информационных справочных систем

Комплект лицензионного программного обеспечения:

1. Web сервер node.js версии не ниже 18 LTS
2. Figma
3. Miro
4. Tilda
5. Kaspersky

Современные профессиональные базы данных и информационные справочные системы:

1. Электронная энциклопедия: <http://ru.wikipedia.org/wiki/Wiki>
2. Размещение ЭУК: <https://www.campus.fa.ru/>
3. Информационно-правовая система «Гарант»
4. Информационно-правовая система «Консультант Плюс»

Сертифицированные программные и аппаратные средства защиты информации:

1. не предусмотрены

12. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине

1. **Учебная аудитория** для проведения учебных занятий, предусмотренных программой, в том числе групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, оснащенная оборудованием и техническими средствами обучения: мебель аудиторная (столы, стулья, доска аудиторная), набор демонстрационного оборудования (проектор, экран)
2. **Компьютерный класс** для проведения учебных занятий, предусмотренных программой, в том числе групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, оснащенный оборудованием и техническими средствами обучения: мебель аудиторная (столы, стулья, доска аудиторная), персональные компьютеры, набор демонстрационного оборудования (проектор, экран)