МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ГРОЗНЕНСКИЙ ГОСУДАРСТВЕННЫЙ НЕФТЯНОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА М.Д.МИЛЛИОНЩИКОВА»

Информационные системы в экономике
(наименование кафедры)

УТВЕРЖДЕН

на заседании кафедры
«02» 09 2026 г., протокол № 1

Заведующий кафедрой
_____ Л.Р. Магомаева
(подпись)

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«Облачные сервисы и серверная разработка мобильных приложений
(наименование дисциплины)

Направление /специальность подготовки
09.03.03 Прикладная информатика
(код и наименование направления/ специальности подготовки)

Направленность (профиль)
Разработка мобильных и Web-приложений
(наименование специализации / профиля подготовки)

Квалификация
бакалавр
(специалист / бакалавр / магистр)

Составитель _____ А.С-А. Хасухаджиев
(подпись)

Грозный – 2026 г.

ПАСПОРТ
ФОНДА ОЦЕНОЧНЫХ СРЕДСТВ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ
Проектирование пользовательского интерфейса для мобильных приложений

№ п/п	Контролируемые разделы (темы) дисциплины	Код контролируемой компетенции (или ее части)	Наименование оценочного средства
1	**Тема 1.** Архитектурные паттерны для серверных приложений.	(УК-1) (ОПК-1)	Лабораторная работа
2	**Тема 2.** Управление базами данных в облачных приложениях.	(ОПК-7) (ПК-5)	Лабораторная работа
3	**Тема 3.** Аутентификация и авторизация в мобильных приложениях.	(ПК-2)	Лабораторная работа
4	**Тема 4.** Разработка API для мобильных клиентов.	(УК-1)	Лабораторная работа
5	**Тема 5.** Обработка и хранение медиа-контента.	(ОПК-5)	Лабораторная работа
6	**Тема 6.** Оптимизация производительности серверной части.	(ОПК-1) (ОПК-2)	Лабораторная работа
7	**Тема 7.** Обработка ошибок и мониторинг серверных приложений.	(ОПК-2)	Лабораторная работа
8	**Тема 8.** Безопасность серверных приложений в облачных средах.	(ПК-5)	Лабораторная работа

ПЕРЕЧЕНЬ ОЦЕНОЧНЫХ СРЕДСТВ

№ п/п	Наименование оценочного средства	Краткая характеристика оценочного средства	Представление оценочного средства в фонде
1	*Лабораторная работа*	Средство проверки умений применять полученные знания по заранее определенной методике для решения задач или заданий по модулю или дисциплине в целом	Комплект заданий для выполнения лабораторных работ

2	*Рубежный контроль*	Форма проверки знаний по дисциплине в виде первой и второй рубежных аттестаций	Вопросы к аттестациям
3	*зачет*	Итоговая форма оценки знаний	Вопросы к зачету
4	*экзамен*	Итоговая форма оценки знаний	Вопросы к экзамену

ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

В качестве оценочных средств используются средства контроля выполнения лабораторных работ по дисциплине. Защита лабораторной работы – ответ на контрольные вопросы после выполнения лабораторной работы.

Средства текущего контроля: устный опрос (собеседование/опрос, разбор учебной ситуации на выбранную тему, подготовка устных сообщений и докладов), практическое задание (выполнение заданий в электронной форме на ПК).

Лабораторная работа 1.	*Цель лабораторной работы:* «Локальные сервера» (УК-1) (ОПК-1)
Лабораторная работа 2.	*Цель лабораторной работы:* «AJAX и общение с сервером» (ОПК-7) (ПК-5)
Лабораторная работа 3.	*Цель лабораторной работы:* «Работа с Promise (ES6)» (ОПК-2)
Лабораторная работа 4.	*Цель лабораторной работы:* «Получение данных с сервера. Async/Await (ES8)» (УК-1)
Лабораторная работа 5.	*Цель лабораторной работы:* «Как сохранить данные без БД. Работа с localStorage» (ОПК-5)
Лабораторная работа 6.	*Цель лабораторной работы:* «Настройка Express + nodemon + TypeScript» (ОПК-1) (ОПК-2)
Лабораторная работа 7.	*Цель лабораторной работы:* «Развёртывание проекта» (ОПК-2)
Лабораторная работа 8.	*Цель лабораторной работы:* «Express middleware» (ПК-5)

Критерии оценки ответов на лабораторные работы (4-й семестр)

Регламентом БРС предусмотрено всего 15 баллов за текущую работу студента. Критерии оценки разработаны, исходя из возможности ответа студентом до 8 лабораторных работ с использованием дополнительного материала по ним. (по 3 балла).

3 балла ставится за работу, выполненную полностью без ошибок и недочетов.

2 балл ставится за работу, выполненную полностью, но при наличии в ней негрубых ошибок и недочетов.

1 балл ставится за работу, выполненную полностью, но не сданную в срок, и при наличии в ней негрубых ошибок и недочетов.

0 баллов ставится, если студент совсем не выполнил ни одного задания.

Максимальное количество баллов за выполнение и защиту лабораторных работ 12.

Максимально высокие баллы за текущий контроль – 15. За активное участие на лекциях и использование дополнительного материала при подготовке к занятиям студент получает дополнительно до 3 баллов.

ГРОЗНЕНСКИЙ ГОСУДАРСТВЕННЫЙ НЕФТЯНОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА М.Д.МИЛЛИОНЩИКОВА

Институт цифровой экономики и технологического предпринимательства
Кафедра «Информационные системы в экономике»

Вопросы к первой рубежной аттестации
«Облачные сервисы и серверная разработка мобильного рпиложения»

1. Чем отличается монолитная архитектура от микросервисной?
2. Какие преимущества предоставляет микросервисная архитектура по сравнению с монолитной?
3. Какие недостатки могут быть у микросервисной архитектуры и как их решить?
4. Что такое сервepless архитектура и какие преимущества она предоставляет?
5. Какие задачи лучше всего подходят для реализации с помощью сервepless архитектуры?
6. Какие основные типы баз данных существуют, и в чем их различия?
7. Как выбрать подходящую базу данных для мобильного приложения?
8. Какие стратегии масштабирования баз данных используются в облачных приложениях?
9. Как осуществляется резервное копирование данных в облачных приложениях?
10. Какие механизмы обеспечивают высокую доступность и надежность баз данных в облачных средах?
11. Чем отличается аутентификация от авторизации?
12. Какие протоколы аутентификации используются в мобильных приложениях?
13. Как работает механизм JWT (JSON Web Token) и в каких случаях его целесообразно применять?
14. Какие преимущества предоставляет протокол OAuth для аутентификации и авторизации в мобильных приложениях?
15. Какие механизмы обеспечивают безопасность и защиту от атак при реализации аутентификации и авторизации в мобильных приложениях?

Вопросы ко второй рубежной аттестации «Проектирование пользовательского интерфейса для мобильных приложений»

1. Какие методы обработки изображений могут быть использованы на сервере для мобильных приложений?
2. Какие форматы хранения видео наиболее распространены при разработке серверной части для мобильных приложений?
3. Какие механизмы сжатия изображений помогают оптимизировать хранение медиа-контента на сервере?

4. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?

5. Какие аспекты следует учитывать при реализации механизма загрузки и скачивания медиа-контента в мобильном приложении?

6. Как кэширование данных может повысить производительность серверных приложений для мобильных клиентов?

7. Какие типы запросов можно асинхронизировать для улучшения производительности сервера?

8. Каким образом масштабирование серверных приложений влияет на их производительность?

9. Какие методы существуют для балансировки нагрузки и оптимизации производительности серверов?

10. Какие инструменты можно использовать для мониторинга производительности серверных приложений?

11. 11. Какие стратегии обработки ошибок могут быть применены в серверной части мобильных приложений?

12. Каким образом можно реализовать логирование для обнаружения и отслеживания ошибок в серверных приложениях?

13. Какие инструменты мониторинга существуют для обнаружения проблем в работе серверной части мобильных приложений?

14. Какие параметры следует отслеживать для обеспечения оптимальной производительности серверов?

15. Как можно автоматизировать процесс обнаружения и устранения проблем в серверных приложениях?

Критерии оценки ответов на рубежной аттестации

Регламентом БРС предусмотрено всего 20 баллов за рубежную аттестацию студента. Критерии оценки разработаны, исходя из возможности ответа студентом на 2 вопроса в билете (по 10 баллов).

10 баллов (5+) заслуживает студент, обнаруживший всестороннее, систематическое и глубокое знание учебного программного материала, самостоятельно выполнивший все предусмотренные программой задания, глубоко усвоивший основную и дополнительную литературу, рекомендованную программой, активно работавший на практических, семинарских, лабораторных занятиях, разбирающийся в основных научных концепциях по изучаемой дисциплине, проявивший творческие способности и научный подход в понимании и изложении учебного программного материала, ответ отличается богатством и точностью использованных терминов, материал излагается последовательно и логично.

9 баллов (5) заслуживает студент, обнаруживший всестороннее, систематическое знание учебного программного материала, самостоятельно выполнивший все предусмотренные программой задания, глубоко усвоивший основную литературу и знаком с дополнительной литературой, рекомендованной программой, активно работавший на практических, семинарских, лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы, а также способность к их самостоятельному пополнению, ответ отличается точностью использованных терминов, материал излагается последовательно и логично.

8 баллов (4+) заслуживает студент, обнаруживший полное знание учебно-программного материала, не допускающий в ответе существенных неточностей, самостоятельно выполнивший все предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, активно работавший на практических, семинарских, лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы, а также способность к их самостоятельному пополнению.

7 баллов (4) заслуживает студент, обнаруживший достаточно полное знание учебно-программного материала, не допускающий в ответе существенных неточностей, самостоятельно выполнивший все предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, активно работавший на практических, семинарских, лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы, а также способность к их самостоятельному пополнению.

6 баллов (4-) заслуживает студент, обнаруживший достаточно полное знание учебно-программного материала, не допускающий в ответе существенных неточностей, самостоятельно выполнивший основные предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, отличавшийся достаточной активностью на практических (семинарских) и лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы.

5 баллов (3+) заслуживает студент, обнаруживший знание основного учебно-программного материала в объёме, необходимом для дальнейшей учебы и предстоящей работы по профессии, не отличавшийся активностью на практических (семинарских) и лабораторных занятиях, самостоятельно выполнивший основные предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, однако допустивший некоторые погрешности при их выполнении и в ответе па экзамене, но обладающий необходимыми знаниями для их самостоятельного устранения.

4 балла (3) заслуживает студент, обнаруживший знание основного учебно-программного материала в объёме, необходимом для дальнейшей учебы и предстоящей работы по профессии, не отличавшийся активностью на практических (семинарских) и лабораторных занятиях, самостоятельно выполнивший основные предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, однако допустивший некоторые погрешности при их выполнении и в ответе на экзамене, но обладающий необходимыми знаниями для устранения под руководством преподавателя допущенных погрешностей.

3 балла (3-) заслуживает студент, обнаруживший знание основного учебно-программного материала в объёме, необходимом для дальнейшей учебы и предстоящей работы по профессии, не отличавшийся активностью на практических (семинарских) и лабораторных занятиях, самостоятельно выполнивший основные предусмотренные программой задания, однако допустивший погрешности при их выполнении и в ответе на экзамене, но обладающий необходимыми знаниями для устранения под руководством преподавателя наиболее существенных погрешностей.

2 балла (2) выставляется студенту, обнаружившему пробелы в знаниях или отсутствие знаний по значительной части основного учебно-программного материала, не выполнившему самостоятельно предусмотренные программой основные задания, допустившему принципиальные ошибки в выполнении предусмотренных программой заданий, не отработавшему основные практические, семинарские, лабораторные занятия, допускающему существенные ошибки при ответе, и который не может продолжить обучение или приступить к профессиональной деятельности без дополнительных занятий по соответствующей дисциплине.

1 балл — нет ответа (отказ от ответа, представленный ответ полностью не по существу содержащихся в экзаменационном задании вопросов)

ГРОЗНЕНСКИЙ ГОСУДАРСТВЕННЫЙ НЕФТЯНОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА М.Д.МИЛЛИОНЩИКОВА

Институт цифровой экономики и технологического предпринимательства
Кафедра информационные системы в экономике

Вопросы к зачету по дисциплине «Проектирование пользовательского интерфейса для мобильных приложений»

1. Чем отличается монолитная архитектура от микросервисной?
2. Какие преимущества предоставляет микросервисная архитектура по сравнению с монолитной?
3. Какие недостатки могут быть у микросервисной архитектуры и как их решить?
4. Что такое сервepless архитектура и какие преимущества она предоставляет?
5. Какие задачи лучше всего подходят для реализации с помощью сервepless архитектуры?
6. Какие основные типы баз данных существуют, и в чем их различия?
7. Как выбрать подходящую базу данных для мобильного приложения?
8. Какие стратегии масштабирования баз данных используются в облачных приложениях?
9. Как осуществляется резервное копирование данных в облачных приложениях?
10. Какие механизмы обеспечивают высокую доступность и надежность баз данных в облачных средах?
11. Чем отличается аутентификация от авторизации?
12. Какие протоколы аутентификации используются в мобильных приложениях?
13. Как работает механизм JWT (JSON Web Token) и в каких случаях его целесообразно применять?
14. Какие преимущества предоставляет протокол OAuth для аутентификации и авторизации в мобильных приложениях?
16. Какие механизмы обеспечивают безопасность и защиту от атак при реализации аутентификации и авторизации в мобильных приложениях?
15. Какие методы обработки изображений могут быть использованы на сервере для мобильных приложений?
16. Какие форматы хранения видео наиболее распространены при разработке серверной части для мобильных приложений?
17. Какие механизмы сжатия изображений помогают оптимизировать хранение медиа-контента на сервере?
18. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?
19. Какие аспекты следует учитывать при реализации механизма загрузки и скачивания медиа-контента в мобильном приложении?

20. Как кэширование данных может повысить производительность серверных приложений для мобильных клиентов?

21. Какие типы запросов можно асинхронизировать для улучшения производительности сервера?

22. Каким образом масштабирование серверных приложений влияет на их производительность?

23. Какие методы существуют для балансировки нагрузки и оптимизации производительности серверов?

24. Какие инструменты можно использовать для мониторинга производительности серверных приложений?

25. Какие стратегии обработки ошибок могут быть применены в серверной части мобильных приложений?

26. Каким образом можно реализовать логирование для обнаружения и отслеживания ошибок в серверных приложениях?

27. Какие инструменты мониторинга существуют для обнаружения проблем в работе серверной части мобильных приложений?

28. Какие параметры следует отслеживать для обеспечения оптимальной производительности серверов?

29. Как можно автоматизировать процесс обнаружения и устранения проблем в серверных приложениях?

Критерии оценки ответов на зачете

Регламентом БРС предусмотрено 20 баллов (максимальный балл) за ответ на вопросы в билете. Критерии оценки разработаны, исходя из возможности ответа студентом на 2 вопроса в билете (по 10 баллов).

10 баллов (5+) заслуживает студент, обнаруживший всестороннее, систематическое и глубокое знание учебного программного материала, самостоятельно выполнивший все предусмотренные программой задания, глубоко усвоивший основную и дополнительную литературу, рекомендованную программой, активно работавший на практических, семинарских, лабораторных занятиях, разбирающийся в основных научных концепциях по изучаемой дисциплине, проявивший творческие способности и научный подход в понимании и изложении учебного программного материала, ответ отличается богатством и точностью использованных терминов, материал излагается последовательно и логично.

9 баллов (5) заслуживает студент, обнаруживший всестороннее, систематическое знание учебного программного материала, самостоятельно выполнивший все предусмотренные программой задания, глубоко усвоивший основную литературу и знаком с дополнительной литературой, рекомендованной программой, активно работавший на практических, семинарских, лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы, а также способность к их самостоятельному пополнению, ответ отличается точностью использованных терминов, материал излагается последовательно и логично.

8 баллов (4+) заслуживает студент, обнаруживший полное знание учебно-программного материала, не допускающий в ответе существенных неточностей, самостоятельно выполнивший все предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, активно работавший на практических, семинарских, лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы, а также способность к их самостоятельному пополнению.

7 баллов (4) заслуживает студент, обнаруживший достаточно полное знание учебно-программного материала, не допускающий в ответе существенных неточностей, самостоятельно выполнивший все предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, активно работавший на практических, семинарских, лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы, а также способность к их самостоятельному пополнению.

6 баллов (4-) заслуживает студент, обнаруживший достаточно полное знание учебно-программного материала, не допускающий в ответе существенных неточностей, самостоятельно выполнивший основные предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, отличавшийся достаточной активностью на практических (семинарских) и лабораторных занятиях, показавший систематический характер знаний по дисциплине, достаточный для дальнейшей учебы.

5 баллов (3+) заслуживает студент, обнаруживший знание основного учебно-программного материала в объёме, необходимом для дальнейшей учебы и предстоящей

работы по профессии, не отличавшийся активностью на практических (семинарских) и лабораторных занятиях, самостоятельно выполнивший основные предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, однако допустивший некоторые погрешности при их выполнении и в ответе па экзамене, но обладающий необходимыми знаниями для их самостоятельного устранения.

4 балла (3) заслуживает студент, обнаруживший знание основного учебно-программного материала в объёме, необходимом для дальнейшей учебы и предстоящей работы по профессии, не отличавшийся активностью на практических (семинарских) и лабораторных занятиях, самостоятельно выполнивший основные предусмотренные программой задания, усвоивший основную литературу, рекомендованную программой, однако допустивший некоторые погрешности при их выполнении и в ответе на экзамене, но обладающий необходимыми знаниями для устранения под руководством преподавателя допущенных погрешностей.

3 балла (3-) заслуживает студент, обнаруживший знание основного учебно-программного материала в объёме, необходимом для дальнейшей учебы и предстоящей работы по профессии, не отличавшийся активностью на практических (семинарских) и лабораторных занятиях, самостоятельно выполнивший основные предусмотренные программой задания, однако допустивший погрешности при их выполнении и в ответе на экзамене, но обладающий необходимыми знаниями для устранения под руководством преподавателя наиболее существенных погрешностей.

2 балла (2) выставляется студенту, обнаружившему пробелы в знаниях или отсутствие знаний по значительной части основного учебно-программного материала, не выполнившему самостоятельно предусмотренные программой основные задания, допустившему принципиальные ошибки в выполнении предусмотренных программой заданий, не отработавшему основные практические, семинарские, лабораторные занятия, допускающему существенные ошибки при ответе, и который не может продолжить обучение или приступить к профессиональной деятельности без дополнительных занятий по соответствующей дисциплине.

1 балл — нет ответа (отказ от ответа, представленный ответ полностью не по существу содержащихся в экзаменационном задании вопросов)

Критерии оценки знаний студента на зачете

Оценка «зачтено» - выставляется студенту, показавшему фрагментарный, разрозненный характер знаний, варьируемых от оценки «отлично» до «удовлетворительно». При этом он владеет основными разделами учебной программы, необходимыми для дальнейшего обучения и может применять полученные знания по образцу в стандартной ситуации.

Оценка «не зачтено» - выставляется студенту, который не знает большей части основного содержания учебной программы дисциплины, допускает грубые ошибки в формулировках основных понятий дисциплины и не умеет использовать полученные знания при решении типовых практических задач.

Приложение 1

КОМПЛЕКТ ЗАДАНИЙ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1

Цель лабораторной работы: «Локальные сервера». JSON формат передачи данных, глубокое клонирование объектов.

Что такое Node.js и как она работает

До появления Node.js приложения, которые написаны на языке программирования JavaScript, можно было запускать только в браузере. С появлением платформы стало возможно писать на JavaScript не только в браузере, но и на сервере.

Платформу разработал Райан Дал, программист из Америки, в 2009 году. Она работает на движке V8, транслирующем JavaScript в машинный код. Простыми словами, Node.js — это приложение C++, которое получает на входе код JavaScript и выполняет его. Чтобы взаимодействовать с устройствами ввода-вывода на компьютере, в платформе предусмотрен собственный интерфейс на C++. Таким образом, платформа превращает специализированный скриптовый язык JavaScript в язык общего назначения, поэтому на Node.js можно писать любые компьютерные программы. Node.js позволяет пользоваться единым языком JavaScript для написания кода и на стороне клиента (Frontend), и на сервере (Backend). Эти возможности Node.js важны для разработки приложений реального времени, которые основаны на событиях.

Для чего нужен Node.js

Платформу используют fronted-разработчики, backend-разработчики и другие. Она позволяет написать программу для разных ОС: Linux, OS X и Windows, может использоваться для создания API. Также Node.js применяется для разработки кросс-платформенных приложений: например, списка задач, который должен работать на разных платформах, синхронизировать данные в реальном времени и отправлять на мобильное устройство. Node.js используется при создании сервисов с постоянным обменом информацией с пользователем: социальных сетей, онлайн-игр, чатов, систем совместной работы над проектом, онлайн-редакторов текста и пр.

Установка Node.js

Первый шаг — установка консоли. На Windows уже предустановлен терминал cmd.exe, но как основное «место работы» он не подходит. В качестве аналога используется эмулятор консоли, встроенный в VS Code терминал. После этого шага с официального сайта Node.js необходимо скачать последнюю версию платформы.

Рис 1 Установка Node JS

Теперь можно начать работу в Node JS. Для этого создайте файл с расширением **.js** и откройте его в VS Code. Для проверки напишем небольшую программу в виде таймера, которая будет выводить новое значение каждую секунду, пока мы не остановим программу.

```
let counter = 0;


const timer = setInterval(() => {
    counter++;
    console.log(counter);
}, 1000)
```

Для вывода результата откройте терминал и введите следующую команду:

`node` *название вашего файла*.js

Остановить действие программы можно сочетанием клавиш ***ctrl + c.***

Создание собственного сервера в Node JS

```
const http = require('http');


const server = http.createServer((request, responce) => {
    responce.write('hello world')
    responce.end()
})


server.listen(3003)
```

Для того, чтобы читать информацию на собственном сервере, нам необходимо подключить модуль, отвечающий за отправку и принятие запросов.

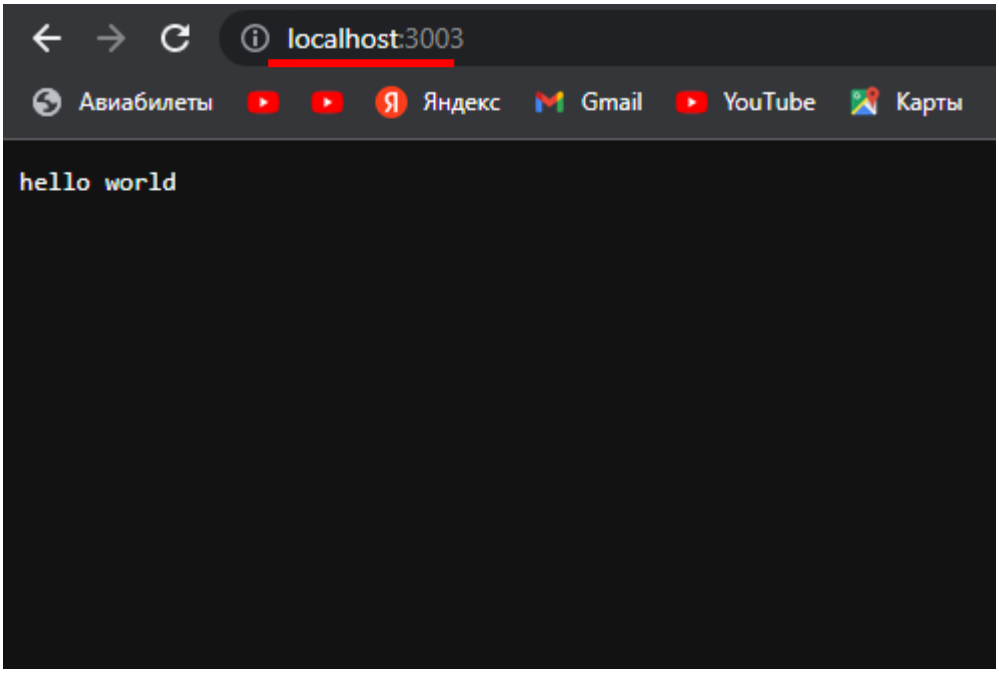
Дело в том, что Node.js состоит из блоков, называемых модулями; и каждый отдельный файл по своей сути — отдельный блок (модуль), чья область видимости изолирована от других таких же блоков (модулей) служит для подключения одного независимого модуля (файла) к другому независимому модулю (файлу). Принцип подключения через *require* заключается в создании в целевом модуле объекта и помещении в этот объект подключаемого модуля. В нашем случае этот модуль отвечает за http-запросы.

Команда .createServer позволяет создать новый сервер, куда мы передаем два параметра (request – запрос, responce - ответ).

С помощью метода write('hello world') мы выводим информацию на сервер.

Для того, чтобы запустить программу необходимо прописать команду, с которым мы уже знакомы: node название вашего файла.js

И указать в браузере адрес сервера, который был указан в `server.listen(3003)`



Самостоятельно:

- поменяйте выводимое значение на любое другое;
- добавьте к этому значению новое, постоянно меняющееся значение с помощью таймера, который реализовали ранее;
- поменяйте выводимое значение в зависимости от URL – сервера, например:

Если URL равен http://localhost:3003/ выводите ‘hello world’, если http://localhost:3003/home выводите сообщение ‘i`ts my home address’ и т.д.

Лабораторная работа 2.

Цель лабораторной работы: «AJAX и общение с сервером»

Реализация скрипта отправки данных на сервер. Красивое оповещение пользователя.

JSON формат передачи данных

JSON – это набор пар ключ-значение. В качестве значений могут быть объекты, массивы, числа, строки, логические значения или *null.*

Для примера возьмем объект:

```
const person = {
    name: 'Alex',
    tel: '+72332233231'
};
```

Преобразовать javascript объект в формат JSON можно при помощи команды *stringify*

```
console.log(JSON.stringify(person));
```

Запустите программу и посмотрите на результат.

Преобразовать JSON в объект javascript можно при помощи команды parse

```
console.log(JSON.parse(JSON.stringify(person)));
```

Глубокое клонирование объектов

Клонирование объектов осуществляется комбинацией методов parse и stringify.

```
const person = {
    name: 'Alex',
    tel: '+72332233231',
    parents: {
        mom: 'Olga',
        dad: 'Mike'
    }
};
```

```
const clone = JSON.parse(JSON.stringify(person));
clone.parents.mom = 'Ann';
console.log(clone);
console.log(person);
```

AJAX и общение с сервером

Встроенный объект в браузер `XMLHttpRequest`

`XMLHttpRequest` это API, который предоставляет клиенту функциональность для обмена данными между клиентом и сервером. Данный API предоставляет простой способ получения данных по ссылке без перезагрузки страницы. Это позволяет обновлять только часть веб-страницы не прерывая пользователя. `XMLHttpRequest` используется в AJAX запросах и особенно в single-page приложениях.

Перед тем, как мы начнем по ближе знакомиться с `XMLHttpRequest`, сверстайте небольшую страничку с двумя формами (input), как показано на **рис. 1**

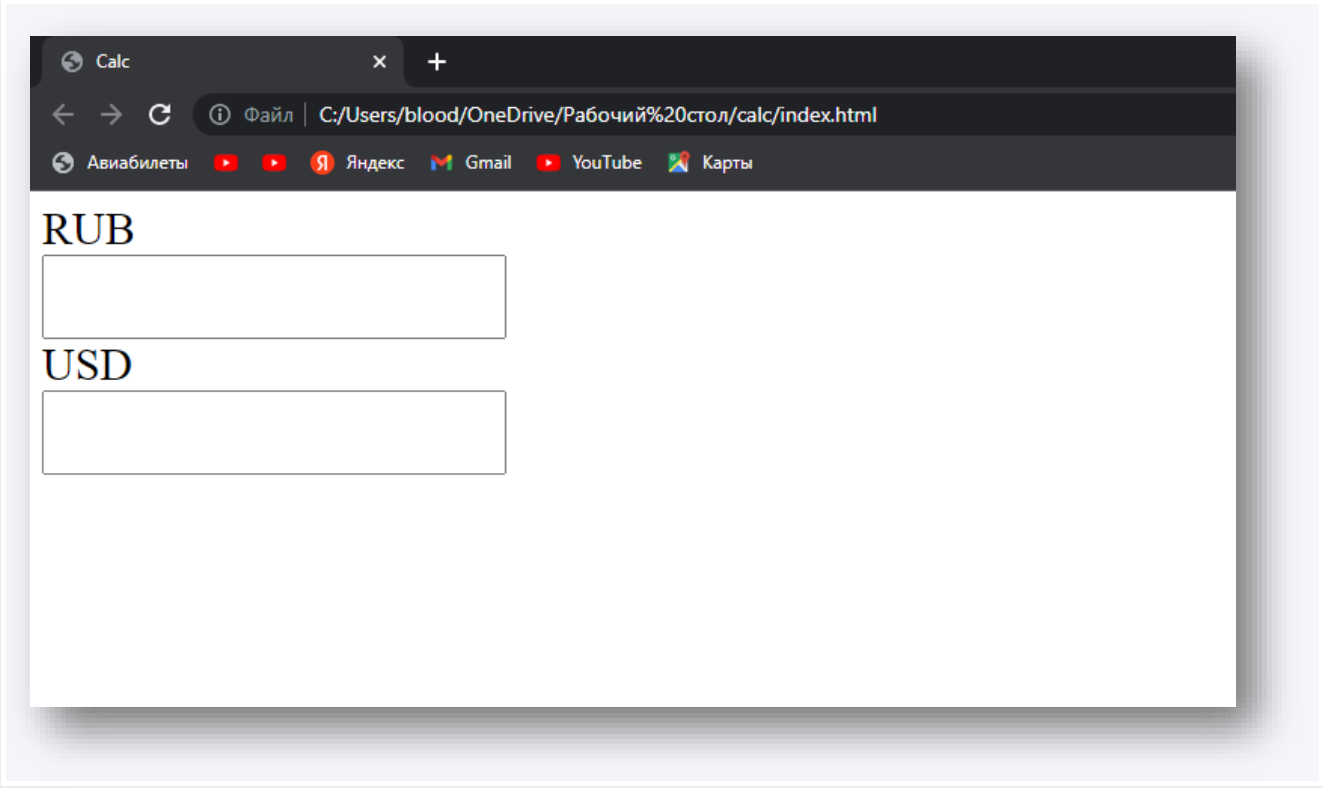


Рис. 1

После того, как сверстали страничку, свяжем ее с JS документом и получим доступ к формам на странице:

```
const inputRub = document.querySelector('#rub'),
      inputUsd = document.querySelector('#usd');
```

Теперь, необходимо повесить обработчик события на inputRub:

```
inputRub.addEventListener('input', () => {
      const request = new XMLHttpRequest();


}
```

Первым методом вызывается `open`, который собирает необходимые для запроса настройки.

```
      request.open(method, url, async, login, pass);
```

- первый параметром задается метод (GET, POST и т.п.)
- вторым аргументом задается путь к серверу
- третьим параметром мы определяем асинхронность (true или false)
- последними двумя аргументами указываются логин и пароль

Давайте, теперь применим этот метод в нашем коде, указывая реальные данные:

```
inputRub.addEventListener('input', () => {
      const request = new XMLHttpRequest();

      request.open('GET', 'js/current.json');


}
```

На данный момент нам достаточно указать метод и путь к серверу. Затем указывается заголовок и отправляем запрос:

```
inputRub.addEventListener('input', () => {
      const request = new XMLHttpRequest();

      request.open('GET', 'js/current.json');
```

```
        request.setRequestHeader('Content-type', 'application/json; charset=utf-8');
//заголовок
        request.send();
}
```

В качестве параметров `send` можно указать какие-то данные которые будут отправляться на сервер (например, при POST запросе).

В качестве ответа от сервера мы получаем:

- status: статус (404, 200 и т.п.)
- statusText: текстовое описание ответа от сервера
- response: ответ от сервера
- readyState: текущее состояние нашего запроса

После того, как мы данные получили, соответственно, мы уже можем оперировать с ними. Повесим обработчик событий на наш объект, который будет отслеживать статус готовности нашего запроса в данный момент времени.

```
inputRub.addEventListener('input', () => {
        const request = new XMLHttpRequest();

        request.open('GET', 'js/current.json');
        request.setRequestHeader('Content-type', 'application/json; charset=utf-8');
//заголовок
        request.send();

        request.addEventListener('load', () => {

          if (request.status === 200) {
              const data = JSON.parse(request.response);
              inputUsd.value = (+inputRub.value / data.current.usd).toFixed(2);
          } else {
              inputUsd.value = "Что-то пошло не так";
          }
      });

}
```

Лабораторная работа 3.

Цель лабораторной работы: «Работа с Promise (ES6)»

Fetch API. Методы перебора массивов.

JavaScript может отправлять сетевые запросы на сервер и подгружать новую информацию по мере необходимости.

Например, мы можем использовать сетевой запрос, чтобы:

- отправить заказ;
- загрузить информацию о пользователе;
- запросить последние обновления с сервера;
- …и т.п.

Для сетевых запросов из JavaScript есть широко известный термин «AJAX» (аббревиатура от Asynchronous JavaScript And XML). XML мы использовать не обязаны, просто термин старый, поэтому в нём есть это слово. Возможно, вы его уже где-то слышали.

Есть несколько способов делать сетевые запросы и получать информацию с сервера.

Метод fetch() — современный и очень мощный, поэтому начнём с него. Он не поддерживается старыми (можно использовать полифил), но поддерживается всеми современными браузерами.

Базовый синтаксис:

```
let promise = fetch(url, [options])
```

- url – URL для отправки запроса.
- options – дополнительные параметры: метод, заголовки и так далее.

Без options это простой GET-запрос, скачивающий содержимое по адресу url.

Браузер сразу же начинает запрос и возвращает промис, который внешний код использует для получения результата.

Процесс получения ответа обычно происходит в два этапа.

Во-первых, promise выполняется с объектом встроенного класса Response в качестве результата, как только сервер пришлёт заголовки ответа.

На этом этапе мы можем проверить статус HTTP-запроса и определить, выполнился ли он успешно, а также посмотреть заголовки, но пока без тела ответа.

Промис завершается с ошибкой, если fetch не смог выполнить HTTP-запрос, например при ошибке сети или если нет такого сайта. HTTP-статусы 404 и 500 не являются ошибкой.

Мы можем увидеть HTTP-статус в свойствах ответа:

- status – код статуса HTTP-запроса, например 200.
- ok – логическое значение: будет true, если код HTTP-статуса в диапазоне 200-299.

Например:

```
let response = await fetch(url);

if (response.ok) { // если HTTP-статус в диапазоне 200-299
  // получаем тело ответа (см. про этот метод ниже)
  let json = await response.json();
} else {
  alert("Ошибка HTTP: " + response.status);
}
```

Во-вторых, для получения тела ответа нам нужно использовать дополнительный вызов метода.

Response предоставляет несколько методов, основанных на промисах, для доступа к телу ответа в различных форматах:

- response.text() – читает ответ и возвращает как обычный текст;
- response.json() – декодирует ответ в формате JSON;
- response.formData() – возвращает ответ как объект FormData (разберём его в следующей главе);
- response.blob() – возвращает объект как Blob (бинарные данные с типом);
- response.arrayBuffer() – возвращает ответ как ArrayBuffer (низкоуровневое представление бинарных данных);
- помимо этого, response.body – это объект ReadableStream, с помощью которого можно считывать тело запроса по частям. Мы рассмотрим и такой пример несколько позже.

Например, получим JSON-объект с последними коммитами из репозитория на GitHub:

```
let url = 'https://api.github.com/repos/javascript-tutorial/en.javascript.inf
let response = await fetch(url);

let commits = await response.json(); // читаем ответ в формате JSON

alert(commits[0].author.login);
```

То же самое без await, с использованием промисов:

```
fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/co
  .then(response => response.json())
  .then(commits => alert(commits[0].author.login));
```

Для получения ответа в виде текста используем await response.text() вместо .json():

```
let response = await fetch('https://api.github.com/repos/javascript-tutorial/

let text = await response.text(); // прочитать тело ответа как текст

alert(text.slice(0, 80) + '...');
```

В качестве примера работы с бинарными данными, давайте запросим и выведем на экран логотип спецификации «fetch» (см. главу Blob, чтобы узнать про операции с Blob):

```
let response = await fetch('/article/fetch/logo-fetch.svg');

let blob = await response.blob(); // скачиваем как Blob-объект

// создаём <img>
let img = document.createElement('img');
img.style = 'position:fixed;top:10px;left:10px;width:100px';
document.body.append(img);

// выводим на экран
img.src = URL.createObjectURL(blob);

setTimeout(() => { // прячем через три секунды
  img.remove();
  URL.revokeObjectURL(img.src);
}, 3000);
```

Самостоятельно:

1. **Получение и вывод данных о погоде:**

Создайте веб-приложение, которое запрашивает у пользователя название города и затем делает сетевой запрос к API погоды (например, OpenWeatherMap). Используйте метод fetch(), чтобы получить данные о текущей погоде для данного города. Выведите полученную информацию о погоде на странице.

2. **Аутентификация и авторизация:**

Реализуйте форму входа для пользователя на вашем веб-сайте. При отправке формы произведите аутентификацию через сетевой запрос к вашему серверу. Используйте метод fetch(), чтобы отправить данные формы (логин и пароль) на сервер и получить ответ о успешной аутентификации или ошибке. В зависимости от ответа сервера, выведите соответствующее сообщение пользователю.

3. **Поиск и отображение информации о фильмах:**

Создайте поле ввода, в которое пользователь может ввести название фильма. При отправке формы сделайте сетевой запрос к API фильмов (например, OMDB API) с использованием метода fetch(). Получите данные о фильме по его названию и выведите информацию о фильме на странице (например, название, год выпуска, описание).

4. **Добавление товара в корзину:**

Разработайте интерфейс для интернет-магазина, где пользователь может просматривать товары и добавлять их в корзину. При добавлении товара в корзину сделайте сетевой запрос к серверу, чтобы добавить товар в базу данных или сессию пользователя. Используйте метод fetch(), чтобы отправить данные о добавленном товаре на сервер. После успешного добавления товара выведите сообщение об успешном добавлении.

5. **Обновление данных без перезагрузки страницы:**

Создайте страницу с динамически обновляемым содержимым. Используйте сетевые запросы с методом fetch(), чтобы периодически получать новые данные с сервера (например, новости, обновления, уведомления). Обновляйте содержимое страницы с помощью полученных данных без перезагрузки всей страницы.

Лабораторная работа 4.

Цель лабораторной работы: «Получение данных с сервера. Async/Await (ES8)»

Дополнительно: Что такое библиотеки. Библиотека axios.

Переход на async / await

Итак, как же приступить к использованию async / await в своих проектах? Если вы уже работаете с промисами, значит вы готовы к переходу на новую технологию. Любая функция, которая возвращает промис, может быть вызвана с использованием ключевого слова await, что приведёт к тому, что она вернёт результат разрешения промиса. Однако, если вы собираетесь переходить на async / await с коллбэков, вам понадобится сначала преобразовать их в промисы.

Переход на async / await с промисов

Если вы один из тех, кто оказался в первых рядах разработчиков, принявших промисы, и в вашем коде, для реализации асинхронной логики, используются цепочки .then(), переход на async / await затруднений не вызовет: нужно лишь переписать каждую конструкцию .then() с использованием await.

Кроме того, блок .catch() надо заменить на стандартные блоки try / catch. Как видите, наконец-то мы можем использовать один и тот же подход для обработки ошибок в синхронном и асинхронном контекстах!

Важно отметить ещё и то, что ключевое слово await нельзя использовать на верхнем уровне модулей. Оно должно использоваться внутри функций, объявленных с ключевым словом async.

```
let hn = require('@datafire/hacker_news').create();

// Старый код с промисами:
Promise.resolve()
  .then(_ => hn.getStories({storyType: 'top'}))
  .then(storyIDs => hn.getItem({itemID: storyIDs[0]}))
  .then(topStory => console.log(topStory))
  .catch(e => console.error(e))

// Новый код с async / await:
(async () => {
  try {
    let storyIDs = await hn.getStories({storyType: 'top'});
    let topStory = await hn.getItem({itemID: storyIDs[0]});
    console.log(topStory);
  }  catch (e) {
    console.error(e);
  }
})();
```

Переход на async / await с коллбэков

Если в вашем коде всё ещё применяются функции обратного вызова, лучший способ перехода на async / await заключается в предварительном преобразовании коллбэков в промисы. Затем, используя вышеописанную методику, код, использующий промисы, переписывают с использованием async / await. О том, как преобразовывать коллбэки в промисы, можно почитать здесь.

Паттерны и подводные камни

Конечно, новые технологии — это всегда и новые проблемы. Вот несколько полезных шаблонов и типовых ошибок, с которыми вы можете столкнуться, переводя свой код на async / await.

Оптимизация

При использовании async / await вам больше не нужно думать о том, что пишете вы асинхронный код. Это прекрасно, но тут кроется и самая опасная ловушка новой технологии. Дело в том, что при таком подходе можно забыть о мелочах, которые способны оказать

огромное влияние на производительность.

Рассмотрим пример. Предположим, мы хотим получить сведения о двух пользователях Hacker News и сравнить их карму. Вот обычная реализация:

```
let hn = require('@datafire/hacker_news').create();

(async () => {

  let user1 = await hn.getUser({username: 'sama'});
  let user2 = await hn.getUser({username: 'pg'});

  let [more, less] = [user1, user2].sort((a, b) => b.karma - a.karma);
  console.log(`${more.id} has more karma (${more.karma}) than ${less.id} (${less.karma

})();
```

Код это вполне рабочий, но второй вызов getUser() не будет выполнен до тех пор, пока не завершится первый. Вызовы независимы, их можно выполнить параллельно. Поэтому ниже приведено более удачное решение:

```
let hn = require('@datafire/hacker_news').create();

(async () => {

  let users = await Promise.all([
    hn.getUser({username: 'sama'}),
    hn.getUser({username: 'pg'}),
  ]);

  let [more, less] = users.sort((a, b) => b.karma - a.karma);
  console.log(`${more.id} has more karma (${more.karma}) than ${less.id} (${less.karma

})();
```

Тут стоит отметить, что прежде чем пользоваться этим методом, стоит удостовериться в том, что желаемого можно достичь за счёт параллельного выполнения команд. Во многих случаях асинхронные операции должны выполняться последовательно.

Задание: Рефакторинг асинхронной загрузки изображений

В вашем проекте есть функция, которая загружает изображения из массива URL-адресов последовательно с помощью сетевых запросов. Вам нужно переписать эту функцию, используя async/await для параллельной загрузки изображений.

1. Начните с создания массива URL-адресов изображений, которые нужно загрузить.
2. Затем создайте асинхронную функцию, которая будет загружать изображения из массива URL-адресов параллельно.
3. Используйте метод fetch() для загрузки каждого изображения. Не забудьте обработать возможные ошибки при загрузке.
4. Для каждого успешно загруженного изображения выведите сообщение о завершении загрузки или отобразите изображение на странице.
5. После завершения всех загрузок выведите сообщение о завершении процесса загрузки.

Пример кода загрузки изображений последовательно с использованием промисов:

```
function loadImage(url) {
  return fetch(url)
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.blob();
    })
    .then(blob => {
      return URL.createObjectURL(blob);
    })
    .catch(error => {
      console.error('There was a problem with fetching the image:', error);
    });
}

function loadImagesSequentially(urls) {
```

```
  let promises = [];
  urls.forEach(url => {
    promises.push(loadImage(url));
  });

  Promise.all(promises)
    .then(images => {
      images.forEach(image => {
        console.log('Image loaded:', image);
        // Действия с загруженным изображением
      });
      console.log('All images loaded successfully!');
    })
    .catch(error => {
      console.error('There was a problem with loading images:', error);
    });
}

const imageUrls = ['url1.jpg', 'url2.jpg', 'url3.jpg'];
loadImagesSequentially(imageUrls);
```

Ваша задача - переписать функцию loadImagesSequentially() с использованием async/await для параллельной загрузки изображений.

Лабораторная работа 5.

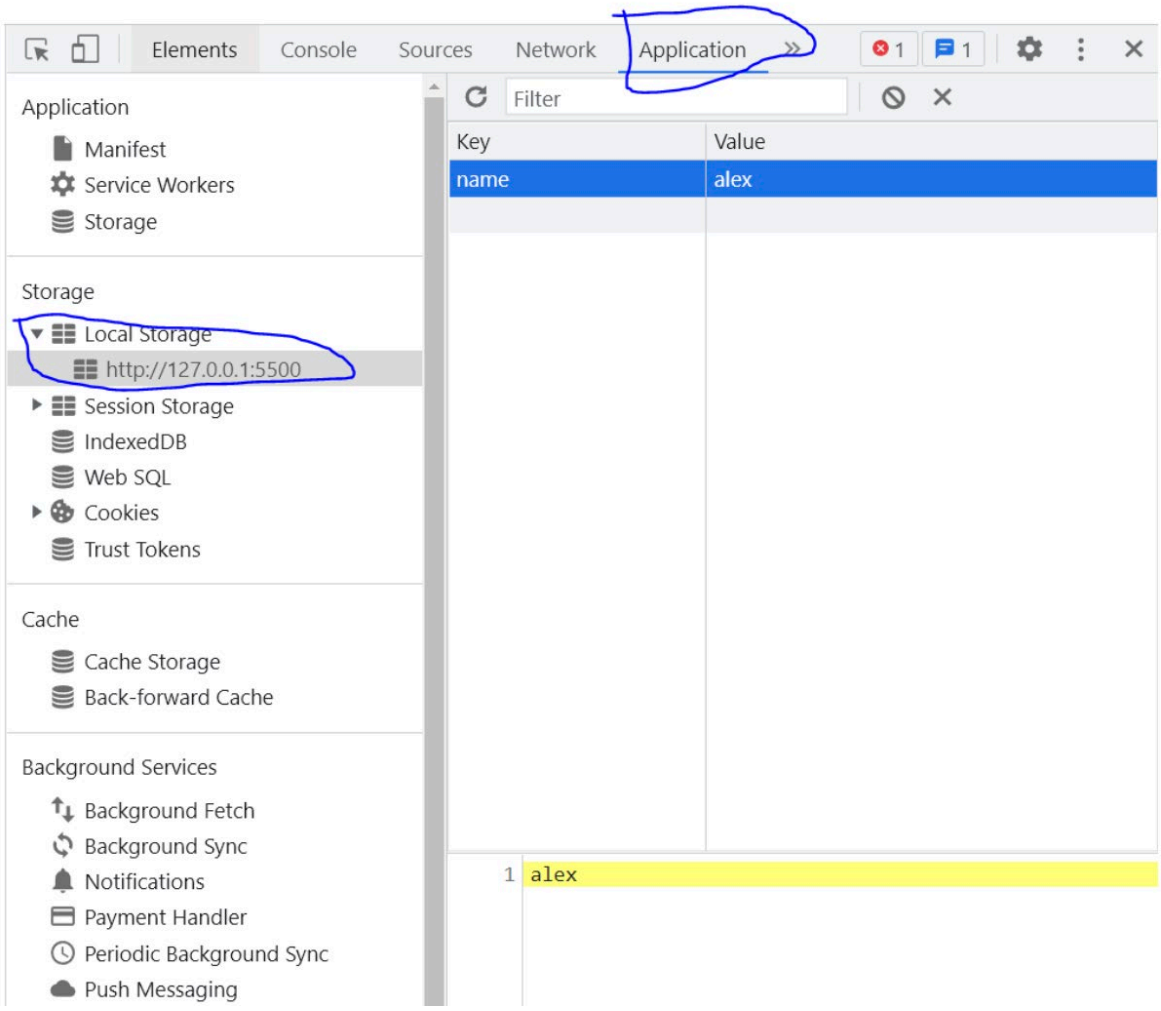
Цель лабораторной работы: «Как сохранить данные без БД. Работа с localStorage»

Создание интерфейса мобильного приложения, который адаптируется к различным устройствам и разрешениям экранов.

Чтобы сохранить данные можем использовать метод setItem(key, value) у localStorage. Этод метод принимает два аргумента: ключ по которому мы будем сохранять информацию и само значение, которое мы хотим сохранить. Например:

```
localStorage.setItem('name', 'Alex');
```

Для просмотра наших записей надо открыть DevTools там найти вкладку Application и там обязательно будет LocalStorage.



Теперь конечно же хочется получить эти данные. Для этого воспользуемся методом getItem(key). Результат можно вывести в консоль, или сохранить в переменную.

```
console.log(localStorage.getItem('name'));
```

Только строки

Как видите, все просто, но есть один маленький нюанс. Значение, которое мы записываем должно быть строкой. Давайте попробуем добавить поле age со значением 5. Для этого напишем:

```
localStorage.setItem('age', 5);
```

Сохраняем файл. И что мы видим? Ошибок никаких нет, а если перейти во вкладку Application, то мы увидим что запись прошла успешно. Это конечно же так, но при записи пятерка стала строкой, то есть была автоматически приведена к типу string. Это не то чтобы плохо, но может привести к неожиданному поведению, к примеру при строгом сравнении. Я бы советовал привести строку к числу, это можно сделать вот так:

```
const age = +localStorage.getItem('age');
```

Если с примитивами все просто, то что на счет объектов? Создадим объект user и попробуем записать его в LocalStorage:

```
const user = {
  name: 'alex',
  age: 5
};

localStorage.setItem('user', user);
```

Переходим во вкладку Application и видим [objectObject].

И есть одна небольшая проблемка у этой строки не будет полей, которые мы определили в объекте user. Это просто обычная строка. И в исходный вид это уже никак не вернуть. Для того, чтобы записать объект в localStorage надо сделать его строкой. У нас как раз есть прекрасный формат под это, и имя его json. Для преобразования объекта в строку нужно использовать JSON.stringify:

```
localStorage.setItem('user', JSON.stringify(user));
```

Теперь все записалось корректно:

Чтобы получить объект в первозданном состоянии используем метод parse у JSON:

```
JSON.parse(localStorage.getItem('user'));
```

Полученный результат можем записать в консоль, или сохранить в переменную.

Задание: Управление списком задач

Ваша задача - создать простое веб-приложение для управления списком задач. Пользователь должен иметь возможность добавлять новые задачи, отмечать их как выполненные и удалять.

1. Создайте форму для добавления новых задач. Введите поле ввода для описания задачи и кнопку "Добавить".
2. При нажатии на кнопку "Добавить" новая задача должна быть добавлена в список задач и отображена на странице.
3. Каждая задача должна иметь чекбокс для отметки выполнения и кнопку для удаления.
4. Реализуйте функционал для отметки задачи как выполненной при клике на чекбокс.
5. Реализуйте функционал для удаления задачи при клике на кнопку "Удалить".
6. Используйте LocalStorage для сохранения списка задач между сеансами. При загрузке страницы проверьте, есть ли сохраненные задачи в LocalStorage, и если есть, отобразите их на странице.
7. При каждом изменении списка задач (добавлении, удалении, изменении статуса выполнения) обновляйте данные в LocalStorage.
8. Добавьте функционал для очистки всего списка задач.

Лабораторная работа 6.

Цель лабораторной работы: «Настройка Express + nodemon + TypeScript»

Установка библиотек Express. Создание модулей и контроллеров.

Настройка сервера Express

Следующий шаг состоит в создании сервера Express в файле server.js.

Из терминала перейдите в корневую директорию и выполните следующую команду:

```
$npm init -y
```

Команда автоматически сгенерирует файл package.json. Затем нам потребуется выполнить следующую команду для установки Express, и она будет сохранена в качестве зависимости в файле package.json.

```
$npm install express --save
```

Теперь отредактируйте файл server.js следующим образом:

```
const express = require('express'); //Строка 1
const app = express(); //Строка 2
const port = process.env.PORT || 5000; //Строка 3

// Сообщение о том, что сервер запущен и прослушивает указанный порт
app.listen(port, () => console.log(`Listening on port ${port}`)); //Строка 6

// Создание GET маршрута
app.get('/express_backend', (req, res) => { //Строка 9
  res.send({ express: 'YOUR EXPRESS BACKEND IS CONNECTED TO REACT' }); //Строка 10
}); //Строка 11
```

Строки 1 и 2 - подключают модуль Express и позволяют использовать его внутри файла server.js.

Строка 3 - Установка порта, на котором будет работать сервер Express.

Строка 6 - будет отображено сообщение в консоли о том, что сервер работает исправно.

Строка 9 и 11 - установка GET маршрута, который позже мы будем получать из нашего клиентского React приложения.

Настройка proxy

На этом шаге Webpack Development Server был автоматически сгенерирован при выполнении команды create-react-app. Наше React приложение работает на Webpack Development Server на стороне фронденда.

Webpack Development Server (WDS) - это инструмент, который помогает разработчикам вносить изменения во фронтенд веб-приложения и автоматически отображает эти изменения в браузере без необходимости обновления страницы.

Он уникален по сравнению с другими инструментами, которые делают то же самое, поскольку содержимое пакета не записывается на диск в виде файлов, а хранится в памяти. Это преимущество крайне важно при отладке кода и стилей.

Мы можем настроить проксирование запросов API с клиентской стороны на API на серверной стороне. API на серверной стороне (Express сервер) будет работать на порту 5000.

Сначала настройте прокси для перехода в директорию client и найдите файл package.json. Добавьте следующую строку в него.

```
“proxy”: “http://localhost:5000”
```

Измененный файл package.json будет выглядеть следующим образом:

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
```

```
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "proxy": "http://localhost:5000"
}
```

Измененный файл package.json позволит Webpack проксировать запросы API на сервер бэкенда Express, работающий на порту 5000.

Задание: Создание простого приложения для заметок

В этом задании вы будете создавать веб-приложение для заметок, используя Express для бэкенда и React для фронтенда. Вы будете настраивать Express сервер, создавать API для работы с заметками, и настраивать проксирование запросов с клиентской стороны на сервер.

1. Настройте Express сервер, используя указанный выше код в файле server.js. Убедитесь, что сервер запускается успешно и прослушивает порт 5000.
2. Создайте API для работы с заметками. API должно поддерживать следующие операции:
   - Получение списка всех заметок.
   - Добавление новой заметки.
   - Удаление заметки по идентификатору.

3. Создайте простой интерфейс React для отображения списка заметок и добавления новых заметок. Используйте компоненты и хуки React для организации кода.

4. Настройте прокси в файле package.json вашего клиентского приложения React для перенаправления всех запросов API на Express сервер, работающий на порту 5000.

5. Реализуйте функционал для отображения списка заметок на фронтенде. Загрузите список заметок с сервера при загрузке страницы и отобразите его на странице.

6. Добавьте возможность добавления новых заметок через форму на фронтенде. После добавления новой заметки обновите список заметок на странице.

7. Добавьте функционал для удаления заметок. Реализуйте кнопку удаления для каждой заметки и обновите список заметок после удаления.

8. Проверьте работу вашего приложения, убедившись, что заметки успешно добавляются, удаляются и отображаются на странице.

Лабораторная работа 7.

Цель лабораторной работы: «Развёртывание проекта»

Развёртывание проекта на heroku, deploy.

Heroku - это облачная платформа как сервис (PaaS), которая позволяет разработчикам легко разворачивать, масштабировать и управлять веб-приложениями без необходимости управления инфраструктурой. Heroku поддерживает различные языки программирования, такие как Node.js, Ruby, Python, Java, PHP и др., что делает её популярным выбором для многих разработчиков.

Преимущества Heroku:

Простота использования: Heroku обеспечивает простой и интуитивно понятный интерфейс для развёртывания приложений. Разработчики могут быстро развернуть свои приложения, не тратя время на настройку инфраструктуры.

Масштабируемость: Heroku автоматически масштабирует ваше приложение в зависимости от нагрузки, что обеспечивает непрерывную работу ваших приложений, даже при резком увеличении трафика.

Широкие возможности: Платформа поддерживает множество языков программирования и фреймворков, что позволяет разработчикам выбирать технологии, наиболее подходящие для их проектов.

Интеграция с Git: Heroku интегрируется непосредственно с системами контроля версий Git, что упрощает развёртывание и обновление приложений.

Процесс развёртывания на Heroku:

1. Создание приложения: После регистрации на Heroku необходимо создать новое приложение через веб-интерфейс или используя Heroku CLI.
2. Подготовка приложения: Приложение должно быть настроено для успешного развёртывания на Heroku. Это включает в себя настройку зависимостей, настройку файлов конфигурации (например, Procfile), и т.д.
3. Загрузка приложения: Локальный репозиторий Git, содержащий ваше приложение, связывается с вашим приложением на Heroku, и приложение загружается на платформу с помощью команды git push heroku master.

4. Развёртывание и запуск: Heroku автоматически разворачивает ваше приложение и запускает его на серверах платформы.

5. Проверка работоспособности: После завершения развёртывания необходимо проверить, что приложение работает корректно, и что оно доступно по соответствующему URL-адресу Heroku.

6. Масштабирование: При необходимости можно настроить масштабирование приложения на Heroku, чтобы оно могло обрабатывать больше трафика или работать с большими объёмами данных.

Задание: Развёртывание приложения на Heroku

В этом задании вы будете разворачивать ваше приложение для заметок на Heroku, чтобы оно стало доступным для публичного использования.

1. Создайте аккаунт на платформе Heroku, если у вас его еще нет.
2. Установите Heroku CLI (Command Line Interface) на ваш компьютер, если еще не сделали это.
3. Создайте новое приложение на Heroku через интерфейс веб-консоли или используя команду heroku create через терминал.
4. Настройте ваше приложение для успешного развёртывания на Heroku. Убедитесь, что все необходимые файлы и зависимости (включая файл package.json) присутствуют в вашем репозитории.
5. Создайте файл Procfile, в котором определите команду для запуска вашего приложения на Heroku. Например, для приложения на Express это может быть web: node server.js.
6. Подготовьте ваше приложение к развёртыванию, убедившись, что все изменения зафиксированы в вашем репозитории Git.
7. Свяжите ваш локальный репозиторий Git с вашим приложением на Heroku, используя команду heroku git:remote -a <название вашего приложения>.
8. Выполните команду git push heroku master, чтобы загрузить ваше приложение на Heroku и развернуть его.
9. После завершения развёртывания, откройте ваше приложение в браузере, используя URL, предоставленный Heroku, и убедитесь, что оно работает корректно.
10. Протестируйте функциональность вашего приложения на Heroku, включая добавление, удаление и отображение заметок.
11. Убедитесь, что ваше приложение готово к использованию в общественном доступе.

Лабораторная работа 8.

Цель лабораторной работы: «Express middleware»

Express middleware, chain of responsibility.

Express - это веб-фреймворк маршрутизации и промежуточной обработки с минимальной собственной функциональностью: приложение Express, по сути, представляет собой серию вызовов функций промежуточной обработки.

Функции промежуточной обработки (middleware) - это функции, имеющие доступ к объекту запроса (req), объекту ответа (res) и к следующей функции промежуточной обработки в цикле “запрос-ответ” приложения. Следующая функция промежуточной обработки, как правило, обозначается переменной next.

Функции промежуточной обработки могут выполнять следующие задачи:

- выполнение любого кода;
- внесение изменений в объекты запросов и ответов;
- завершение цикла “запрос-ответ”;
- вызов следующей функции промежуточной обработки из стека.

Если текущая функция промежуточной обработки не завершает цикл “запрос-ответ”, она должна вызвать next() для передачи управления следующей функции промежуточной обработки. В противном случае запрос зависнет.

Приложение Express может использовать следующие типы промежуточных обработчиков:

- промежуточный обработчик уровня приложени;
- промежуточный обработчик уровня маршрутизатора;
- промежуточный обработчик для обработки ошибок;
- встроенные промежуточные обработчики;
- промежуточные обработчики сторонних поставщиков ПО.

Промежуточные обработчики уровня приложения и уровня маршрутизатора можно загружать с помощью необязательного пути для монтирования. Также можно загрузить последовательность функций промежуточной обработки одновременно, в результате чего создается вспомогательный стек системы промежуточных обработчиков в точке монтирования.

Промежуточный обработчик уровня приложения

Свяжите промежуточный обработчик уровня приложения с экземпляром объекта приложения, воспользовавшись функциями app.use() и app.METHOD(), где METHOD - метод HTTP запроса, обрабатываемый функцией промежуточной обработки (например, GET, PUT или POST) в нижнем регистре.

В данном примере представлена функция промежуточной обработки без пути монтирования. Эта функция выполняется при каждом получении запроса приложением.

```
var app = express();

app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

В данном примере представлена функция промежуточной обработки, монтируемая в путь /user/:id. Эта функция выполняется для всех типов запросов HTTP в пути /user/:id.

```
app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

В данном примере представлен маршрут и функция его обработки (система промежуточных обработчиков). Эта функция обрабатывает запросы GET, адресованные ресурсам в пути /user/:id.

```
app.get('/user/:id', function (req, res, next) {
  res.send('USER');
});
```

Ниже приводится пример загрузки последовательности функций промежуточной обработки в точку монтирования, с указанием пути монтирования. Этот пример иллюстрирует создание вспомогательного стека промежуточных обработчиков, с выводом информации о запросе для всех типов запросов HTTP, адресованных ресурсам в пути /user/:id.

```
app.use('/user/:id', function(req, res, next) {
  console.log('Request URL:', req.originalUrl);
  next();
}, function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

Обработчики маршрутов позволяют определить несколько маршрутов для одного пути. В приведенном ниже примере определено два маршрута для запросов GET, адресованных ресурсам в пути /user/:id. Второй маршрут не создает никаких неудобств, но его вызов никогда не будет выполнен, поскольку первый маршрут завершает цикл “запрос-ответ”.

В данном примере представлен вспомогательный стек промежуточных обработчиков для обработки запросов GET, адресованных ресурсам в пути /user/:id.

```
app.get('/user/:id', function (req, res, next) {
  console.log('ID:', req.params.id);
  next();
}, function (req, res, next) {
  res.send('User Info');
});

// handler for the /user/:id path, which prints the user ID
app.get('/user/:id', function (req, res, next) {
  res.end(req.params.id);
});
```

Для того чтобы пропустить остальные функции дополнительной обработки в стеке промежуточных обработчиков маршрутизатора, вызовите next('route') для передачи управления следующему маршруту. ПРИМЕЧАНИЕ: next('route') работает только в функциях промежуточной обработки, загруженных с помощью функций app.METHOD() или router.METHOD().

В данном примере представлен вспомогательный стек промежуточных обработчиков для обработки запросов GET, адресованных ресурсам в пути /user/:id.

```
app.get('/user/:id', function (req, res, next) {
  // if the user ID is 0, skip to the next route
  if (req.params.id == 0) next('route');
  // otherwise pass the control to the next middleware function in this stack
  else next(); //
}, function (req, res, next) {
  // render a regular page
  res.render('regular');
});

// handler for the /user/:id path, which renders a special page
app.get('/user/:id', function (req, res, next) {
  res.render('special');
});
```

Задание: Практика с функциями промежуточной обработки в Express

1. Создание промежуточного обработчика:

- Создайте новое Express-приложение.
- Добавьте промежуточный обработчик, который выводит в консоль текущее время при каждом получении запроса.

2. Промежуточный обработчик с маршрутизацией:

- Создайте промежуточный обработчик, который будет применяться к маршруту /api/*.
- Выведите в консоль информацию о типе запроса и URL при каждом обращении к маршруту /api/*.

3. Множественные обработчики маршрутов:

- Определите два маршрута для GET-запросов к /user/:id.
- Первый маршрут должен выводить в консоль идентификатор пользователя (ID).
- Второй маршрут должен просто отправлять ответ с идентификатором пользователя.

4. Пропуск функций промежуточной обработки:

- Создайте два маршрута для GET-запросов к /admin/:id.
- Первый маршрут должен проверять, является ли идентификатор администратора (например, 0).

- Если идентификатор администратора, то пропустите остальные функции промежуточной обработки и перейдите к следующему маршруту.
- Второй маршрут должен выводить сообщение о статусе пользователя (администратор или обычный пользователь).

5. Дополнительное задание:

- Используйте промежуточный обработчик для обработки ошибок (error handling middleware), который будет перехватывать ошибки и отправлять соответствующий HTTP-статус и сообщение об ошибке в ответе.

Приложение 2

КОМПЛЕКТ ОЦЕНОЧНЫХ СРЕДСТВ К РУБЕЖНОМУ КОНТРОЛЮ

Первая рубежная аттестация

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 1

1. Что такое сервepless архитектура и какие преимущества она предоставляет?

2. Какие протоколы аутентификации используются в мобильных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 2

1. Как выбрать подходящую базу данных для мобильного приложения?

2. Чем отличается аутентификация от авторизации?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 3

1. Как работает механизм JWT (JSON Web Token) и в каких случаях его целесообразно применять?

2. Какие задачи лучше всего подходят для реализации с помощью сервepless архитектуры?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 4

1. Какие стратегии масштабирования баз данных используются в облачных приложениях?

2. Какие механизмы обеспечивают высокую доступность и надежность баз данных в облачных средах?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 5

1. Как работает механизм JWT (JSON Web Token) и в каких случаях его целесообразно применять?

2. Что такое serverless архитектура и какие преимущества она предоставляет?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 6

1. Какие задачи лучше всего подходят для реализации с помощью serverless архитектуры?

2. Какие стратегии масштабирования баз данных используются в облачных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 7

1. Какие недостатки могут быть у микросервисной архитектуры и как их решить?

2. Чем отличается монолитная архитектура от микросервисной?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"

Билет № 8

1. Какие механизмы обеспечивают безопасность и защиту от атак при реализации аутентификации и авторизации в мобильных приложениях?

2. Как работает механизм JWT (JSON Web Token) и в каких случаях его целесообразно применять?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 9

1. Какие основные типы баз данных существуют, и в чем их различия?

2. Чем отличается монолитная архитектура от микросервисной?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 10

1. Чем отличается монолитная архитектура от микросервисной?

2. Какие недостатки могут быть у микросервисной архитектуры и как их решить?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 1

1. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?

2. Какие аспекты следует учитывать при реализации механизма загрузки и скачивания медиа-контента в мобильном приложении?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 2

1. Какие стратегии обработки ошибок могут быть применены в серверной части мобильных приложений?

2. Каким образом можно реализовать логирование для обнаружения и отслеживания ошибок в серверных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 3

1. Какие методы существуют для балансировки нагрузки и оптимизации производительности серверов?

2. Какие стратегии обработки ошибок могут быть применены в серверной части мобильных приложений?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 4

1. Какие инструменты мониторинга существуют для обнаружения проблем в работе серверной части мобильных приложений?

2. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 5

1. Какие методы обработки изображений могут быть использованы на сервере для мобильных приложений?

2. Какие стратегии обработки ошибок могут быть применены в серверной части мобильных приложений?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 6

1. Каким образом можно реализовать логирование для обнаружения и отслеживания ошибок в серверных приложениях?

2. Как можно автоматизировать процесс обнаружения и устранения проблем в серверных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 7

1. Какие параметры следует отслеживать для обеспечения оптимальной производительности серверов?

2. Какие типы запросов можно асинхронизировать для улучшения производительности сервера?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"

Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 8

1. Какие параметры следует отслеживать для обеспечения оптимальной производительности серверов?

2. Какие инструменты мониторинга существуют для обнаружения проблем в работе серверной части мобильных приложений?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 9

1. Какие механизмы сжатия изображений помогают оптимизировать хранение медиа-контента на сервере?

2. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 10

1. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?

2. Какие стратегии обработки ошибок могут быть применены в серверной части мобильных приложений?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 1

1. Какие параметры следует отслеживать для обеспечения оптимальной производительности серверов?

2. Какие механизмы обеспечивают безопасность и защиту от атак при реализации аутентификации и авторизации в мобильных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 2

1. Как выбрать подходящую базу данных для мобильного приложения?

2. Как осуществляется резервное копирование данных в облачных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 3

1. Какие стратегии масштабирования баз данных используются в облачных приложениях?

2. Какие механизмы обеспечивают высокую доступность и надежность баз данных в облачных средах?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 4

1. Каким образом можно обеспечить безопасность хранения медиа-контента на сервере?

2. Какие механизмы обеспечивают безопасность и защиту от атак при реализации аутентификации и авторизации в мобильных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 5

1. Чем отличается аутентификация от авторизации?

2. Каким образом можно реализовать логирование для обнаружения и отслеживания ошибок в серверных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 6

1. Какие типы запросов можно асинхронизировать для улучшения производительности сервера?

2. Какие задачи лучше всего подходят для реализации с помощью serverless архитектуры?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 7

1. Чем отличается монолитная архитектура от микросервисной?

2. Как осуществляется резервное копирование данных в облачных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"

Билет № 8

1. Какие типы запросов можно асинхронизировать для улучшения производительности сервера?

2. Какие недостатки могут быть у микросервисной архитектуры и как их решить?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 9

1. Какие основные типы баз данных существуют, и в чем их различия?

2. Какие механизмы сжатия изображений помогают оптимизировать хранение медиа-контента на сервере?

Подпись преподавателя_____ Подпись заведующего кафедрой_____

Грозненский государственный нефтяной технический университет им.акад. М.Д. Миллионщикова
Институт "Институт цифровой экономики и технологического предпринимательства"
Группа "ПИ" Семестр "7"
Дисциплина "Облачные сервисы и серверная разработка мобильных приложений"
Билет № 10

1. Как выбрать подходящую базу данных для мобильного приложения?

2. Каким образом можно реализовать логирование для обнаружения и отслеживания ошибок в серверных приложениях?

Подпись преподавателя_____ Подпись заведующего кафедрой_____
