
OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

Меня хорошо видно && слышно?

Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу


GraphQL

Определение

GraphQL – это спецификация на одноимённый декларативный язык запросов.

GraphQL был разработан Facebook

Есть несколько спецификаций GraphQL:

от Facebook (эталонная) [https://graphql.org/]

от Apollo (расширяет Facebook: добавляет subscriptions) [https://www.apollographql.com/docs/]

Пример запроса graphql

Запрос

```
{
  posts {
    title
    image
    author {
      name
    }
  }
}
```

Ответ


```
{
  "data": {
    "posts": [
      {
        "title": "Possimus numquam amet voluptate eos molestias perferendis est quis.",
        "image": "https://picsum.photos/600/400",
        "author": {
          "name": "Elouise Heller"
        }
      },
      {
        "title": "Nesciunt sint odio excepturi at nisi.",
        "image": "https://picsum.photos/600/400",
        "author": {
          "name": "Leanne Schmidt"
        }
      }
    ]
  }
}
```

REST API vs. GraphQL

REST API	GraphQL
Один URL на один ресурс	Один URL (endpoint) на все ресурсы
Только один ресурс (или массив однотипных ресурсов) на один запрос (решение: можно отойти от стандарта и сделать отдельный URL, который будет возвращать необходимое)	Все необходимые ресурсы на один запрос
Возвращаются все данные ресурса даже если они не нужны (решение: можно передавать список нужных полей в параметре запроса)	Возвращаются только нужные данные
Получение связанных ресурсов последовательно (сначала получаем id родителя, а потом делаем запрос на получение детей этого родителя) (решение: отдельный URL, который будет возвращать необходимые данные)	Получение сразу за один запрос всех необходимых связанных ресурсов любой вложенности
Легче реализовать сетевое кэширование (используя возможности HTTP)	Сетевое кэширование осложнено из-за одного URL на все ресурсы (решение: кэшировать на клиенте)
Стандарт не предусматривает наличие схемы данных (решение: дополнительно добавить схему данных, например, используя json-schema)	Схема данных лежит в основе всего
Сложнее версионировать API	"Естественное версионирование" (просто добавляется новый тип или новое поле)
Загрузка файлов по HTTP	Нет стандарта на загрузку файлов на сервер (решение: добавить свою загрузку файлов)

Структура GraphQL

На клиенте

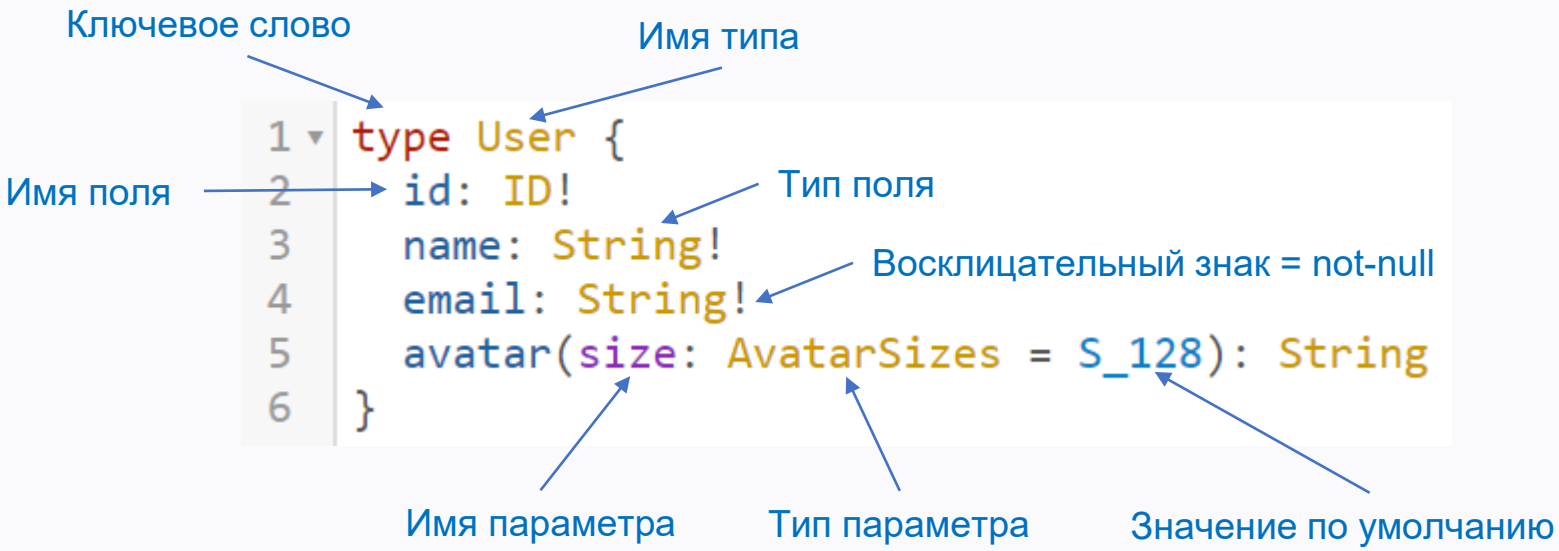
- query
- mutation
- variable
- fragment
- directive

На сервере

- type
- resolver
- input type
- enum
- interface
- directive

Структура GraphQL – Type

Type описывает схему данных ресурса/объекта/сущности. Количество типов не ограничено




```
type User {
  id: ID!
  name: String!
  email: String!
  avatar(size: AvatarSizes = S_128): String
}
```

Стандартные типы: ID (String), String, Int, Float, Boolean

Структура GraphQL – Root Types

Есть два главных корневых типа: Query и Mutation

```
type Query {
  users: [User!]!
  user(id: ID!): User
  posts: [Post!]!
  post(id: ID!): Post
  comments(postId: ID!): [Comment!]!
}

type Mutation {
  addComment(comment: CommentInput!): Comment
  addPost(post: PostInput!): Post
}
```

Оба определяют возможные запросы к серверу, только

Query выполняются параллельно, а

Mutation обязательно последовательно (для избежания race condition),

поэтому query – запросы на получение данных, а mutation – на запись/изменение.

Структура GraphQL – Input

Input имеет тот же синтаксис, что и type, только первый не может иметь свойств с параметром.

Используются в качестве типа для входных данных для мутаций

```
input PostInput {
  authorId: ID!
  title: String!
  text: String!
  image: String
}
```

Структура GraphQL – Enum

Enum – тип-перечисление, используется если значения параметра могу принимать только заранее оговорённые значения

```
enum AvatarSizes {
  S_32
  S_64
  S_128
  S_512
}
```

Структура GraphQL – Interface

Interface используется для определения списка полей, которые обязательно должны присутствовать в других типах, которые явно наследуются от интерфейса (implements).

В типах унаследованные поля должны быть явно указаны

```
interface TextWithCreatedAt {
  text: String!
  createdAt: DateTime!
}

type Post implements TextWithCreatedAt {
  id: ID!
  title: String!
  text: String!
  createdAt: DateTime!
}

type Comment implements TextWithCreatedAt {
  id: ID!
  post: Post!
  text: String!
  createdAt: DateTime!
}
```

Структура GraphQL – Resolver

Resolver – это тот код, который будет выполнен при запросе query или mutation на стороне сервера, аналог роута в веб-фреймворках

```
user({id}) {
    const user = users.find(u => u.id == id)
    if (user) {
        user['avatar'] = ({size}) => 'https://picsum.photos/'+size.substr(2)
    }

    return user
}
```

Структура GraphQL – Query

Query – это запрос, посылаемый клиентом на сервер и содержащий shape, в формате которого сервер и будет отвечать

```
{
  posts {
    id
    title
    author {
      name
    }
    image
    comments {
      text
      author {
        name
      }
    }
  }
}
```

Ответ сервера – JSON с полем data, если запрос выполнился успешно или errors, если есть ошибки

В зависимости от схемы данных (type) значением поля может быть объект, скалярное значение или массив

Структура GraphQL – Query

Query может иметь опциональное имя. Имя может использоваться сервером для логов запросов

```
query allPosts {
  posts {
    id
    title
    author {
      name
    }
    image
    comments {
      text
      author {
        name
      }
    }
  }
}
```

Структура GraphQL – Query

Можно использовать переменные в запросе. Обязательно указывать ключевое слово query и лучше для наглядности явно указывать имя запроса

```
query allUsers($avatarSize: AvatarSizes) {
  users {
    name
    avatar(size: $avatarSize)
  }
}
```

QUERY VARIABLES

```
{
  "avatarSize": "S_64"
}
```

Структура GraphQL – Mutation

Синтаксис мутаций аналогичен синтаксису query

```
mutation addCommet($comment: CommentInput!) {
  addComment(comment: $comment) {
    text
    author {
      name
    }
  }
}
```

QUERY VARIABLES

```
{
  "comment": {
    "authorId": 2,
    "postId": 3,
    "text": "This is my comment"
  }
}
```

```
{
  "data": {
    "addComment": {
      "text": "This is my comment",
      "author": {
        "name": "Jaquan Kerluke"
      }
    }
  }
}
```

Используем GraphQL во Vue приложении

```
> vue add apollo #vue2
```

Проверяем настройки src/vue-apollo.js

```
> npm install --save graphql graphql-tag @apollo/client @vue/apollo-option #vue3
```

Используем GraphQL во Vue приложении

Теперь нам доступны секция apollo, свойство $apolloData и хелпер $apollo во всех компонентах

```
import gql from 'graphql-tag';

export default {
    apollo: {
        posts: gql`{ posts {title} }`
    }
};
```

В секции apollo указываются поля модели и graphql запросы

Результат запроса добавляется в data() компонента

Имена поля результата и поля объекта должны совпадать

Используем разные имена

Явно указываем, какой объект брать из результата

```
apollo: {
  world: {
    query: gql`query {
      hello
    }`,
    update: data => data.hello
  }
}
```

Переименовываем поле в запросе

```
apollo: {
  world: gql`query {
    world: hello
  }`
}
```

Используем переменные

```
// Apollo-specific options
apollo: {
  // Query with parameters
  ping: {
    // gql query
    query: gql`query PingMessage($message: String!) {
      ping(message: $message)
    }`,
    // Static parameters
    variables: {
      message: 'Meow',
    },
  },
},
```

Флаг загрузки

You can display a loading state thanks to the `$apollo.loading` prop:

```vue
<div v-if="$apollo.loading">Loading...</div>
```

Or for this specific `ping` query:

```vue
<div v-if="$apollo.queries.ping.loading">Loading...</div>
```

Функция вместо объекта

```
// Apollo-specific options
apollo: {
  // Query with parameters
  ping () {
    // This is called once when the component is created
    // It must return the option object
    return {
      // gql query
      query: gql`query PingMessage($message: String!) {
        ping(message: $message)
      }`,
      // Static parameters
      variables: {
        message: 'Meow',
      },
    }
  },
},
```

Динамическое изменение запроса

```
// The featured tag can be either a random tag or the last added tag
featuredTag: {
  query () {
    // Here you can access the component instance with 'this'
    if (this.showTag === 'random') {
      return gql`{
        randomTag { id label type }
      }`
    } else if (this.showTag === 'last') {
      return gql`{
        lastTag { id label type }
      }`
    }
  },
  // We need this to assign the value of the 'featuredTag' component property
  update: data => data.randomTag || data.lastTag,
},
```

Реактивные переменные запроса

```
// Apollo-specific options
apollo: {
  // Query with parameters
  ping: {
    query: gql`query PingMessage($message: String!) {
      ping(message: $message)
    }`,
    // Reactive parameters
    variables () {
      // Use vue reactive properties here
      return {
          message: this.pingInput,
      }
    },
  },
},
```

Интервальное обновление результата запроса

```
// Apollo-specific options
apollo: {
  // 'tags' data property on vue instance
  tags: {
    query: gql`query tagList {
      tags {
        id,
        label
      }
    }`,
    pollInterval: 300, // ms
  },
},
```

Выключение интервального обновления запроса

```
// Apollo-specific options
apollo: {
  tags: {
    // GraphQL Query
    query: gql`query tagList ($type: String!) {
      tags(type: $type) {
        id
        label
      }
    }`,
    // Reactive variables
    variables () {
      return {
        type: this.type,
      }
    },
    // Disable the query
    skip () {
      return this.skipQuery
    },
  },
},
```

Либо

```
this.$apollo.queries.tags.skip = true
```

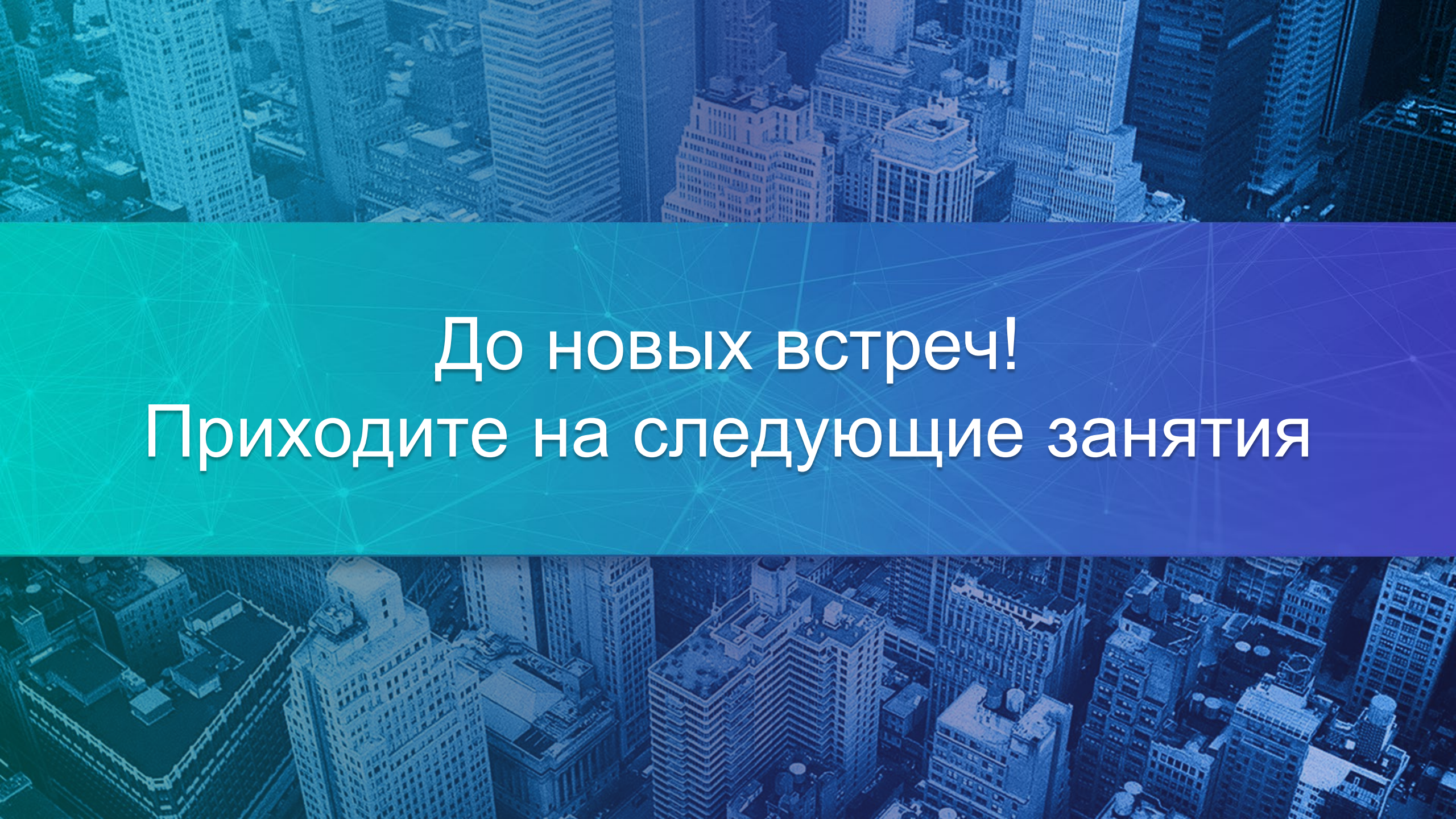
Мутации

```
methods: {
  async addTag() {
    // Call to the graphql mutation
    const result = await this.$apollo.mutate({
      // Query
      mutation: gql`mutation ($label: String!) {
        addTag(label: $label) {
          id
          label
        }
      }`,
      // Parameters
      variables: {
        label: this.newTag,
      },
    })
  }
}
```

Список материалов для изучения

- https://graphql.org/
- https://graphql.org/learn/
- https://apollo.vuejs.org/guide/
- https://www.apollographql.com/
- https://www.apollographql.com/apollo-client
- https://www.apollographql.com/docs/apollo-server/


Заполните, пожалуйста,
опрос о занятии по ссылке в чате


До новых встреч!
Приходите на следующие занятия