Тема 12. Firestore. Работа с данными из JavaScript.

хекслет колледж

Цель занятия:

Научить студентов выполнять все основные операции с Firestore из JavaScript: создавать, читать, обновлять и удалять данные (CRUD), а также делать простые запросы.

Учебные вопросы:

1. Основные CRUD операции в firestore
2. Простые запросы (фильтрация данных)

1. Основные CRUD операции в firestore

Что такое CRUD?
CRUD — это 4 базовые операции, которые можно делать с любыми данными в любом приложении:

- Create — Создать (добавить новые данные)
- Read — Прочитать (получить данные)
- Update — Обновить (изменить существующие данные)
- Delete — Удалить (убрать ненужные данные)

Это основа ЛЮБОГО приложения

Как работать с коллекциями и документами в Firestore

Метод **db.collection()** — работа с коллекциями

Что делает?

- Получает ссылку на коллекцию (папку). Это НЕ данные, а "адрес" коллекции.

Что принимает?

- Название коллекции (строка)

db.collection(‘products')

Что возвращает?

- CollectionReference — объект-ссылка на коллекцию.

Пример работы с коллекцией:

```
// Получаем ссылку на коллекцию
const productsRef = db.collection("products")

// Получаем все документы коллекции
const snapshot = await productsRef.get()

// Выводим данные
snapshot.forEach(doc => console.log(doc.id));
snapshot.forEach(doc => console.log(doc.data()));
```

```
// Добавляем данные
const newProduct = {
  name: "Samsung",
  category: "smartphones",
  price: 199000
}
await productsRef.add(newProduct);

// Слушаем изменения всей коллекции
const unsubscribe = productsRef.onSnapshot((snapshot) => {
  console.log("Данные изменились!");
  snapshot.forEach((doc) => console.log(doc.data()));
});
```

Метод db.collection().doc() — работа с документами

Что делает?

- Получает ссылку на конкретный документ (файл) в коллекции.

Что принимает?

- ID документа (строка)

```
db.collection('books').doc('book123')      // файл book123
```

Что возвращает?

- DocumentReference — объект-ссылка на документ.

```
const bookRef = db.collection('books').doc('book123')
```

Важно!

- DocumentReference ≠ данные документа!
- Это только "адрес", где документ находится или будет находиться.

Пример:

```
const docRef = db.collection("products").doc("6pSjcsO6qqoit4QrEEBn");

// Получить данные документа
const snapshot = await docRef.get();
if (snapshot.exists) {
  const data = snapshot.data();
  console.log(data.name);
}

// Создать/перезаписать документ
await docRef.set({ name: "iPhone 17", price: "499999" })
```

```
// Обновить документ (частично)
await docRef.update({ price: "540000" });

// Удалить документ
await docRef.delete()

// Слушать изменения документа
const unsubscribe = docRef.onSnapshot((snapshot) => {
  if (snapshot.exists()) {
        console.log("Данные документа изменились!");
        console.log(snapshot.data());
  }
});
```

CRUD - операции

1. CREATE — Создание данных

add() — создать с авто-ID

```
// Что делает: Создает новый документ с автоматическим ID
// Что принимает: Объект с данными
// Что возвращает: Promise с DocumentReference нового документа
const newDocRef = await db.collection('books').add({
  title: "JavaScript",
  author: "Иван",
  year: 2023
});
console.log("Создан документ с ID:", newDocRef.id); // Например:
"fH3kLm9pQrS"
```

set() — создать/перезаписать с указанным ID
// Что делает: Создает или полностью перезаписывает документ
// Что принимает: ID документа + объект с данными
// Что возвращает: Promise (успех/ошибка)

```
await db.collection('books').doc('my-book-id').set({
  title: "React",
  author: "Анна",
  year: 2024
});

// Вариант с merge (объединить, а не перезаписать):
await db.collection('books').doc('book123').set({
  year: 2025  // Только это поле изменится
}, { merge: true }); // Остальные поля сохранятся!
```

2. READ — Чтение данных

get() для коллекции — получить ВСЕ документы

Что делает:

- Получает все документы коллекции (разово)

Что возвращает:

- Promise с QuerySnapshot

```
const snapshot = await db.collection('books').get();
```

```
// Работа с результатом:
if (snapshot.empty) {
  console.log("Документов нет");
} else {
  console.log(`Нашлось ${snapshot.size} документов`);

  snapshot.forEach(doc => {
    console.log(doc.id);              // ID документа
    console.log(doc.data());          // Данные документа {title: "...", ...}
    console.log(doc.exists);          // Всегда true для коллекции
  });

  // Или как массив:
  const booksArray = snapshot.docs.map(doc => ({
    id: doc.id,
    ...doc.data()
  }));
}
```

get() для документа — получить ОДИН документ

Что делает:

- Получает один конкретный документ

Что возвращает:

- Promise с DocumentSnapshot

```
const snapshot = await db.collection('books').doc('book123').get();

if (snapshot.exists()) {
  const book = snapshot.data();  // {title: "...", author: "..."}
  console.log(book.title);
} else {
  console.log("Документ не найден");
}
```

onSnapshot() — слушать изменения в реальном времени

```
// ДЛЯ КОЛЛЕКЦИИ:
const unsubscribe =
db.collection('books').onSnapshot(snapshot => {
  // Выполнится: Сразу при подписке и  при каждом
изменении
  snapshot.forEach(doc => {
    console.log(doc.data());
  });
});
```

```
// ДЛЯ ДОКУМЕНТА:
const unsubscribe = db.collection('books').doc('book123').onSnapshot(
  snapshot => {
    if (snapshot.exists()) {
      console.log("Новые данные:", snapshot.data());
    } else {
      console.log("Документ удален");
    }
  },
  error => {
    console.error("Ошибка:", error);
  }
);

// отписаться:
unsubscribe();
```

3. UPDATE — Обновление данных

update() — частичное обновление

Что делает:

- Обновляет только указанные поля

Что принимает:

- Объект с полями для обновления

Что возвращает:

- Promise (успех/ошибка)

```
await db.collection('books').doc('book123').update({
  year: 2025,
  price: 1000,
  // Можно обновлять вложенные поля:
  "stats.views": 150,
  "tags[0]": "новый тег"
});

// Важно: Документ должен существовать! Иначе ошибка.
```

Специальные операторы обновления:

```
import { FieldValue } from "firebase/firestore"; // или
firebase.firestore.FieldValue

await db.collection('books').doc('book123').update({
  views: firebase.firestore.FieldValue.increment(1),
// Увеличить на 1
  tags: firebase.firestore.FieldValue.arrayUnion("js"),
// Добавить в массив
  oldField: firebase.firestore.FieldValue.delete(),
// Удалить поле
});
```

4. DELETE — Удаление данных

delete() — удалить документ

Что делает:

- Полностью удаляет документ

Что возвращает:

- Promise (успех/ошибка)

```
await db.collection('books').doc('book123').delete();
console.log("Документ удален");

// Важно: После удаления document.exists будет false
const snapshot = await db.collection('books').doc('book123').get();
console.log(snapshot.exists()); // false

// Удаление поля в документе:
await db.collection('books').doc('book123').update({
  oldField: firebase.firestore.FieldValue.delete()
});
```

ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ

Получение ссылок (References)

db.collection() — ссылка на коллекцию

```
const booksRef = db.collection('books');
// booksRef.id = 'books'
// booksRef.path = 'books’
```

.doc() — ссылка на документ

```
const bookRef = db.collection('books').doc('book123');
// bookRef.id = 'book123'
// bookRef.path = 'books/book123'
// bookRef.parent = CollectionReference на 'books’
```

.ref — получение ссылки из snapshot

```
const snapshot = await db.collection('books').doc('book123').get();
const docRef = snapshot.ref; // DocumentReference на тот же документ
```

2. Простые запросы (фильтрация данных)

Базовые методы запросов:

where() — фильтрация по условию

```
// Операторы: ==, !=, <, <=, >, >=, array-contains, in, array-contains-any
const query = db.collection('books')
  .where('year', '==', 2023)      // Год равен 2023
  .where('price', '<', 1000)      // Цена меньше 1000
  .where('tags', 'array-contains', 'js') // Массив содержит 'js'
  .where('author', 'in', ['Иван', 'Анна']); // Автор Иван ИЛИ Анна

const snapshot = await query.get();
```

orderBy() — сортировка

```
const query = db.collection('books')
  .orderBy('year', 'desc')      // Сначала новые (2024, 2023, ...)
  .orderBy('title', 'asc');     // Второй уровень сортировки

// Важно: Для orderBy по полю нужен индекс в Firebase Console
```

limit() — ограничение количества

```
// Только первые 10 книг
const query = db.collection('books')
  .orderBy('year', 'desc')
  .limit(10);

const snapshot = await query.get();
console.log(snapshot.size); // максимум 10
```

startAt(), endAt(), startAfter(), endBefore() — пагинация

```
// Пагинация: пропустить первые 10, взять следующие 10
const firstPage = db.collection('books')
  .orderBy('title')
  .limit(10);

const snapshot = await firstPage.get();
const lastDoc = snapshot.docs[snapshot.docs.length - 1];

const nextPage = db.collection('books')
  .orderBy('title')
  .startAfter(lastDoc)
  .limit(10);
```

Когда НУЖЕН индекс?

- orderBy() + where() — ВСЕГДА нужен индекс
- orderBy() по разным полям — нужен индекс
- where() с неравенствами + orderBy() — нужен индекс

Как создать индекс?

Способ 1: По ссылке из ошибки (самый простой)

- Запустите запрос в коде.
- Получите ошибку в консоли браузера:

FirebaseError: The query requires an index.
You can create it here: https://console.firebase.google.com/...

- Перейдите по ссылке — она откроет Firebase Console
- Нажмите "Create index" — Firebase всё сделает за вас

Способ 2: Вручную в Firebase Console

Откройте Firebase Console

Выберите ваш проект

Слева: Firestore Database → Indexes

Нажмите "Create index"

Заполните поля:

- Collection ID: books (ваша коллекция)
- Fields to index:
- Field 1: year (тип: Ascending)
- Field 2: title (тип: Ascending)
- Query scope: Collection

Нажмите "Create" (создание займет 1-5 минут)

Разделение прав между обычными пользователями и администратором

3 основных подхода:

1. Custom Claims (самый правильный способ)

Что это?

Firebase позволяет добавлять "кастомные утверждения" (custom claims) к токену пользователя.

Это специальные поля, которые хранятся в токене JWT и проверяются на сервере.

Как работает:

В токене пользователя появляется:

```
{
  uid: "user123",
  email: "admin@site.com",
  // ↓ Custom claims ↓
  role: "admin",
  permissions: ["create", "update", "delete"]
}
```

Плюсы:

- Быстро (проверка на клиенте)
- Безопасно (можно проверить в правилах Firestore)
- Гибко (любые роли и права)

Минусы:

- Нужен Admin SDK (Node.js/Cloud Functions)
- Нельзя установить с фронтенда

2. Роль в Firestore
Как работает:
Храним роль пользователя в документе Firestore:

Коллекция users/{userId}:

```
{
  email: "admin@site.com",
  role: "admin", // или "user"
  createdAt: "20246-01-15"
}
```

Плюсы:

- Просто реализовать
- Можно делать с фронтенда
- Подходит для лабораторной

Минусы:

- Менее безопасно (пользователь может изменить свои данные)
- Нужно дополнительно проверять

3. Проверка по email (самый простейший)
Как работает:
Просто проверяем email пользователя:

```
const user = firebase.auth().currentUser;
if (user.email === "admin@site.com") {
  // Показать админ-панель
}
```

Плюсы:
- Супер просто
- 5 минут на реализацию

Минусы:
- Небезопасно (email можно подделать)
- Негибко (только один админ)

Выводы по теме:

Операции CRUD:

- Create → .add(), .set()
- Read → .get(), .onSnapshot()
- Update → .update()
- Delete → .delete()

Сначала ссылка, потом данные:

```
// 1. Ссылка (где данные?)
const ref = db.collection('books').doc('id')
// 2. Данные (что там?)
const data = await ref.get().data()
```

Два типа чтения:

- Разовое → .get() (один запрос)
- Постоянное → .onSnapshot() (автообновление)

Контрольные вопросы:

- Что такое CRUD?
- Как создать документ с авто-ID?
- Как создать документ со своим ID?
- Как получить ВСЕ документы коллекции?
- Как получить ОДИН документ?
- Чем .get() отличается от .onSnapshot()?
- Как обновить документ?
- Как удалить документ?
- Что такое "ссылка" (Reference)?
- Как проверить, существует ли документ?
- Как фильтровать данные?
- Как сортировать данные?
- Что возвращает .get()?
- Как преобразовать snapshot в массив?
- Как слушать изменения документа?

хекслет колледж

@HEXLY.KZ