Тема 11. Правила безопасности и аутентификация в Firebase.

хекслет колледж

Цель занятия:

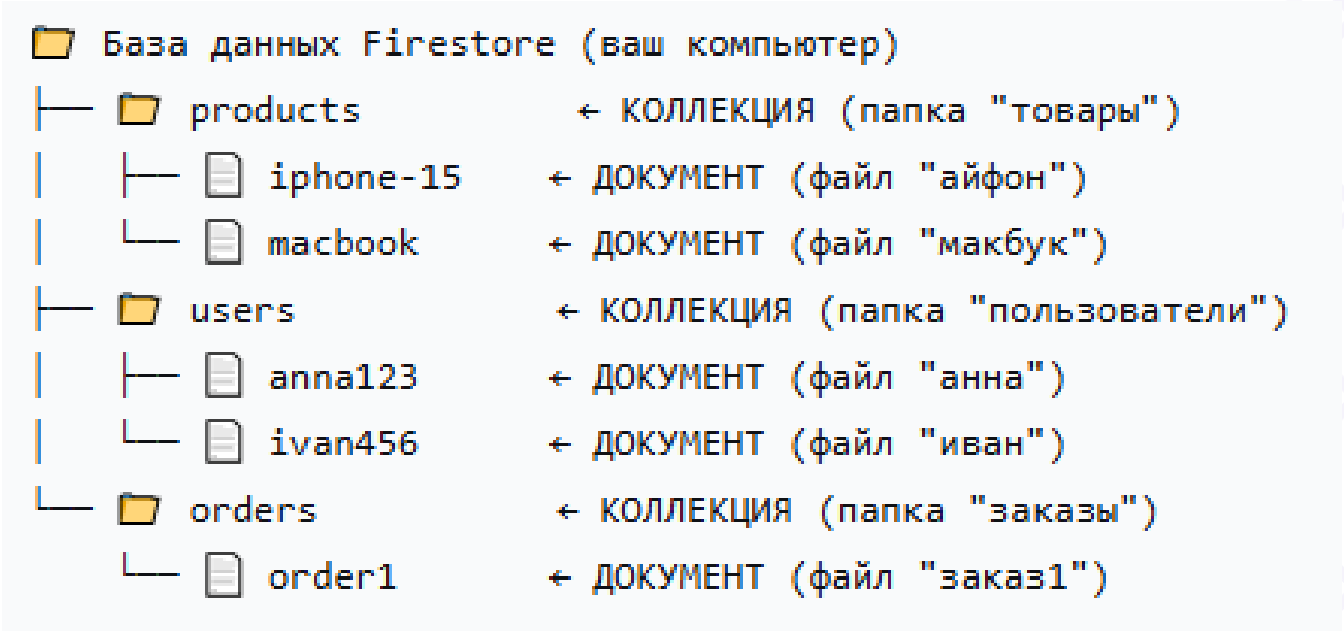
Научить студентов защищать данные в Firestore через правила безопасности и реализовывать систему регистрации/входа пользователей.

Учебные вопросы:

1. Правила безопасности Firestore
2. Firebase Authentication

1. Правила безопасности Firestore

Что такое правила безопасности?
Firestore устроен как папки на компьютере:



Правила безопасности — это разрешения для каждой папки и файла:

- Кто может открыть эту папку?
- Кто может прочитать этот файл?
- Кто может изменить этот файл?

Термины в правилах:

match /products/{productId} — "Для коллекции 'products'..."

Разбираем:

- products — имя коллекции
- {productId} — переменная, означает "любой документ в этой коллекции"
- productId — это ID документа (например: "iphone-15", "macbook")

allow read, write — "Разрешить читать и писать"

allow read: if condition; // Читать документы

allow write: if condition; // Создавать/изменять/удалять документы

if condition — это "УСЛОВИЕ"

Пример. "Всегда ДА":

allow read: if true;
Перевод: "Разрешить читать всегда, без условий"

Пример. "Всегда НЕТ":

allow read: if false;
Перевод: "Никогда не разрешать читать"

if request.auth != null — "Если пользователь вошёл"

- request.auth — информация о пользователе
- null — "ничего" (пользователь не вошёл)
- != null — "не равно ничего" (пользователь вошёл)

Примеры правил для интернет-магазина:

Пример 1: Коллекция "products" (товары)

```
// Для коллекции "products" и любого документа внутри
match /products/{productId} {
  // Читать документы могут ВСЕ
  allow read: if true;

  // Писать (добавлять/изменять/удалять) могут ТОЛЬКО вошедшие пользователи
  allow write: if request.auth != null;
}
```

Что это значит:

- Любой человек может смотреть товары ✅
- Добавить новый товар может только тот, кто вошёл в аккаунт ✅
- Гость (не вошедший) не может изменить товары ❌

Пример 2: Коллекция "users" (пользователи)

```
// Для коллекции "users" и любого документа внутри
match /users/{userId} {
  // Читать и писать может ТОЛЬКО владелец этого документа
  allow read, write: if request.auth.uid == userId;
}
```

Что это значит:

- В коллекции users есть документы: "anna123", "ivan456", "petr789"
- У Анны (документ "anna123") есть её данные
- Анна может читать и менять СВОЙ документ "anna123" ✅
- Анна НЕ может читать документ "ivan456" (данные Ивана) ❌
- Иван НЕ может менять документ "anna123" (данные Анны) ❌

Пример 3: Коллекция "orders" (заказы)

```
// Для коллекции "orders" и любого документа внутри
match /orders/{orderId} {
  // Читать документ могут: 1. Пользователь, который создал этот заказ. 2. Администратор
  allow read: if
    request.auth != null &&
    (
      resource.data.userId == request.auth.uid ||
      request.auth.token.isAdmin == true
    );
  // Создавать документы могут ТОЛЬКО вошедшие пользователи
  allow create: if request.auth != null;
}
```

Что это значит:

- В коллекции orders есть документы с заказами
- Каждый документ содержит поле userId (чей заказ)
- Пользователь видит только СВОИ заказы ✅
- Администратор видит ВСЕ заказы ✅
- Гость не видит НИКАКИХ заказов ❌
- Любой вошедший может создать новый документ-заказ ✅

Специальные переменные:

1. request — "Запрос пользователя"

// Что пользователь хочет сделать?
request.operation // "read" или "write"
request.time // Когда пытается
request.auth // Кто пытается

Пример:
// "Разрешить только до 15 марта"
allow write: if request.time < timestamp.date(2026, 3, 15);

request.auth — "Паспорт пользователя"
// Информация о вошедшем пользователе
request.auth.uid // Уникальный ID пользователя
request.auth.token.role // Роль (admin, user, moderator)
request.auth != null // Вошёл ли вообще?

Примеры:
// "Только вошедшие пользователи"
allow read: if request.auth != null;
// "Только пользователь anna123"
allow write: if request.auth.uid == "anna123";
// "Только администраторы"
allow delete: if request.auth.token.role == "admin";

2. resource — "Существующий документ"
// Данные УЖЕ лежащие в базе
resource.data // Все поля документа
resource.data.ownerId // Конкретное поле

Пример:
// Документ в базе:
{
"title": "iPhone 15",
"price": 999,
"ownerId": "anna123" ← **resource.data.ownerId**
}

// Правило:
allow read: if request.auth.uid == resource.data.ownerId;
// "Анна может читать, если она владелец"

Опасные правила (НИКОГДА так не делайте):

ОПАСНО: Разрешить всё для всех документов
Для коллекции products и ЛЮБОЙ ВЛОЖЕННОСТИ

```
match /products/{document=**} {
  allow read, write: if true;  // ⚠ ОПАСНО!
}
```

Что значит {document=**}:

- * — один уровень (только документы в коллекции)
- ** — любая вложенность (документы + подколлекции + их документы)

Что случится: Любой человек в интернете сможет:

- Удалить все документы в коллекции products
- Изменить данные в любом документе
- Добавить миллион фейковых документов

Правильный подход к правилам:

Шаг 1: Запретить всё по умолчанию.
Для любой коллекции и любого документа

```
match /{document=**} {
  allow read, write: if false;  // ВСЁ ЗАПРЕЩЕНО
}
```

Шаг 2: Постепенно разрешать нужное
1. Разрешить всем читать товары

```
match /products/{productId} {
  allow read: if true;
  allow write: if false;
}
```

2. Разрешить пользователям читать свои данные

```
match /users/{userId} {
  allow read, write: if request.auth.uid == userId;
}
```

3. Разрешить создавать заказы

```
match /orders/{orderId} {
  allow create: if request.auth != null;
  allow read: if request.auth.uid ==
resource.data.userId;
}
```

Схема работы правил:

1. Пользователь пытается прочитать документ
   ↓
2. Firebase смотрит: в какой коллекции этот документ?
   ↓
3. Firebase находит правила для этой коллекции
   ↓
4. Проверяет условие после `if`
   ↓
5. Если условие true → разрешает
   Если условие false → блокирует

Пример правильных правил для коллекции products

```
match /products/{productId} {
  // 1. Все видят товары
  allow read: if true;

  // 2. Админы управляют товарами
  allow write: if
    request.auth != null &&
    request.auth.token.role == "admin";

  // 3. Проверка данных при создании
  allow create: if
    request.auth != null &&
    request.auth.token.role == "admin" &&
    request.resource.data.price > 0 &&
    request.resource.data.name.size() >= 3 &&
    request.resource.data.category in ["electronics", "clothing", "books"];
}
```

Выводы:

Правила нужны чтобы защитить данные от:
❌ Чужих глаз (конфиденциальность)
❌ Вандализма (удаление/изменение)
❌ Спама (фейковые данные)

3 главных принципа:
- Запрещено по умолчанию — начинай с **if false**
- Разные правила для разных коллекций — товары, пользователи, заказы
- Проверяй кто перед тобой — гость, пользователь, админ

✅ Как писать правила:

```
match /коллекция/{документ} {
  allow read: if условие;    // Кто может смотреть
  allow write: if условие;   // Кто может изменять
}
```

🚫 Никогда не делай:
- allow read, write: if true; // ⚠ Опасно!
- Открытая база = сломают за 5 минут

2. Firebase Authentication

Что такое Firebase Authentication?

Это сервис, который:

- Регистрирует пользователей
- Проверяет логины и пароли
- Помнит, кто вошёл в приложение
- Работает бесплатно для 10,000 пользователей

Настройка в Console

Шаг 1: Включите аутентификацию

Firebase Console → Ваш проект → Build →
Authentication
↓
[Get started]

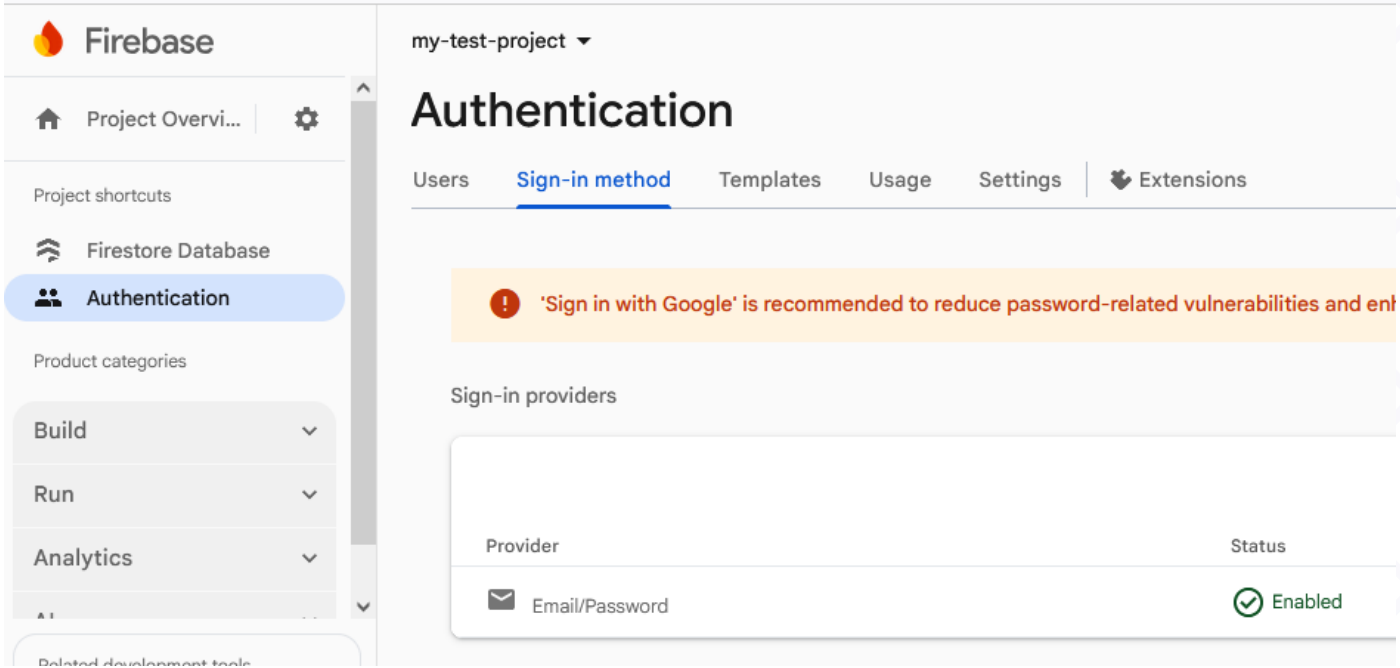
Шаг 2: Выберите способы входа

Sign-in method → [Add new provider]
Основные методы:

- Email/Password — по почте и паролю (самый простой)
- Google — вход через аккаунт Google
- Phone — по номеру телефона
- GitHub — для разработчиков

Для начала включите:

- Email/Password → Включить → Сохранить
- Готово! Аутентификация настроена.



Основные методы Firebase Authentication

Настройка и инициализация:

```
<!-- Подключаем Firebase SDK -->
<script src="https://www.gstatic.com/firebasejs/12.8.0/firebase-auth-compat.js"></script>

// Инициализация
  const app = firebase.initializeApp(firebaseConfig);
  const auth = firebase.auth(); // ← Главный объект для работы
```

auth — это ваш "ключ" ко всем методам аутентификации. Через него вы регистрируете, входите, выходите.

Основные методы (для Email/Password)

Регистрация пользователя.

createUserWithEmailAndPassword(email, password)

Что делает:

- Создает нового пользователя в системе Firebase Authentication.

Что принимает:

- auth, // объект аутентификации (получаем через firebase.auth())
- "user@example.com", // строка, email пользователя
- "password123" // строка, пароль (минимум 6 символов)

Что возвращает:
Promise (обещание), который при успехе возвращает объект UserCredential:

```
{
  user: {
    uid: "abc123def456...",         // Уникальный ID пользователя
    email: "user@example.com",      // Email пользователя
    emailVerified: false,           // Подтвержден ли email
    displayName: null,              // Имя пользователя (пока null)
    photoURL: null,                 // Фото пользователя (пока null)
    metadata: {
      creationTime: "2024-01-15T10:30:00Z",  // Когда создан
      lastSignInTime: "2024-01-15T10:30:00Z" // Когда последний раз входил
    }
    // ... другие свойства
  },
  providerId: "password",           // Способ аутентификации
  operationType: "signIn"           // Тип операции
}
```

Пример использования:

```
// 1. Получаем объект аутентификации
const auth = firebase.auth();

// 2. Вызываем метод
auth.createUserWithEmailAndPassword("ivan@mail.ru", "mypassword")
  .then((userCredential) => {
    // 3. Работаем с результатом
    const user = userCredential.user;
    console.log("Создан пользователь с ID:", user.uid);
    console.log("Email:", user.email);
  })
  .catch((error) => {
    // 4. Обрабатываем ошибки
    console.error("Ошибка:", error.message);
  });
```

Что происходит внутри:

- Firebase проверяет, нет ли уже пользователя с таким email
- Хеширует пароль (превращает в нечитаемую строку)
- Сохраняет данные в своей защищенной базе
- Автоматически авторизует пользователя
- Возвращает информацию о созданном пользователе

Вход пользователя

signInWithEmailAndPassword(email, password)

Что делает:

- Авторизует существующего пользователя (вход в систему).

Что принимает:
То же самое, что и createUserWithEmailAndPassword:

- объект аутентификации
- email (строка)
- пароль (строка)

Что возвращает:

- Тот же UserCredential объект, что и при регистрации.

Пример использования:

```
const auth = firebase.auth();

auth.signInWithEmailAndPassword("ivan@mail.ru", "mypassword")
  .then((userCredential) => {
    const user = userCredential.user;
    // После успешного входа:
    console.log("Добро пожаловать,", user.email);
    console.log("Ваш ID:", user.uid);
   })
  .catch((error) => {
    if (error.code === "auth/user-not-found") {
      alert("Пользователь не найден!");
    } else if (error.code === "auth/wrong-password") {
      alert("Неверный пароль!");
    }
  });
```

Что происходит внутри:

- Firebase находит пользователя по email
- Сравнивает хеши паролей (не сами пароли!)
- Если совпадает — создает сессию
- Возвращает данные пользователя
- Запоминает пользователя в браузере

Важно: После успешного входа пользователь остается авторизованным даже если:

- Закрыть вкладку браузера
- Перезагрузить страницу
- Выключить и включить компьютер
- Пока пользователь явно не выйдет (signOut) или не очистит куки.

Выход пользователя

signOut()
Что делает:

- Завершает текущую сессию пользователя (выход из системы).

Что принимает:

- Ничего не принимает, или можно передать объект аутентификации:

//Пример:

```
const auth = firebase.auth();
auth.signOut();
```

Что возвращает:

- Promise, который завершается успешно когда:
  - Сессия завершена
  - Все связанные токены удалены

Пример использования:

```
const auth = firebase.auth();

function logout() {
  auth.signOut()
    .then(() => {
      console.log("Выход выполнен успешно");

      // Обновляем интерфейс
      document.getElementById("user-menu").style.display = "none";
      document.getElementById("login-form").style.display = "block";
    })
    .catch((error) => {
      console.error("Ошибка при выходе:", error);
    });
}

// Кнопка выхода
document.getElementById("logout-btn").addEventListener("click", logout);
```

Что происходит внутри:

- Удаляет токены аутентификации из localStorage/sessionStorage
- Сообщает Firebase серверу о завершении сессии
- onAuthStateChanged получает null (нет пользователя)

Отслеживание состояния пользователя

onAuthStateChanged(callback)
Что делает:

- Слушает (подписывается) на изменения состояния аутентификации.

Что принимает:

- Функцию-обработчик (callback), которая принимает один параметр:

```
onAuthStateChanged((user) => {
  // user - объект пользователя или null
});
```

Что возвращает:
Функцию для отписки (unsubscribe):

```
const unsubscribe =
auth.onAuthStateChanged((user) => {
  // ...
});

// Когда нужно перестать слушать:
unsubscribe();
```

Пример использования:

```
const auth = firebase.auth();

// Подписываемся на изменения
const unsubscribe = auth.onAuthStateChanged((user) => {
  console.log("Состояние изменилось!");

  if (user) {
    // Есть пользователь (вошел или уже был авторизован)
    console.log("Пользователь:", user.email);
    showUserInterface(user);
  } else {
    // Нет пользователя (вышел или никогда не входил)
    console.log("Пользователь не авторизован");
    showLoginInterface();
  }
});
```

Эта функция вызывается Firebase автоматически:

- При загрузке страницы
- После успешного входа (signInWithEmailAndPassword)
- После выхода (signOut)
- При истечении сессии
- При ошибках аутентификации

Зачем нужно отслеживать состояние авторизации:

- Чтобы не просить логин/пароль каждый раз
- Чтобы данные не терялись при обновлении страницы
- Чтобы показывать разные интерфейсы

Работа с профилем пользователя

Получение текущего пользователя:

auth.currentUser

Что делает:

- Возвращает текущего авторизованного пользователя.

Что принимает:

- Ничего не принимает.

Что возвращает:

- Объект пользователя, если пользователь авторизован

null, если пользователь не авторизован

Пример использования:

```
const auth = firebase.auth();

// Проверка в любом месте кода
function checkIfLoggedIn() {
  const user = auth.currentUser;

  if (user) {
    console.log("Пользователь авторизован:", user.email);
    return true;
  } else {
    console.log("Пользователь не авторизован");
    return false;
  }
}
```

Использование перед важными действиями

```
function addToCart(productId) {
  if (!auth.currentUser) {
    alert("Войдите, чтобы добавлять товары в корзину!");
    return;
  }

  // Пользователь есть, можно добавлять
  const userId = auth.currentUser.uid;
  // ... добавляем товар
}
```

Важные моменты:

- currentUser обновляется автоматически
- Может быть null сразу после загрузки страницы (пока onAuthStateChanged не сработал)
- Используйте вместе с onAuthStateChanged для надежности

Обновление профиля

user.updateProfile(profileData)

Что делает:

- Обновляет базовую информацию о пользователе (имя и фото).

Что принимает:

- Объект с полями для обновления:

```
{
  displayName: "Новое имя",  // строка (не обязательно)
  photoURL: "https://..."    // строка, URL фото (не обязательно)
}
```

Что возвращает:

- Promise, который завершается когда профиль обновлен.

Пример использования:

```
const auth = firebase.auth();

function updateUserProfile() {
  // Получаем текущего пользователя
  const user = auth.currentUser;

  if (!user) {
    alert("Войдите, чтобы обновить профиль!");
    return;
  }

  // Готовим данные для обновления
  const newName = document.getElementById("name-input").value;
  const newPhoto = document.getElementById("photo-input").value;

  const profileUpdates = {
    displayName: newName || user.displayName, // если пусто - оставить старое
    photoURL: newPhoto || user.photoURL
  };
```

```
// Вызываем метод
 user.updateProfile(profileUpdates)
   .then(() => {
     console.log("Профиль успешно обновлен!");

     // Обновляем интерфейс
     document.getElementById("user-name").textContent = user.displayName;

     // Проверяем результат
     console.log("Новое имя:", user.displayName);
     console.log("Новое фото:", user.photoURL);
   })
   .catch((error) => {
     console.error("Ошибка обновления:", error);
   });
}
```

Что НЕЛЬЗЯ обновить через updateProfile:

- Email (используйте user.updateEmail())
- Пароль (используйте user.updatePassword())
- UID (никогда не меняется)
- Email verification status

Что происходит при обновлении:

- Firebase обновляет данные в своей базе
- НЕ обновляет auth.currentUser сразу (только после перезагрузки страницы)
- Что бы увидеть изменения сразу используйте user.reload()

Сводная таблица методов

Метод	Принимает	Возвращает	Когда использовать
createUserWithEmailAndPassword()	email, password	UserCredential	При регистрации нового пользователя
signInWithEmailAndPassword()	email, password	UserCredential	При входе существующего пользователя
signOut()	ничего	Promise	При выходе из системы
onAuthStateChanged()	callback-функцию	функция отписки	При загрузке страницы, для отслеживания состояния
auth.currentUser	ничего	User или null	В любой момент для проверки авторизации
user.updateProfile()	объект с displayName/photoURL	Promise	При изменении имени или фото пользователя

Выводы по теме:

Firebase Authentication. ДЛЯ ЧЕГО:

- Регистрация новых пользователей
- Вход в систему
- Выход из системы
- Запоминание пользователя

ПОДДЕРЖИВАЕТ:

- Email/пароль (самое простое)
- Google, Facebook, GitHub
- Телефон (SMS)
- Анонимный вход

КАК РАБОТАЕТ:

- Auth → хранит email и пароль
- Firestore → хранит имя, адрес, профиль
- Один ID на обоих сервисах

БЕЗОПАСНОСТЬ Firebase:

- Сначала всё запретить!
- Потом разрешить нужное
- Каждый пользователь = только свои данные

Контрольные вопросы:

- Что делает Firebase Authentication?
- Как зарегистрировать пользователя?
- Как войти в систему?
- Как выйти из системы?
- Как проверить, вошёл ли пользователь?
- Где хранятся логин/пароль?
- Где хранить имя и адрес пользователя?
- Какие правила безопасности самые важные?
- Что будет при обновлении страницы?

Документация Firebase:
https://firebase.google.com/docs/firestore/security/get-started?hl=ru

Правила в песочнице:
https://firebase.google.com/docs/rules/simulator

Примеры правил:
https://firebase.google.com/docs/firestore/security/rules-recipes

хекслет колледж

@HEXLY.KZ