Teach Me Skills

Занятие 24.
JWT Auth

1

Основные понятия

Идентификация

Это процесс определения или подтверждения уникальной личности субъекта или сущности. При идентификации выясняется, кто или что что-то является. Обычно это включает предоставление и проверку идентификационной информации, такой как имя пользователя, номер учетной записи или физические характеристики (отпечатки пальцев, сетчатка глаза и т. д.). Цель идентификации - установить уникальность личности или объекта.

логин-пароль
email-пароль
username-пароль

Аутентификация

Это процесс проверки подлинности или подтверждения того, что идентифицированный субъект или объект является тем, за кого или что себя выдаёт. При аутентификации предоставленная идентификационная информация проверяется на соответствие заранее установленным критериям или учетным данным, таким как пароль, ключ безопасности, сертификат или биометрические данные.

Цель аутентификации - убедиться, что пользователь или объект действительно является тем, кто или что он утверждает.

Авторизация

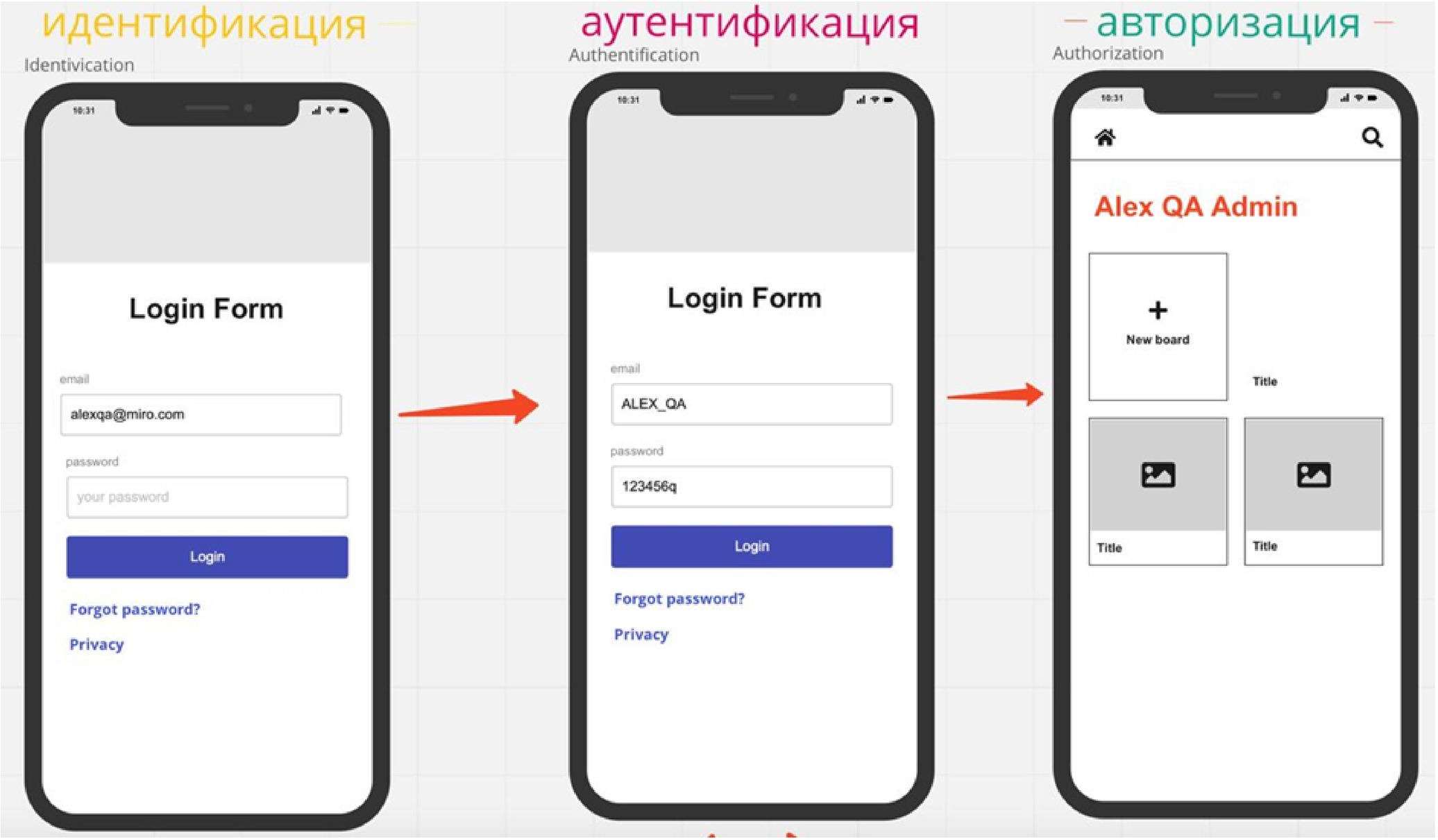
Это процесс предоставления разрешений и определения уровня доступа после успешной идентификации и аутентификации. При авторизации система определяет, какие ресурсы, данные или функции пользователь или объект может использовать или выполнять.

Например, в системе с несколькими уровнями доступа разные пользователи могут иметь разные уровни привилегий.

Авторизация контролирует доступ и определяет, какие действия могут быть выполнены после успешной аутентификации.

Аутентификация нужна, чтобы проверить право доступа к данным, авторизация — это когда вы получаете доступ

Как это выглядит



2

Аутентификация в Django

Session-based authentication - по дефолту

Аутентификация на основе сеансов (Session-based authentication):

Django поддерживает аутентификацию на основе сеансов, используя куки (cookies) для идентификации и аутентификации пользователя. При успешной аутентификации Django создает уникальный идентификатор сеанса, который отправляется пользователю в виде куки. Дальнейшие запросы от пользователя включают этот идентификатор сеанса для аутентификации

django.contrib.auth
@login_required

Authentication backend

Парольная аутентификация (Authentication backends):

Django предоставляет встроенную парольную аутентификацию, которая основана на сравнении хэшей паролей. Пользователи могут создавать учетные записи с уникальными именами пользователей и паролями. Django также предоставляет возможность хранить пароли в безопасной форме с использованием алгоритмов хэширования, таких как bcrypt или Argon2

Token-based authentication

Аутентификация на основе токенов (Token-based authentication):

Django предоставляет поддержку аутентификации на основе токенов. Пользователь получает токен после успешной аутентификации, и этот токен используется для последующей аутентификации запросов. Токены могут быть переданы в заголовке запроса или в куках. Django также предоставляет представления (views) и декораторы, которые облегчают работу с токенами.

Social authentication

Социальная аутентификация (Social authentication):

Django также поддерживает аутентификацию через сторонние провайдеры социальных сетей, таких как Facebook, Google, Twitter и других. С помощью пакетов, таких как django-allauth или python-social-auth, можно настроить социальную аутентификацию в Django и позволить пользователям аутентифицироваться через свои учетные записи в социальных сетях.

3

JWT

JWT

JWT (JSON Web Token) - это открытый стандарт, который используется для безопасной передачи информации в формате JSON между двумя сторонами. Он обычно применяется для аутентификации и авторизации пользователей в веб-приложениях и API

JWT состоит из трех основных компонентов:

- заголовка (header)
- полезной нагрузки (payload)
- подписи (signature)

Header, payload, signature

- Информация о типе токена
  {"alg": "HS256","typ": "JWT"}

- Полезная нагрузка (Payload) содержит утверждения (claims), которые представляют собой информацию, которую вы хотите передать, например, идентификатор пользователя, срок действия токена и другие пользовательские данные. Пример:
  {"sub": "1234567890","name": "John Doe","exp": 1624242424}

- Подпись (Signature) представляет собой хэш (hash), который используется для проверки целостности токена. Она создается путем кодирования заголовка и полезной нагрузки, а затем подписывается секретным ключом: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

Как это работает

Когда пользователь успешно аутентифицируется, сервер генерирует JWT и отправляет его обратно клиенту. Клиент сохраняет токен, обычно в локальном хранилище (например, в cookies или в localStorage), и отправляет его с каждым последующим запросом на сервер в заголовке **Authorization**

PyJWT или **djangorestframework-simplejwt**

Зачем делать refresh токена

1. **Безопасность**: Обновление токена позволяет сократить время действия каждого отдельного токена. Если токен украден или скомпрометирован, его действие будет ограничено и оно будет ограничено лишь до следующего обновления. Это помогает уменьшить окно атаки и повышает безопасность вашей системы.
2. **Снижение нагрузки на сервер:** Если каждый раз переполучать токен, это может оказывать негативное влияние на производительность сервера. Refresh token позволяет продлить его срок действия без необходимости выполнять полноценную аутентификацию снова.
3. **Управление сессиями и разрешениями:** Обновление токена может быть полезным для управления сессиями и правами пользователей. Например, при обновлении токена можно обновить разрешения пользователя или добавить дополнительные данные, связанные с его сеансом

4

Middleware

Middleware

Middleware в Django - это компонент, который позволяет обрабатывать запросы и ответы в промежуточном слое между веб-сервером и представлениями (views) в Django приложении. Он выполняет функции обработки, изменения или проверки запросов и ответов перед тем, как они достигнут представления или после их обработки представлениями.

1. Идет по цепочке в порядке, в котором они указаны в MIDDLEWARE
2. Каждый Middleware имеет доступ к запросу и может выполнить операции над ним. Например, Middleware может проверить аутентификацию пользователя, применить локализацию, добавить или изменить заголовки запроса и многое другое
3. После обработки запроса вьюхой, Middleware снова начинает обрабатывать ответ

Middleware - примеры

- Аутентификация и авторизация пользователей.
- Обработка исключений и ошибок.
- Добавление или изменение заголовков запроса и ответа.
- Кеширование запросов.
- Управление сессиями и cookies.
- Логирование запросов и ответов.
- Обработка CORS (Cross-Origin Resource Sharing).
- И многое другое.

SecurityMiddleware

- Защита от кликджекинга (Clickjacking Protection) - добавляет X-Frame-Options, чтобы злоумышленик не могу внедрить в сайт iframe и пытаться манипулировать действиями пользователя
- ащита от атаки типа MIME (MIME Type Sniffing Protection) - X-Content-Type-Options: nosniff, не предугадывает тип контента
- CSRF
- Защита от клиентских эксплойтов (XSS Protection) - добавляет заголовок X-XSS-Protection в ответы, чтобы браузеры могли активировать механизмы предотвращения атак на межсайтовый скриптинг (XSS). Это помогает защитить ваше приложение от вредоносных скриптов, внедряемых злоумышленниками

SessionMiddleware

- SessionMiddleware проверяет наличие сессионной информации в запросе. Для этого оно анализирует значение сессионной куки, которая обычно называется session_id иначе создает новую
- Если сессионная кука с действительным значением найдена, SessionMiddleware извлекает данные сессии из хранилища, указанного в настройках Django (например, в базе данных)
- Полученные данные сессии доступны во время обработки запроса представлением (view) и могут быть использованы для хранения и извлечения информации, связанной с пользователем
- Если данные сессии изменены во время обработки запроса представлением, SessionMiddleware обновляет хранилище сессии, чтобы сохранить изменения.
- Устанавливает сессионную куку в ответе

CsrfViewMiddleware

предоставляет защиту от атаки межсайтовой подделки запроса (CSRF)

- Генерируем CSRF токен и вставляем при помощи csrf_token
- HTTP 403 Forbidden если токен не пришел с запросом POST

```
from django.views.decorators.csrf import ensure_csrf_cookie
from django.http import JsonResponse

@ensure_csrf_cookie
def get_csrf_token(request):
    return JsonResponse({'csrfToken': request.COOKIES['csrftoken']})
```

Спасибо за внимание!