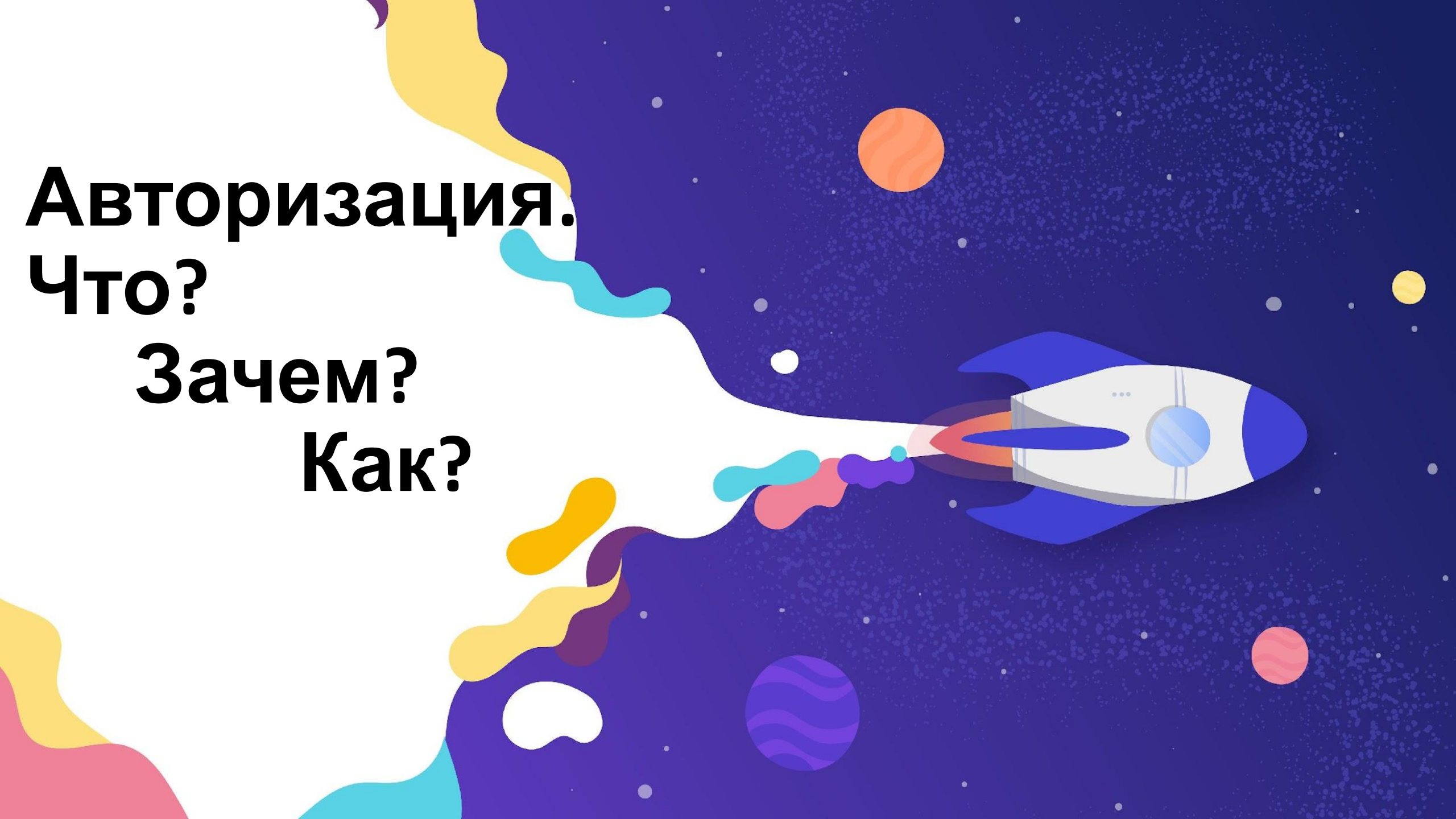
Авторизация.
Что?
Зачем?
Как?



Авторизация. Общая информация

Один из основных вопросов, которые возникают при работе с клиент-серверными приложениями, - это вопрос авторизации.

Авторизация - это процесс проверки легитимности пользователя или приложения, которое запрашивает доступ к защищенным ресурсам.

В контексте клиент-серверных приложений авторизация используется для определения, имеет ли пользователь или приложение право на доступ к определенным ресурсам на сервере.

Механизм авторизации для клиент-серверных приложений подразумевает под собой использование т.н. сессий или, проще говоря, создание сеанса для пользователя, при котором нет необходимости при посещении каждой страницы или при каждом запросе какого-либо ресурса авторизовываться.

Авторизация. Способы авторизации

Существуют различные методы авторизации в клиент-серверных приложениях.

1. **Basic Auth** (авторизация на уровне веб-сервера)
2. **Username \ password** (авторизация на основе проверки приложением введенных имени пользователя и пароля)
3. **Email или phone \ one-time-password** (авторизация на основе одноразовых паролей)
4. **Social networks auth** (авторизация через социальные сети или другие третье сторонние сервисы)
5. **OAuth** (авторизация по протоколу OAuth)

Авторизация. Basic Auth

Базовая авторизация является одним из простейших методов аутентификации пользователей, который основывается на передаче имени пользователя и пароля в заголовке HTTP-запроса.

Данные передаются в открытом виде, поэтому этот метод не рекомендуется использовать для передачи конфиденциальных данных.

Однако, базовая авторизация может быть полезна для простой защиты доступа к ресурсам веб-сервера.

Обратите внимание, что для использования базовой авторизации, необходимо также убедиться, что соединение между клиентом и сервером защищено протоколом HTTPS.

Использование базовой авторизации без HTTPS делает передачу имени пользователя и пароля уязвимой к атакам перехвата данных.

Авторизация. Basic Auth. Nginx

Механизм подобной авторизации реализуется с помощью веб-сервера (например, Nginx или Apache).

Для реализации базовой авторизации на веб сервере Nginx, необходимо добавить следующий блок в конфигурационный файл сервера:

```
location /protected {
    auth_basic "Restricted Access";
    auth_basic_user_file /etc/nginx/.htpasswd;
    ...
}
```

В этом примере, мы указываем, что для доступа к ресурсам в папке "protected" необходимо пройти базовую аутентификацию.

Файл "/etc/nginx/.htpasswd" содержит пары имя пользователя/хэш пароля, которые будут использоваться для проверки доступа.

Для создания файла .htpasswd можно использовать утилиту htpasswd.

Авторизация. Basic Auth. Apache2

Для реализации базовой авторизации на веб сервере Apache2, необходимо создать файл .htaccess в папке, которую мы хотим защитить, и добавить следующие строки:

```
AuthType Basic
AuthName "Restricted Area"
AuthUserFile /path/to/.htpasswd
Require valid-user
```

В этом примере, мы указываем, что для доступа к ресурсам в этой папке необходима базовая аутентификация. Файл "/path/to/.htpasswd" содержит пары имя пользователя/хэш пароля, которые будут использоваться для проверки доступа.

Для создания файла .htpasswd можно использовать утилиту htpasswd.

Важно: при подобной авторизации приложение не знает ничего о пользователе и не поднимает сессию самостоятельно.
Управлением сессиями пользователей занимается веб-сервер.

Авторизация. Username - password

Метод username-password является одним из наиболее распространенных методов аутентификации пользователей в веб-приложениях.

Он основывается на проверке сочетания имени пользователя и пароля, которые пользователь вводит при попытке входа в систему.

Важно понимать, что за проверку корректности введенных данных отвечает само приложение.

Наиболее распространенным способом хранения пользовательских данных является база данных.

Авторизация. Username - password

Простейшим сценарием подобной авторизации является следующий:

1. Пользователь вводит имя пользователя и пароль.
2. Введенные данные передаются на сервер.
3. Сервер проверяет корректность имени пользователя и пароля.
4. Если имя пользователя и пароль являются корректными, сервер создает JWT токен и отправляет его пользователю.
5. Пользователь сохраняет полученный токен в локальном хранилище браузера.
6. При каждом запросе на сервер, пользователь передает JWT токен в заголовке запроса.
7. Сервер проверяет корректность токена и предоставляет доступ к запрашиваемому ресурсу, если токен является действительным.

Авторизация. JWT

JWT токен - это JSON Web Token, который используется для передачи утверждений между сторонами.

Он состоит из трех частей: заголовка, полезной нагрузки и подписи.

Заголовок содержит информацию о типе токена и используемом алгоритме шифрования.

Полезная нагрузка содержит утверждения, которые передаются между сторонами. Подпись используется для проверки целостности токена.

Авторизация. JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Заголовок (Header)

Заголовок состоит из двух частей: типа токена (JWT) и используемого алгоритма шифрования (HS256). В данном случае используется алгоритм HMAC-SHA256.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

Авторизация. JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Полезная нагрузка (Payload)
Полезная нагрузка содержит утверждения, которые передаются между сторонами.
В данном примере она состоит из трех утверждений: идентификатора пользователя (sub), имени пользователя (name) и времени создания токена (iat).

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ
```

Авторизация. JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Подпись (Signature)

Подпись используется для проверки целостности токена.
Она создается на основе заголовка, полезной нагрузки и секретного ключа, который известен только серверу.
В данном примере используется секретный ключ "secret".

```
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Авторизация. One-time-password

Клиент-серверная авторизация с помощью email или телефона и одноразового пароля - это безопасный способ аутентификации пользователей, который часто используется в современных веб-приложениях и мобильных приложениях.

Процесс аутентификации с помощью email или телефона и одноразового пароля начинается с того, что пользователь вводит свой email или номер телефона на странице входа в систему.

Затем система отправляет пользователю на email или телефон одноразовый пароль, который пользователь должен ввести на странице входа, чтобы подтвердить свою личность.

Авторизация. One-time-password

Одноразовый пароль, как правило, действителен только на определенное время (например, 5-10 минут), после чего он становится недействительным. Это обеспечивает дополнительный уровень безопасности, так как злоумышленник не сможет использовать одноразовый пароль, если его украдут или получат несанкционированный доступ к нему после того, как он станет недействительным.

Одним из главных преимуществ такого метода авторизации является удобство для пользователей. Они могут использовать свой email или телефон, который уже зарегистрирован в системе, без необходимости запоминать сложные пароли или создавать новые. Кроме того, такой метод авторизации более безопасен, чем использование обычных паролей, которые могут быть украдены или подобраны злоумышленниками.

Авторизация. One-time-password. Недостатки

Однако, есть и недостатки.

1. Такой метод требует наличия соответствующей инфраструктуры для отправки одноразовых паролей на email или телефон.

2. Стоимость такого метода может быть неподходящей для бизнеса.

3. Если злоумышленник получит доступ к email или телефону пользователя, он может получить доступ к одноразовому паролю и зайти в систему от имени пользователя.

Чтобы уменьшить риски, рекомендуется использовать дополнительные меры безопасности, например, двухфакторную аутентификацию.

Авторизация. 2FA

Двухфакторная аутентификация – это метод обеспечения безопасности, который требует от пользователя предоставления двух форм идентификации для проверки его личности.

Этот метод используется для защиты от несанкционированного доступа к пользовательским аккаунтам, и он становится все более популярным в настоящее время.

Первый фактор аутентификации - это обычно пароль, который пользователь вводит при входе в систему или другие способы авторизации, которые уже были рассмотрены.

Авторизация. 2FA

Второй фактор может быть различным в зависимости от реализации метода двухфакторной аутентификации. Например, это может быть временный код, отправленный на телефон или по электронной почте, скан отпечатка пальца или лица, а также использование аппаратного токена, такого как Yubikey.

Для включения двухфакторной аутентификации пользователь должен пройти процедуру регистрации в системе и настроить свой второй фактор аутентификации.

Как только включена двухфакторная аутентификация, при следующем входе в систему пользователю будет предложено ввести временный код или предоставить скан отпечатка пальца или лица, дополнительно к паролю.

Авторизация. 2FA

Двухфакторная аутентификация повышает безопасность пользователя, поскольку для того, чтобы получить доступ к аккаунту, злоумышленнику нужно не только узнать пароль пользователя, но и обладать доступом к его устройству или другим методам аутентификации.

Это усложняет задачу злоумышленникам, которые могут попытаться получить доступ к пользовательским аккаунтам.

Однако, несмотря на увеличение безопасности, двухфакторная аутентификация может быть неудобной для пользователей, особенно если они используют различные устройства для доступа к системе.

Кроме того, она может привести к дополнительным затратам для предприятий, которые должны внедрить систему двухфакторной аутентификации.

Авторизация. Google 2FA

Google Auth – это сервис, который позволяет добавить двухфакторную аутентификацию в приложение или веб-сайт. Он основан на алгоритме TOTP (Time-Based One-Time Password), который генерирует временные одноразовые пароли.

Двухфакторная аутентификация с помощью Google Auth обеспечивает более высокий уровень безопасности, чем обычный пароль, поскольку даже если злоумышленник получит доступ к паролю пользователя, ему потребуется одноразовый пароль из Google Auth, чтобы получить доступ к системе.

Некоторые приложения и сайты также позволяют настроить дополнительные опции безопасности, такие как проверка наличия подтверждения входа на устройстве пользователя или уведомления о входах в систему с нового устройства.

Авторизация. Google 2FA

Принцип работы двухфакторной аутентификации с помощью Google Auth следующий:

1. Пользователь регистрируется в системе, подключает двухфакторную аутентификацию и скачивает приложение Google Auth на свой мобильный телефон.
2. Приложение генерирует уникальный ключ, который пользователь вводит в систему.
3. Система и Google Auth используют этот ключ для синхронизации времени. Приложение Google Auth начинает генерировать одноразовые пароли, которые имеют ограниченное время жизни.
4. При входе в систему пользователь вводит свой логин и пароль, после чего запрашивается одноразовый пароль из Google Auth.
5. Приложение проверяет, совпадает ли введенный пароль с паролем, который был сгенерирован в Google Auth. Если все верно, пользователь получает доступ к системе.

Авторизация. Google 2FA

Как правило, приложения, которые подключают Google Auth, генерируют ключ по алгоритму, известному из документации.

Этот ключ может быть преобразован в QR - код, который может быть отсканирован с помощью устройства пользователя (например, мобильного телефона).

Еще одной опцией является то, что данный ключ можно ввести вручную. Такой способ обычно применяется, когда подразумевается использование этого способа аутентификации и на других устройствах

Авторизация. Протокол OAuth

OAuth (Open Authorization) - это протокол авторизации, который позволяет пользователям дать разрешение на доступ к их ресурсам на сторонних сайтах без передачи им логина и пароля.

OAuth позволяет пользователям легко и безопасно управлять своими личными данными и предоставлять доступ к ним.

OAuth использует токен доступа для аутентификации пользователей и предоставления доступа к их ресурсам.

Этот токен выдается на определенный период времени и может быть использован только для доступа к определенным ресурсам.

Авторизация. Протокол OAuth

Протокол OAuth состоит из четырех основных ролей:

1. Resource Owner (Владелец ресурса) - это пользователь, который владеет данными, к которым запрашивается доступ.
2. Client (Клиент) - это веб-приложение или сервис, который запрашивает доступ к ресурсам пользователя.
3. Authorization Server (Сервер авторизации) - это сервер, который выдает токены авторизации клиенту после того, как пользователь дал согласие на доступ.
4. Resource Server (Сервер ресурсов) - это сервер, который хранит ресурсы, к которым запрашивается доступ.

Авторизация. Протокол OAuth

Для авторизации с помощью OAuth необходимо выполнить следующие шаги:

1. Клиент запрашивает авторизацию пользователя через редирект (переадресацию) на сервер авторизации.
2. Сервер авторизации запрашивает у пользователя авторизацию и получает согласие на доступ к запрашиваемым ресурсам.
3. Сервер авторизации выдает клиенту токен авторизации.
4. Клиент отправляет запрос к серверу ресурсов, передавая токен авторизации.
5. Сервер ресурсов проверяет токен авторизации и, если он действителен, предоставляет доступ к запрашиваемым ресурсам.

Авторизация. Протокол OAuth

Токен авторизации, полученный в OAuth 2.0, имеет ограниченный срок действия и может быть обновлен с помощью Refresh Token, который также выдается сервером авторизации. Это позволяет клиенту продлить доступ к ресурсам пользователя без необходимости запроса нового токена.

OAuth имеет несколько версий, и каждая версия имеет свои особенности и требования. Например, версия OAuth 2.0 является наиболее распространенной и используется во многих современных веб-приложениях.

Использование OAuth позволяет упростить процесс авторизации и предоставления доступа к ресурсам пользователей. Однако необходимо учитывать, что без должной защиты, OAuth может стать уязвимостью в системе безопасности веб-приложения.

Авторизация. Social Networks

В современных веб-приложениях пользователи часто имеют возможность авторизоваться через социальные сети, такие как Yandex, VK, Google, Github и т.д.

Это позволяет пользователям быстро и удобно зарегистрироваться в приложении без необходимости заполнять большое количество полей регистрации и запоминать еще один логин и пароль.

Клиент-серверная авторизация через социальные сети осуществляется при помощи протокола OAuth, который позволяет приложению запрашивать доступ к ресурсам пользователя на стороне социальной сети.

Авторизация. Social Networks

Процесс авторизации через социальную сеть состоит из нескольких шагов:

1. Пользователь выбирает социальную сеть, через которую он хочет авторизоваться в приложении.
2. Приложение отправляет запрос на авторизацию к социальной сети, в котором указывает свой идентификатор приложения и запрашивает доступ к ресурсам пользователя.
3. Социальная сеть запрашивает у пользователя разрешение на доступ к ресурсам и, если пользователь дает согласие, возвращает приложению токен доступа.
4. Приложение использует токен доступа для получения информации о пользователе (например, имя, электронная почта, профиль) и создает в своей базе данных новую учетную запись или обновляет существующую.
5. Пользователь авторизуется в приложении и получает доступ к его функционалу.

Авторизация. Social Networks

Для реализации авторизации через социальные сети в приложении необходимо зарегистрировать приложение в соответствующей социальной сети и получить идентификатор приложения и секретный ключ. Затем нужно подключить библиотеку для работы с протоколом OAuth и настроить авторизацию в приложении.

Достоинствами авторизации через социальные сети являются удобство и скорость регистрации пользователей в приложении, а также возможность использования информации из профиля пользователя для персонализации функционала приложения.

Однако, следует учитывать, что при авторизации через социальные сети пользователи предоставляют доступ к своим личным данным, что может вызывать определенные опасения в плане безопасности.

Авторизация. Механизм сессий

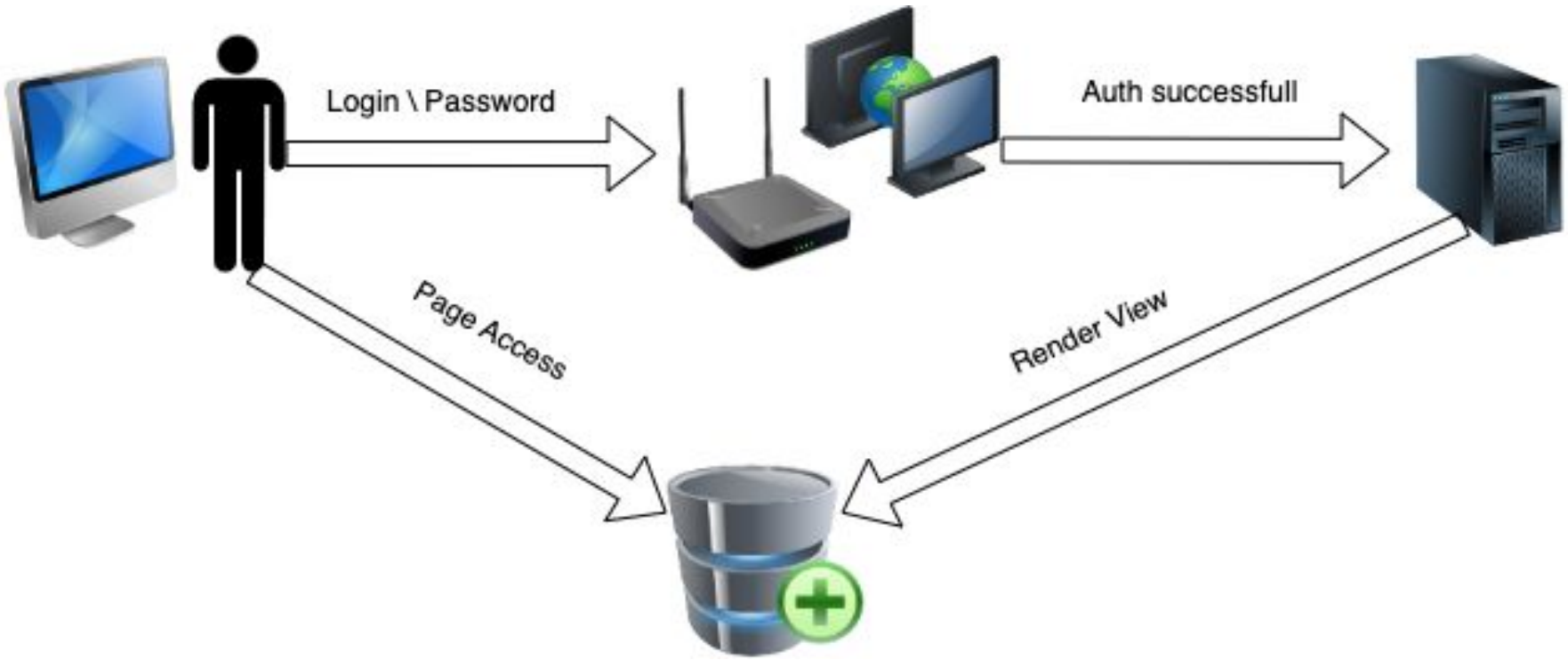
Сессия – это период времени, в течение которого пользователь взаимодействует с веб-приложением.

В этот период времени сервер сохраняет информацию о состоянии пользователя.

Механизм сессий позволяет сохранять данные о состоянии пользователя между запросами и перенаправлениями без необходимости каждый раз авторизовываться.

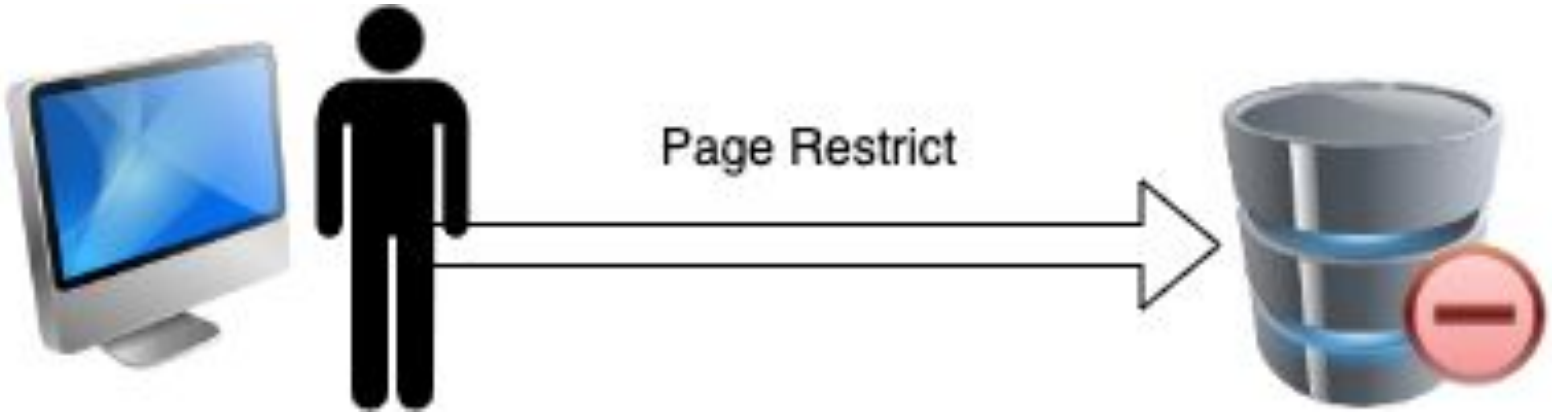
Ниже проиллюстрирован пример использования авторизации без механизма сессий.

Авторизация. Механизм сессий



Как видно, пользователь вводит данные для авторизации, сервер их проверяет на корректность. При успешной проверке сервер дает доступ пользователю на просмотр страницы.

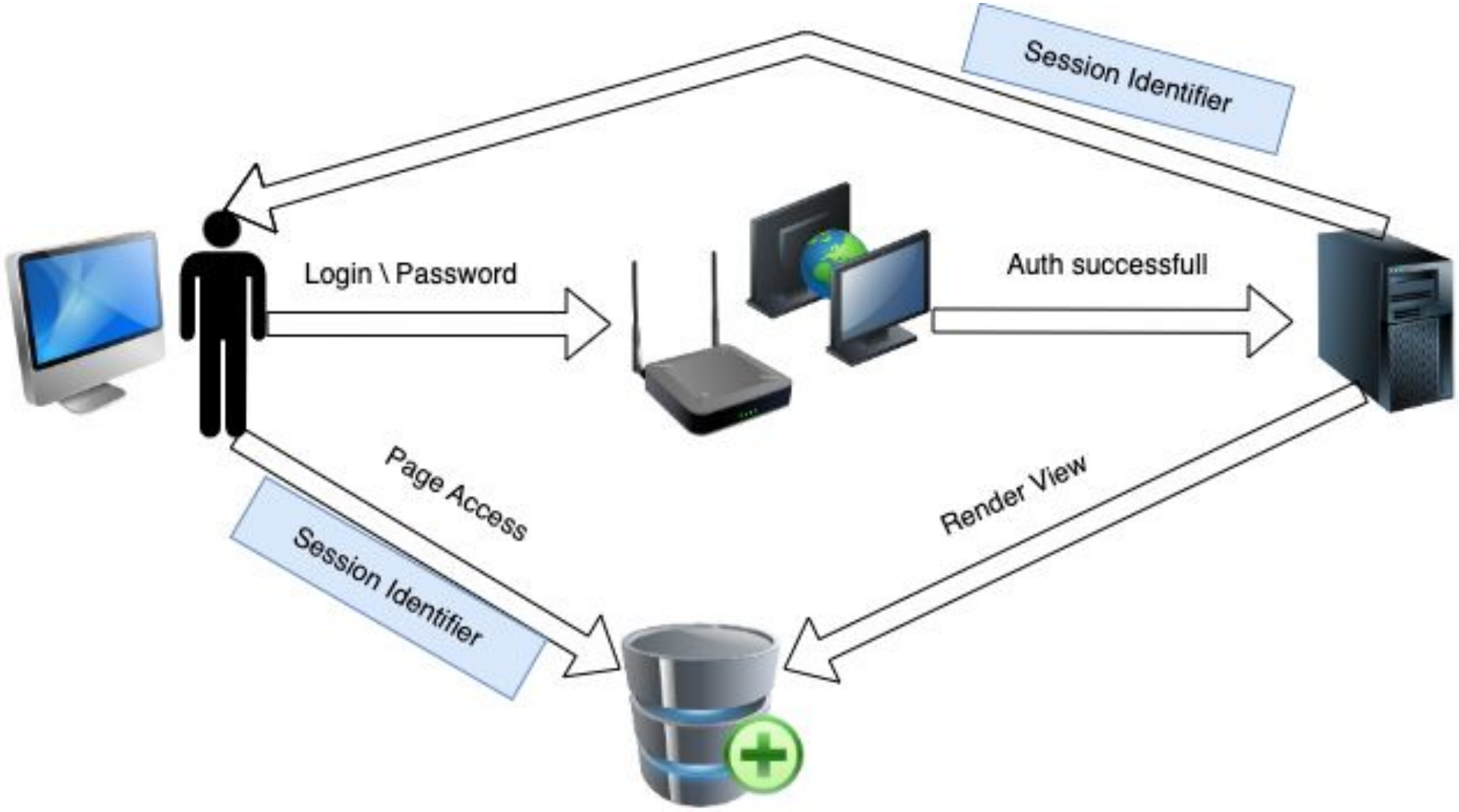
Авторизация. Механизм сессий



Однако, при повторном запросе на просмотр этой же страницы, пользователь не должен вводить данные авторизации повторно.
Но при текущей схеме невозможно предоставить санкционированный доступ к просмотру страницы.

Чтобы реализовать подобный функционал необходимо при авторизации на сервере выдавать пользователю некий уникальный идентификатор, который бы пользователь использовал при повторном обращении к ресурсу.

Авторизация. Механизм сессий



Авторизация. Механизм сессий

Наиболее простым и распространенным способом хранения и передачи уникального идентификатора пользователя является механизм cookie и идентификатора сессии.

Идентификатор сессии генерируется сервером и отправляется клиенту в виде cookie.

Cookie – это данные, которые сервер отправляет браузеру, а браузер сохраняет их на стороне пользователя.

При каждом последующем запросе к серверу браузер автоматически отправляет cookie в заголовке запроса.

Сервер сохраняет данные о состоянии пользователя по идентификатору сессии, который был отправлен в cookie. Для каждой новой сессии сервер генерирует новый идентификатор.

Авторизация. Механизм сессий

Сессии могут быть временными или постоянными.

Временные сессии хранятся на сервере до тех пор, пока пользователь не закроет браузер или до истечения TTL (Time To Live).

Постоянные сессии хранятся на сервере дольше и могут быть использованы для автоматической авторизации пользователя при следующем заходе на сайт.

Для грамотной реализации временных сессий следует задавать время жизни сессии, которое может быть продлено при каждом взаимодействии пользователя с сервером.

Механизм постоянных сессий не рекомендуется использовать с целью обеспечения безопасности данных пользователей.

Авторизация. Механизм сессий

Существуют и иные способы реализации механизма сессий, а также их хранения.

В самом простом варианте идентификатор сессии хранится в файле на сервере. Каждый файл - свой идентификатор сессии.

Из недостатков такого способа стоит отметить неоптимизированный доступ к файлам, избыточность при хранении данных.

Другим вариантом хранения сессий является реляционная СУБД. В данном случае идентификатор хранится в таблице, в которой можно указать дополнительные свойства сессии, такие как дата и время последнего обращения пользователя к серверу и тд.

Из недостатков стоит отметить возможный конкурентный доступ к СУБД, особенно при высоких нагрузках.

Авторизация. Авторизация в Laravel

Laravel предоставляет несколько разных подходов и механизмов, доступных для реализации механизма авторизации и аутентификации пользователя.

“Из коробки” фреймворк содержит встроенные службы аутентификации и сессии, которые обычно доступны через фасады Auth и Session.

Этот функционал обеспечивают аутентификацию на основе файлов Cookies для запросов, которые инициируются из веб-браузеров.

Они предоставляют методы, которые позволяют приложению проверять учетные данные пользователя и аутентифицировать пользователя.

Кроме того, эти службы автоматически сохраняют необходимые данные аутентификации в сессии пользователя и выдают cookie сессии пользователя.

Авторизация. Авторизация в Laravel

Порядок действий для реализации простой авторизации в Laravel:

1. Создать таблицу users, скорректировать модель User.
2. Создать Контроллер, метод и маршрут (метод GET) для отображения View с формой авторизации.
3. В этом же контроллере реализовать метод, который принимает данные из формы авторизации.
4. Реализовать маршрут (метод POST), который ведет на этот метод.
5. В методе проверить, что пользователь с такими данными существует, получить объект модели $user.
6. Вызвать метода фасада Auth::login(), куда передать объект модели пользователя
7. К необходимым маршрутам, которые требуют доступа авторизованного пользователя привязать Authenticate middleware.

Авторизация. Laravel API.

Реализация аутентификации при работе с API несколько отличается от классической. Laravel предоставляет два пакета для этой цели:

- Passport
- Sanctum

Важно понимать, что эти пакеты и встроенные в Laravel пакеты аутентификации на основе файлов cookie не являются взаимоисключающими.

Эти пакеты в основном ориентированы на аутентификацию токена API, в то время как встроенные службы аутентификации ориентированы на web-аутентификацию на основе файлов cookie.

Многие приложения будут использовать как встроенные службы аутентификации Laravel на основе файлов cookie, так и один из пакетов API-аутентификации Laravel.

Авторизация. Laravel API. Passport

Passport – это провайдер аутентификации **OAuth2**, предлагающий различные OAuth2 Grant Types («способы запросы»), которые позволяют выдавать различные типы токенов.

Это надежный, но сложный пакет для аутентификации API.

Однако большинству приложений не требуются сложный функционал, предлагаемый спецификацией OAuth2, что может сбивать с толку как пользователей, так и разработчиков.

Для установки пакета, необходимо его установить с помощью composer:

```
composer require laravel/passport
```

Авторизация. Laravel API. Passport

Далее необходимо запустить миграции, которые необходимы для работы пакета:

```
php artisan migrate
```

Следующим шагом необходимо выполнить команду установки всех необходимых файлов пакета. Для этого можно воспользоваться командой install из пространства имен команды passport:

```
php artisan passport:install
```

Авторизация. Laravel API. Passport routes

Пакет Passport помимо структуры БД и файлов конфигурации предоставляет заранее заготовленные маршруты для всех вариантов авторизации. Чтобы воспользоваться маршрутами пакета, необходимо их добавить в загрузчик соответствующего ServiceProvider (AuthServiceProvider)

```php
use Laravel\Passport\Passport;

public function boot()
{
    $this->registerPolicies();

    Passport::routes();
}
```

Авторизация. Laravel API. Passport routes

При вызове метода Passport::routes() генерируются следующие маршруты:

1. /oauth/authorize - маршрут авторизации OAuth 2.0, который используется для редиректа пользователей на страницу авторизации.
2. /oauth/clients - маршрут для получения списка клиентов OAuth.
3. /oauth/clients/{client-id} - маршрут для просмотра и редактирования информации о клиенте OAuth.
4. /oauth/personal-access-tokens - маршрут для создания новых персональных токенов.
5. /oauth/personal-access-tokens/{token-id} - маршрут для удаления персональных токенов.

Авторизация. Laravel API. Passport routes

Чтобы создать маршруты только для определенных функций, можно добавить параметр к методу Passport::routes(), например:

```
Passport::routes(function ($router) {
    $router->forAuthorization();
    $router->forAccessTokens();
});
```

Это создаст маршруты только для авторизации и получения токенов доступа.

Авторизация. Laravel API. Passport routes

После установки и настройки пакета, необходимо создать свой маршрут для создания токена аутентификации

```php
use Illuminate\Http\Request;

Route::post('/api/login', function (Request $request) {
    $credentials = $request->only('email', 'password');

    if (Auth::attempt($credentials)) {
        $user = Auth::user();
        $token = $user->createToken('MyApp')->accessToken;
        return response()->json(['token' => $token]);
    }

    return response()->json(['error' => 'Invalid credentials'], 401);
});
```

Авторизация. Laravel API. Passport

Ниже перечислены несколько полезных методов для работы с Passport:

- createToken($name): Создает новый токен аутентификации для пользователя с заданным именем.
- tokens(): Возвращает все токены аутентификации, созданные для пользователя.
- token(): Возвращает текущий токен аутентификации для пользователя.
- revoke(): Отзывает текущий токен аутентификации пользователя.
- revokeOtherTokens(): Отзывает все токены аутентификации, кроме текущего.

Passport можно использовать как для авторизации пользователей, так и для авторизации сервисов, которые будут взаимодействовать с текущим.

Авторизация. Laravel API. Sanctum

Sanctum - это официальный пакет аутентификации для Laravel, который предоставляет простые инструменты для аутентификации SPA (Single Page Applications), мобильных приложений и API.

Sanctum использует токены аутентификации, которые передаются через заголовок авторизации Bearer для аутентификации пользователя и взаимодействия с API.

Пакет обычно используется для клиент-серверной аутентификации. Для реализации аутентификации сервер-сервер рекомендуется использовать протокол OAuth 2.0 и пакет Passport.

Авторизация. Laravel API. Sanctum

Для установки Sanctum необходимо воспользоваться composer:

```
composer require laravel/sanctum
```

Следующим шагом необходимо выполнить миграции, которые необходимы для работы пакета:

```
php artisan migrate
```

Далее необходимо вынести конфигурационный файл пакета в приложение:

```
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
```

Авторизация. Laravel API. Sanctum

Для функционирования пакета для web и для api маршрутов, сконфигурируем ядро HTTP (`app/Http/Kernel.php`).

Это позволит Sanctum создавать токены аутентификации при выполнении запросов.

```php
use Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful;

protected $middlewareGroups = [
    'web' => [
        // ...
        EnsureFrontendRequestsAreStateful::class,
    ],

    'api' => [
        // ...
        EnsureFrontendRequestsAreStateful::class,
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];
```

Авторизация. Laravel API. Sanctum

Создадим необходимые маршруты. Это добавит маршруты для создания и удаления токенов аутентификации, а также маршрут для обновления токена доступа.

```php
use Laravel\Sanctum\Http\Controllers\CsrfCookieController;
use Laravel\Sanctum\Http\Controllers\NewAccessTokenController;
use Laravel\Sanctum\Http\Controllers\TokenController;

Route::post('/login', [LoginController::class, 'login']);
Route::post('/logout', [LoginController::class, 'logout']);

Route::post('/tokens/create', [TokenController::class, 'store']);
Route::delete('/tokens/{token}', [TokenController::class, 'destroy']);

Route::get('/csrf-cookie', CsrfCookieController::class.'@show');

Route::post('/tokens/refresh', [NewAccessTokenController::class, 'store']);
```

Авторизация. Laravel API. Sanctum

Последним шагом необходимо закрыть необходимые маршруты авторизацией

```
Route::middleware('auth:sanctum')->get('/dashboard', function () {
    return view('dashboard');
});
```

Авторизация. Laravel API. Sanctum. Конфигурация

Файл config/sanctum.php в Laravel предоставляет настройки для пакета Sanctum, который обеспечивает аутентификацию и авторизацию API в приложениях Laravel.

1. stateful: настройка определяет, должен ли Sanctum сохранять состояние аутентификации на сервере или нет. По умолчанию этот параметр установлен в true.
2. expiration: настройка определяет время жизни токена Sanctum в минутах. Значение по умолчанию - null, что означает, что токен будет действителен до тех пор, пока пользователь не выйдет из системы.
3. personal_access_token_expiration: настройка определяет время жизни персональных токенов Sanctum в днях. Значение по умолчанию - null, что означает, что токен будет действителен до тех пор, пока пользователь не выйдет из системы.

Авторизация. Laravel API. Sanctum. Конфигурация

4. guard: настройка определяет охранника, который будет использоваться Sanctum для аутентификации пользователя. Значение по умолчанию - web.

5. prefix: настройка определяет префикс маршрута Sanctum. Значение по умолчанию - api.

6. middleware: настройка определяет список посредников, которые будут применяться к маршрутам Sanctum.

7. prefix_personal_access_tokens: настройка определяет префикс URI, который используется для создания, просмотра и удаления персональных токенов Sanctum. Значение по умолчанию - api.

8. expiration_enforced: настройка определяет, должен ли Sanctum проверять время жизни токена при каждом запросе. Значение по умолчанию - false.

Авторизация. Laravel API. Sanctum. Конфигурация

9. abilities: настройка определяет список разрешений, которые можно использовать с персональными токенами Sanctum.

10. table_names: настройка определяет имена таблиц, которые используются Sanctum для хранения токенов и связей.

Авторизация. Laravel. Roles and Permissions

Роль - это набор разрешений, которые предоставляются пользователю для выполнения определенных действий.

Например, администратор может иметь все права на сайте, в то время как обычный пользователь может иметь доступ только к определенным страницам.

Право - это отдельное действие, которое может быть выполнено пользователем.

Например, право на создание новых записей, редактирование существующих записей и т.д.

Как правило, роль содержит в себе набор определенных прав и бизнес-логикой приложения предусматривается наличие ролей с предустановленным набором прав, либо же система максимально конфигурируемая.

Авторизация. Laravel. Roles and Permissions

В Laravel нет встроенного пакета для работы с правами и ролями. Тем не менее, разработчики фреймворка рекомендуют использовать пакет стороннего разработчика spatie.

Для установки пак

```
composer require spatie/laravel-permission
```

и запустить команду миграции

```
php artisan migrate
```

Пакетом будут созданы несколько новых таблиц, которые являются максимально универсальными и подойдут для любой схемы реляционной БД. Следующим шагом необходимо создать модели для новых таблиц.

Для модели User нелобходимо добавить трейт HasRoles или HasPermissions.

Авторизация. Laravel. Roles and Permissions

Для того, чтобы назначить роли и права пользователям, необходимо произвести следующие действия:

```php
$user = User::find(1);
$user->assignRole('admin');

$role = Role::create(['name' => 'writer']);
$role->givePermissionTo('edit articles');
```

Данный код для user с id равным 1 добавит роль admin, а также создаст новую роль writer, которой предоставит права edit articles.

При назначении роли writer любому пользователю, тот автоматически получит права edit articles.

Авторизация. Laravel. Roles and Permissions

Определить доступность маршрута для пользователя с ролью admin можно следующим образом:

```
Route::group(['middleware' => ['auth', 'role:admin']], function () {
    Route::resource('roles', 'RoleController');
    Route::resource('permissions', 'PermissionController');
});
```

Это значит, что пользователь с ролью admin может получить доступ к ресурсам roles и permissions соответствующих контроллеров.

В данной бизнес-логике подразумевается доступ к управлению ролями и правами в системе.

Авторизация. Laravel. Roles and Permissions

Определить доступность маршрута для пользователя по праву можно следующим образом:

```
Route::group(['middleware' => ['auth', 'permission:edit articles']], function () {
    // ...
});
```

Проверить право пользователя в рамках метода, сервиса и тд можно с помощью следующего кода:

```
if (auth()->user()->can('edit articles')) {
    // ...
}
```

Авторизация. Б - безопасность

Важно понимать, что авторизация подразумевает под собой наличие в системе неких защищенных разделов, доступ к которым предоставляется ограниченному кругу лиц.

Исходя из этого, логично предположить, что есть вероятность, отличная от нуля, что существует ряд лиц, которые захотят получить несанкционированный доступ к защищенным разделам системы.

Разработчики подобных систем должны подходить к функционалу авторизации и аутентификации максимально серьезно: любая допущенная уязвимость может привести к взлому всей системы.

Чтобы не допустить подобных сценариев, выделим ряд потенциальных уязвимостей, которым может быть подвержена система.

Авторизация. Б - безопасность

Уязвимости в хранении паролей: Если пароли пользователей хранятся в незашифрованном виде, злоумышленник может получить доступ к базе данных и узнать пароли пользователей.
Решение: хранить пароли в виде хэша, созданного с помощью асимметричного алгоритма шифрования.

Атаки перебора пароля: Если система не ограничивает количество неудачных попыток ввода пароля, то злоумышленник может осуществить атаку перебора пароля, что может привести к успешной авторизации.
Решение: реализовать подсистему, которая бы блокировала аккаунт после N неудачных попыток авторизации.

Авторизация. Б - безопасность

Атаки типа "Man-in-the-middle": Злоумышленник может перехватить данные, передаваемые между клиентом и сервером, и получить доступ к личной информации пользователя.
Решение: использовать только защищенные протоколы передачи конфиденциальных данных, например, HTTPS.

Уязвимости валидации входных данных: злоумышленник может осуществить атаку внедрения SQL-кода и получить доступ к базе данных.
Решение: валидация входящих данных, экранирование специальных символов.

Авторизация. Б - безопасность

Уязвимости в процессе аутентификации: Некоторые системы могут использовать слабые методы аутентификации, такие как Basic Auth, которые могут быть скомпрометированы злоумышленником.
Решение: Не использовать слабые методы аутентификации, либо использовать их параллельно с более надежными.

Уязвимости в процессе авторизации: Некоторые системы могут не достаточно проверять права доступа пользователя к различным ресурсам, что может привести к несанкционированному доступу к конфиденциальной информации.
Решение: сконфигурировать систему таким образом, чтобы для каждого защищенного маршрута определялся набор прав для доступа к нему.

Авторизация. К - конец.

В - Вопросы?