

Лекция 7

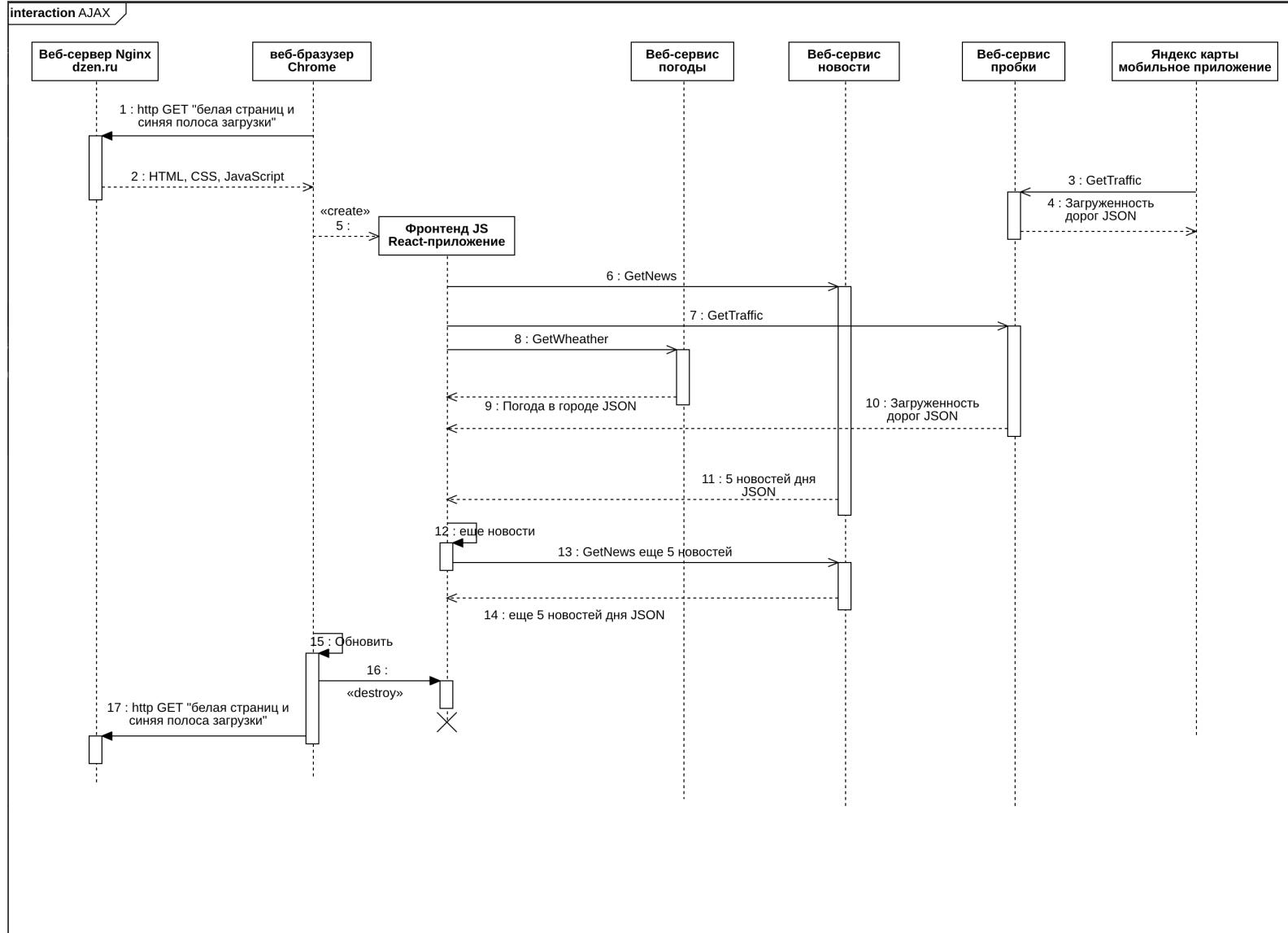
JWT. SSO

Проектирование систем и продуктовая веб-разработка

Канев Антон Игоревич

Диаграмма последовательности

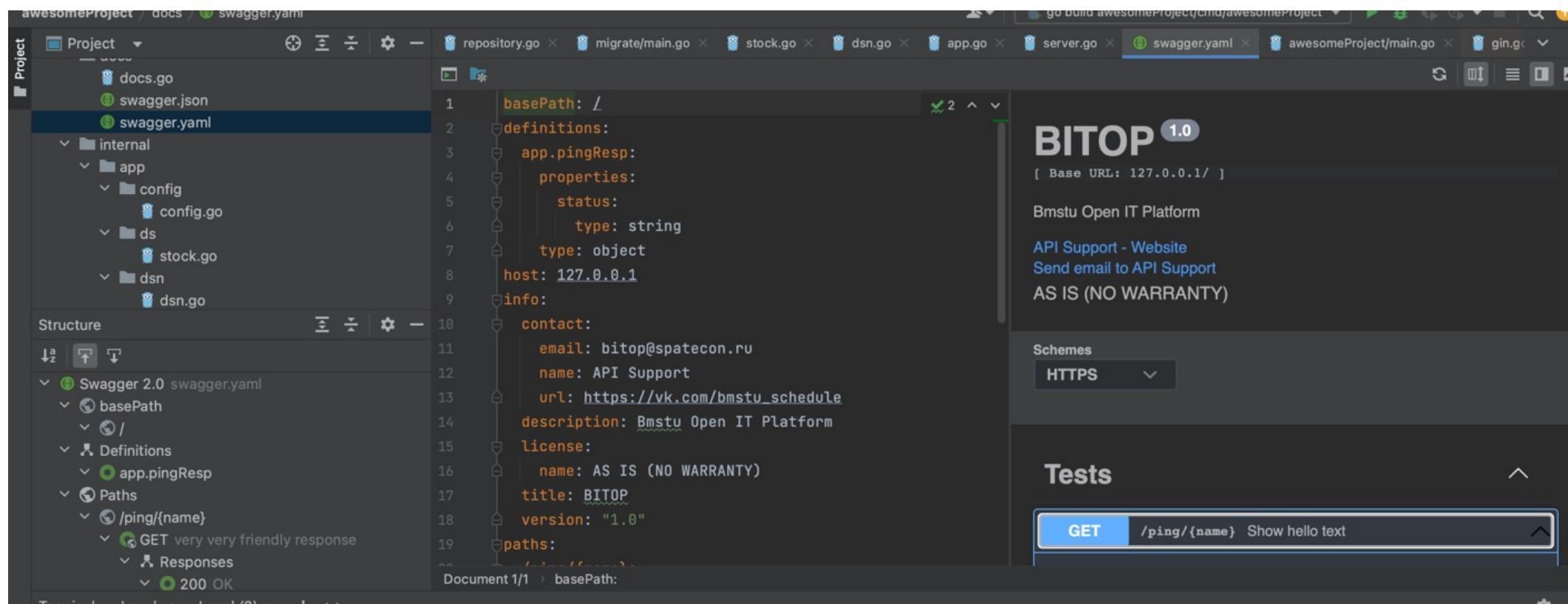
- Этот пример плохой - **Слишком мелкая диаграмма для отчета**
- На этой диаграмме мы хотим описать последовательность действий
- Например HTTP запросы, которые нам нужно выполнить для «ура сценария», например, купить товар.
- Эта последовательность нам пригодится при тестировании сервиса
- Мы укажем какие действия в каком порядке надо выполнить и какие данные мы получаем в результате



Swagger Go

- Опишем наши методы в коде
- Сгенерируем файлы документации

```
// Ping godoc
// @Summary      Show hello text
// @Description   very very friendly response
// @Tags          Tests
// @Produce       json
// @Success       200 {object} pingResp
// @Router        /ping/{name} [get]
func (a *Application) Ping(gCtx *gin.Context) {
    name := gCtx.Param("name")
    gCtx.String(http.StatusOK, "Hello %s", name)
}
```



Swagger editor (для курсовой)

- Описанные файлы swagger можно легко просматривать, редактировать с помощью различных сервисов
- Например, swagger editor – позволяет посмотреть в графическом виде загруженный файл

The screenshot shows the Swagger Editor interface. On the left, the raw OpenAPI 3.0 YAML code for the Petstore API is displayed. On the right, the graphical API documentation is shown, featuring a navigation bar with tabs for 'Swagger Petstore - OpenAPI 3.0' (version 1.0.11, OAS 3.0), 'Try our new Editor', and 'Edit'. Below the tabs, there's a note about switching from OAS 2.0 to OAS 3.0. The main content area includes sections for 'Some useful links', 'Terms of service', 'Contact the developer', 'Apache 2.0', and 'Find out more about Swagger'. At the bottom, there's a 'Servers' dropdown set to 'https://petstore3.swagger.io/api/v3' and an 'Authorize' button. The graphical part shows the 'pet' resource with three operations: 'PUT /pet' (Update an existing pet), 'POST /pet' (Add a new pet to the store), and 'GET /pet/findByStatus' (Finds Pets by status). Each operation has a lock icon and a dropdown menu.

<https://editor.swagger.io>

Аутентификация, Авторизация (Повтор)

Аутентификация (*authentication*) — процедура проверки подлинности, например:

- проверка подлинности пользователя путём сравнения введённого им пароля (для указанного логина) с паролем, сохранённым в базе данных пользовательских логинов;
- подтверждение подлинности электронного письма путём проверки цифровой подписи письма по открытому ключу отправителя;

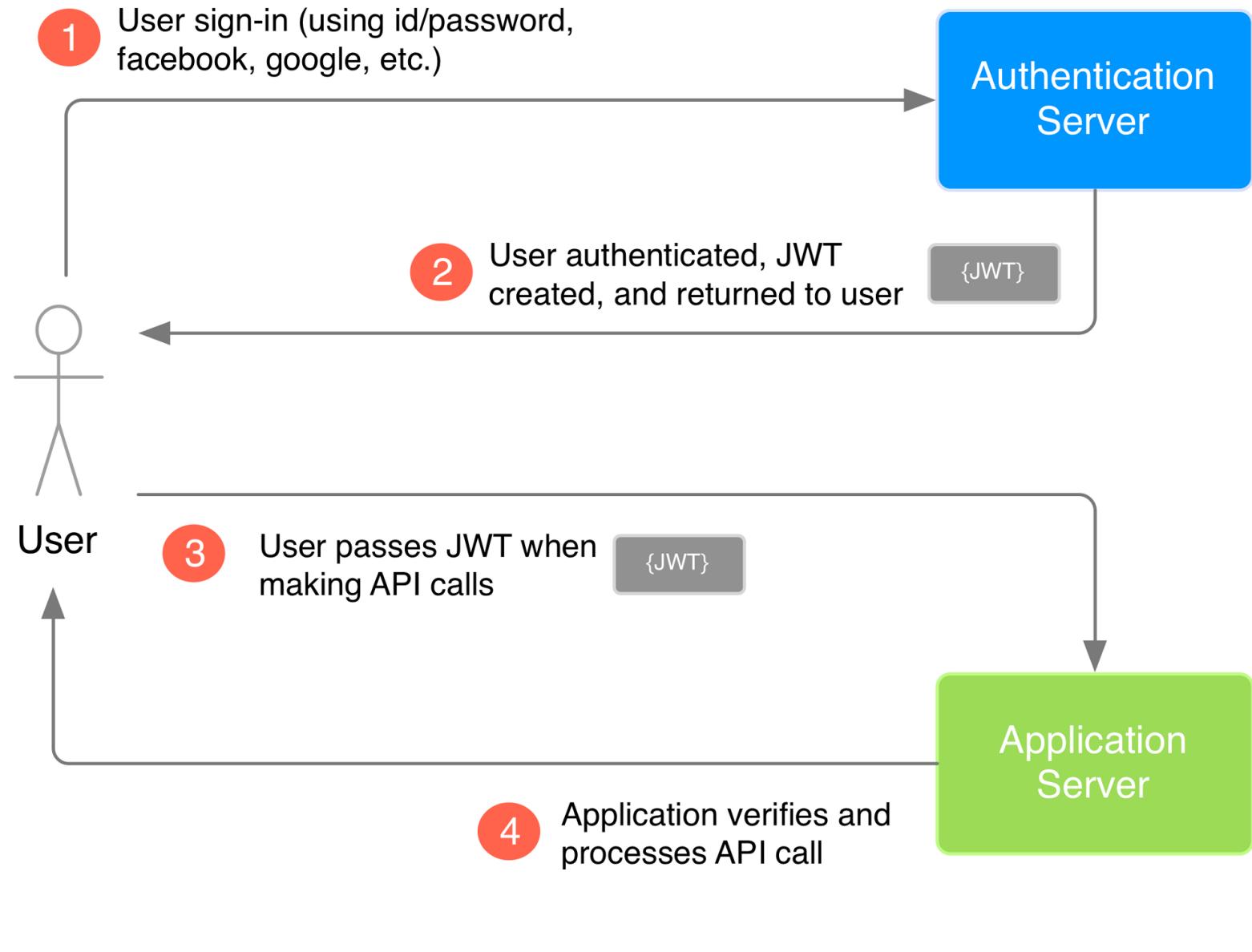
Идентификация — процедура, в результате выполнения которой для субъекта идентификации выявляется его идентификатор, однозначно определяющий этого субъекта в информационной системе (**Определяем** пользователя для заявки).

Авторизация (*authorization* «разрешение; уполномочивание») — предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий (**гость** смотрит товар, **покупатель** добавляет в заказ, **продавец** - редактирует).

- Авторизация производит контроль доступа к различным ресурсам системы в процессе работы легальных пользователей после успешного прохождения ими аутентификации.

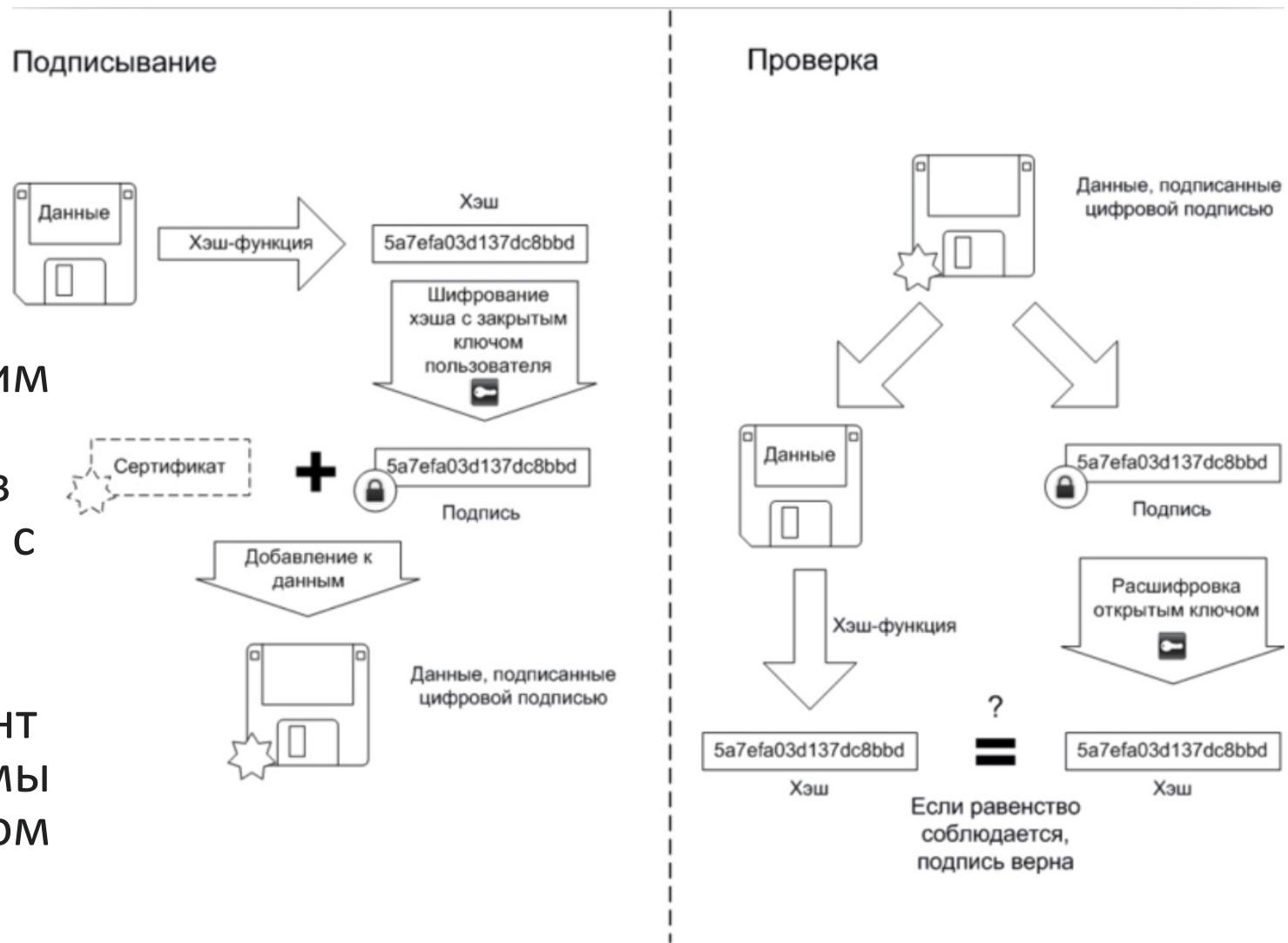
JWT

- **JSON Web Token**
- Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.
- Токены создаются сервером, **подписываются** секретным ключом (цифровая подпись) и передаются клиенту, который в дальнейшем использует данный токен для подтверждения подлинности аккаунта.



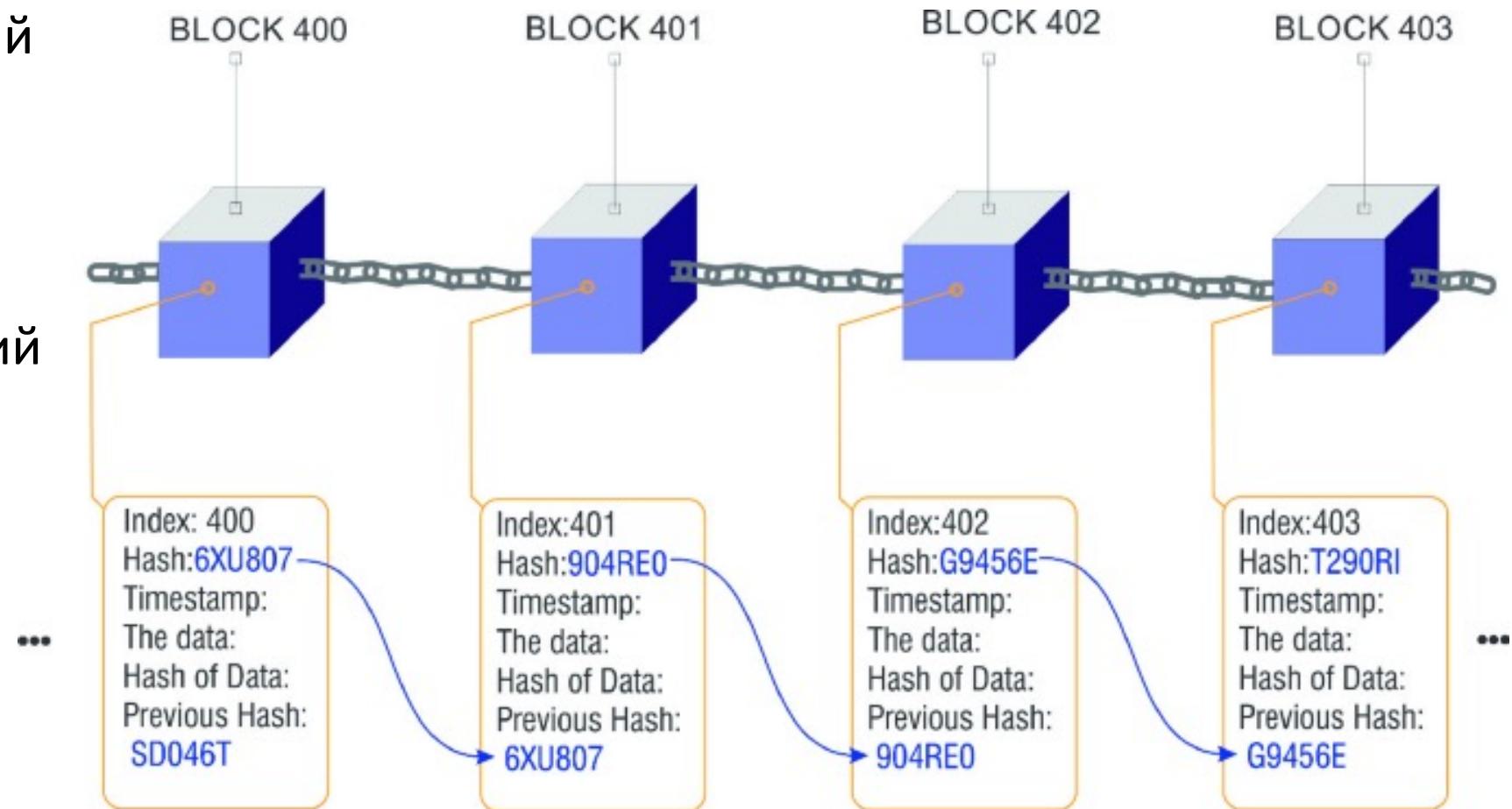
Цифровая подпись

- Электронная цифровая подпись позволяет подтвердить авторство электронного документа
- Подпись связана как с автором (ключ), так и с самим документом с помощью криптографических методов и не может быть подделана с помощью обычного копирования.
- Автор подписывает документ (хэш) закрытым ключом, а мы проверяем открытым ключом (который есть у всех)



Blockchain

- Последовательность действий/операций
- В каждом блоке новые действия
- Каждый следующий блок подписывает ссылку на предыдущий
- Подписываем «историю» действий



RSA

- Первый алгоритм одновременно для асимметричного шифрования и цифровой подписи
- Для цифровой подписи просто меняем местами
- Открытая экспонента **e** и модуль **n** публикуются
- Экспонента **d** и модуль **n** это закрытый ключ автора

$$(kx + 1) \bmod x = ?$$

$$(kx + 1)^e \bmod x = ?$$

$$((kx + b) \bmod x)^e \bmod x = ?$$

Шифрование:

- Генерируем два простых числа $n = p \cdot q$
 - Считаем для них функцию Эйлера $\varphi(n) = (p - 1) \cdot (q - 1)$;
 - Выбираем **d** и **e** взаимно простые с значением функции Эйлера $d \cdot e \equiv 1 \pmod{\varphi(n)}$
- Взять открытый ключ (e, n) Алисы
 - Взять открытый текст m
 - Зашифровать сообщение с использованием открытого ключа Алисы:
$$c = E(m) = m^e \pmod{n} \quad (1)$$

Алгоритм расшифрования:

- Принять зашифрованное сообщение c
- Взять свой закрытый ключ (d, n)
- Применить закрытый ключ для расшифрования сообщения:

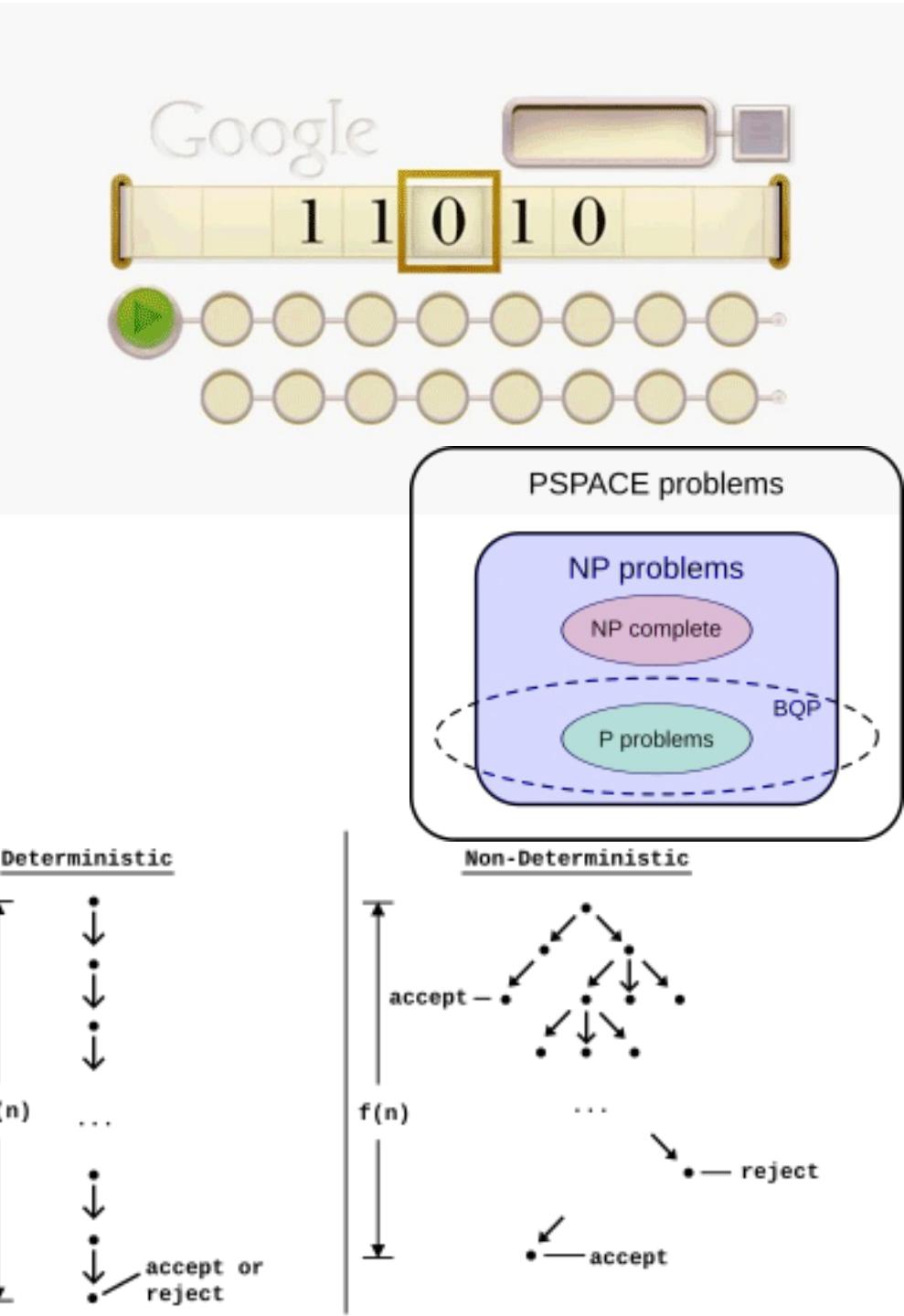
$$m = D(c) = c^d \pmod{n} \quad (2)$$

$$D(E(m)) = E(D(m)) = m^{ed} \pmod{n}$$

Доказательство можно проверить: <https://ru.wikipedia.org/wiki/RSA>

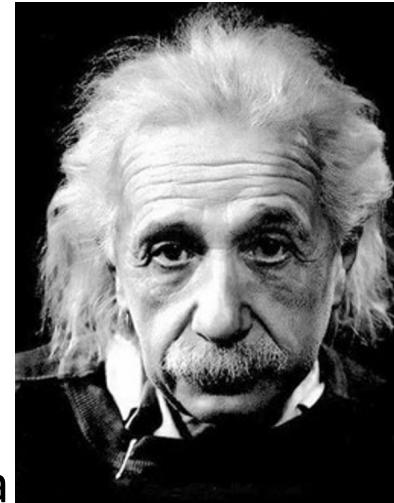
NP-трудные задачи

- В основе любого современного компьютера лежит машина Тьюринга
- Факторизация (взлом RSA) обладает экспоненциальной сложностью – растет экспоненциально от количества разрядов числа.
- Но решение проверяется (расшифровка по ключу) полиномиально - NP
- Недерминированная машина может одновременно решать несколько вариантов, «клонироваться»
- NP-трудные задачи выполняются на недерминированной машине Тьюринга за полиномиальное время
- BQP – полиномиальные квантовые алгоритмы

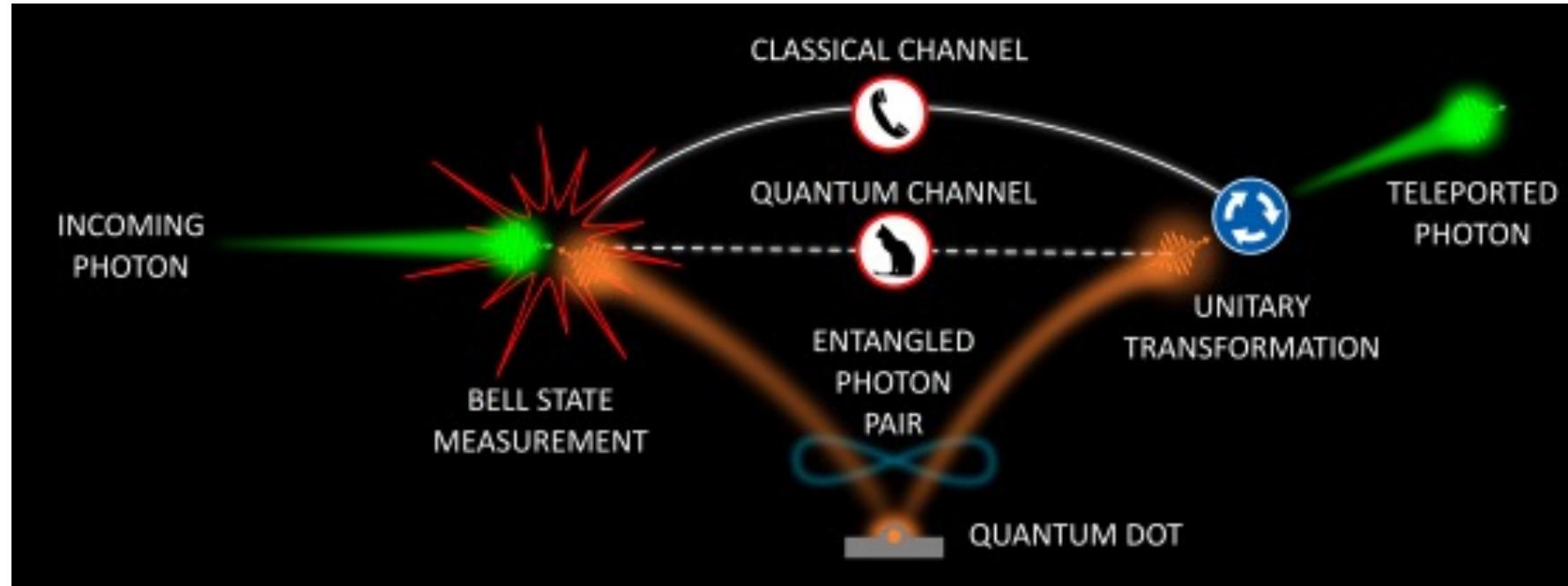


Квантовая телепортация

- В 1935 Эйнштейн, Розен и Подольский опубликовали статью с ЭПР-парадоксом, который говорил о мгновенной передаче информации при измерении одной частицы – значит квантовая механика не полна
- Нобелевская премия по физике 2022 –доказательство нарушения неравенств Белла. КМ полна
- Но для передачи информации нужен обычный канал – осмысленный сигнал передать быстрее света не получится
- Зато можно создать два генератора случайных чисел в разных местах

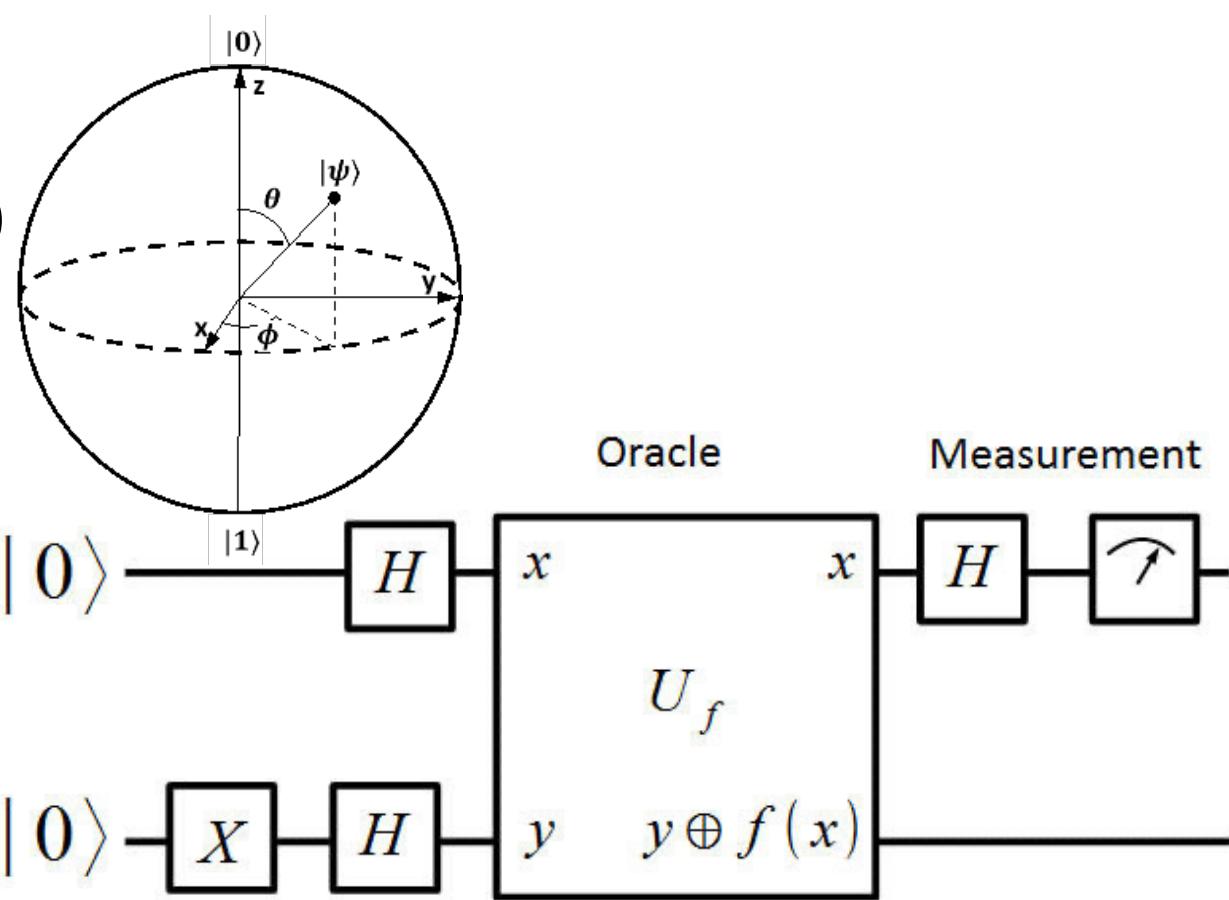


I like to think that the **Moon** is there
even if I am **not looking** at it



Квантовый компьютер

- Алгоритм Шора позволяет разложить число на простые множители за полиномиальное время, в 2001 запущен на квантовом компьютере IBM
- Слева пример алгоритма Дойча: два кубита в суперпозиции, а затем запутаны



- Одной из популярных на данный момент интерпретаций КМ является many-worlds interpretation - мы видим результат **только нашего world**
- Наблюдатель как бы расщепляется на несколько версий, мы видим только один вариант – коллапс после измерения, не можем знать все результаты

JWT

- Токен JWT состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи или данных шифрования.
- Первые два элемента — это JSON объекты определенной структуры. Третий элемент вычисляется на основании первых и зависит от выбранного алгоритма (в случае использования неподписанного JWT может быть опущен).
- Токены могут быть перекодированы в компактное представление (JWS/JWE Compact Serialization): к заголовку и полезной нагрузке применяется алгоритм кодирования Base64-URL, после чего добавляется подпись и все три элемента разделяются точками («.»).

The screenshot shows the jwt.io debugger interface. At the top, the URL is "jwt.io" and the page title is "JSON Web Tokens - jwt.io". The header bar includes links for "Debugger", "Libraries", "Ask", and "Get a T-shirt!". A dropdown menu for "ALGORITHM" is set to "HS256".

The main area is divided into two sections: "Encoded" and "Decoded".

Encoded: Displays the compact representation of the JWT token:
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpxVCJ9.eyJzdWIiOiIxMjM0NTY
30DkwIiwibmFtZSI6IkpvAG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxfjoYZge
F0NFh7HgQ

Decoded: Shows the individual components of the JWT token.

HEADER:
{
 "alg": "HS256",
 "typ": "JWT"
}

PAYOUT:
{
 "sub": "1234567890",
 "name": "John Doe",
 "admin": true
}

VERIFY SIGNATURE
HMACSHA256(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),
 secret
) secret base64 encoded

A large blue button at the bottom right says "Signature Verified" with a checkmark icon.

Метод аутентификации

- Создадим новый endpoint login
- Обязательно POST запрос
- Параметры GET запроса не шифруются

```
r.POST("/login", a.Login) // там где мы ра  
...  
type loginReq struct {  
    Login      string `json:"login"  
    Password   string `json:"password"  
}  
  
type loginResp struct {  
    ExpiresIn   int     `json:"expires_in"  
    AccessToken string `json:"access_token"  
    TokenType   string `json:"token_type"  
}  
  
func (a *Application) Login(gCtx *gin.Context) {  
    ...  
}
```

Метод аутентификации

- Проверка логина и пароля
- Формирование JWT

```
if req.Login == login && req.Password == password {
    // значит проверка пройдена
    // генерируем ему jwt
    token := jwt.NewWithClaims(cfg.JWT.SigningMethod, &ds.JWTClaims{
        StandardClaims: jwt.StandardClaims{
            ExpiresAt: time.Now().Add(cfg.JWT.ExpiresIn).Unix(),
            IssuedAt:  time.Now().Unix(),
            Issuer:     "bitop-admin",
        },
        UserUUID: uuid.New(), // test uuid
        Scopes:   []string{}, // test data
    })
}

if token == nil {
    gCtx.AbortWithError(http.StatusInternalServerError, fmt.Errorf("token is nil"))

    return
}

strToken, err := token.SignedString([]byte(cfg.JWT.Token))
if err != nil {
    gCtx.AbortWithError(http.StatusInternalServerError, fmt.Errorf("cant create str token"))

    return
}

gCtx.JSON(http.StatusOK, loginResp{
    ExpiresIn:  cfg.JWT.ExpiresIn,
    AccessToken: strToken,
    TokenType:   "Bearer",
})
```

Авторизация Golang

- Авторизацию выносим в middleware
- Она будет применяться ко всем нашим методам - endpoint

```
func (a *Application) WithAuthCheck(gCtx *gin.Context) {  
    jwtStr := gCtx.GetHeader("Authorization")  
    if !strings.HasPrefix(jwtStr, jwtPrefix) { // если нет префикса то  
        gCtx.AbortWithStatus(http.StatusForbidden) // отдаём что не  
  
        return // завершаем обработку  
    }  
  
    // отрезаем префикс  
    jwtStr = jwtStr[len(jwtPrefix):]  
  
    _, err := jwt.ParseWithClaims(jwtStr, &ds.JWTClaims{}, func(token *  
        return []byte(a.config.JWT.Token), nil  
    })  
    if err != nil {  
        gCtx.AbortWithStatus(http.StatusForbidden)  
        log.Println(err)  
  
        return  
    }  
}
```

Пример 403 Golang

- 403 ответ если пароль и логин не подходят

```
$ curl -v --location --request POST 'http://127.0.0.1:8080/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "login": "login",
    "password": "check1223"
}'

Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:8080...
*   Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /login HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.84.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 53
>
*   Mark bundle as not supporting multiuse
< HTTP/1.1 403 Forbidden
< Date: Sun, 20 Nov 2022 19:09:16 GMT
< Content-Length: 0
<
* Connection #0 to host 127.0.0.1 left intact
```

Пример 200 Golang

- 200 если логин с правильным логином и паролем

```
$ curl --location --request POST 'http://127.0.0.1:8080/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "login": "login",
    "password": "check123"
}'
{"expires_in":360000000000, "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2Njg5NzUwNz}
```

Вкладка Application. Cookie

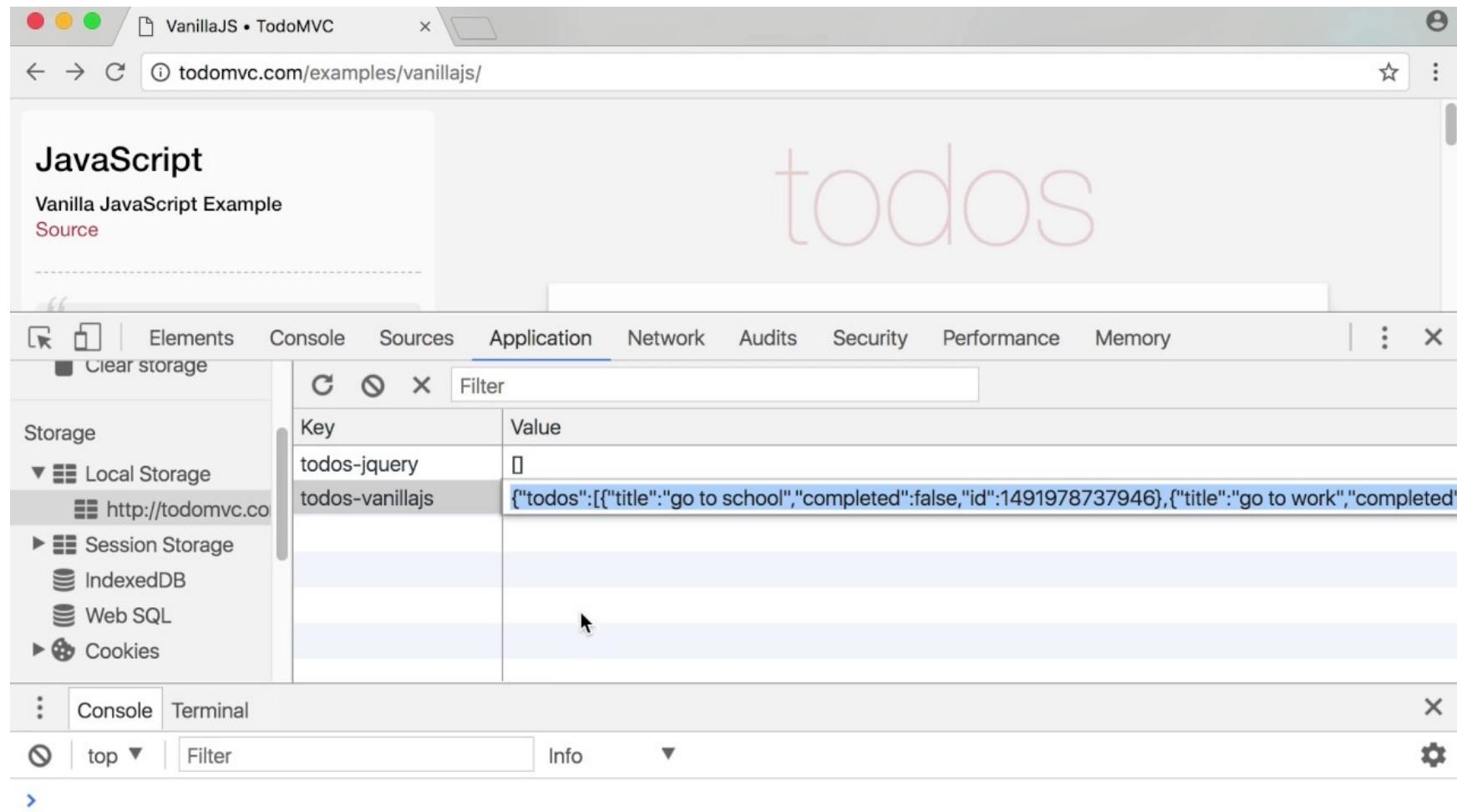
The screenshot shows the Google Chrome DevTools interface with the 'Application' tab selected. On the left, the 'Storage' section is expanded, showing 'Local Storage', 'Session Storage', 'IndexedDB', 'Web SQL', and 'Cookies'. The 'Cookies' section for the domain 'https://www.google.com' is selected, highlighted with a blue background. A blue circle with a white outline highlights the filter icon (a magnifying glass with a minus sign) in the top navigation bar. The main area displays a table of cookies:

Name	Value	Do...	P..	Expires...	Size	H.	S...	Sa...	P...	Priority
DEVTOOLS!	value	ww...	/	Session	14	✓	✓	Lax		Medium
CONSENT	PENDING+095	.go...	/	2024-0...	18		✓			Medium
SOCS	CAISHAgBE...	.go...	/	2024-0...	58		✓	Lax		Medium
NID	511=Tsugv0...	.go...	/	2023-0...	178	✓	✓	No...		Medium
AEC	ARSKqsJsU...	.go...	/	2023-0...	61	✓	✓	Lax		Medium
S	sso=sA0bSc...	.go...	/	Session	37	✓	✓			Low

At the bottom center, a message reads: 'Select a cookie to preview its value'.

Вкладка Application. Local Storage

- В варианте с JWT в 7 лабораторной мы используем Local Storage (локальное хранилище) браузера для хранения JWT
- В обоих случаях нас интересует вкладка **Application**



Проблемы

- Одной из проблем аутентификации и информационной безопасности является то, что пользователи, как правило, используют несколько различных сервисов (например, на Google, Microsoft, Apple и др.), и, соответственно, несколько учётных записей со своими логинами и паролями.
- Таким образом пользователям требуется хранить и защищать множество логинов-паролей.
- Поскольку каждый из сервисов имеет собственную систему безопасности со своими уязвимостями и недостатками, то всё это наносит ущерб удобству и безопасности пользователям
- **Решение** – использование SSO

Используйте аккаунт

Телефон или e-mail

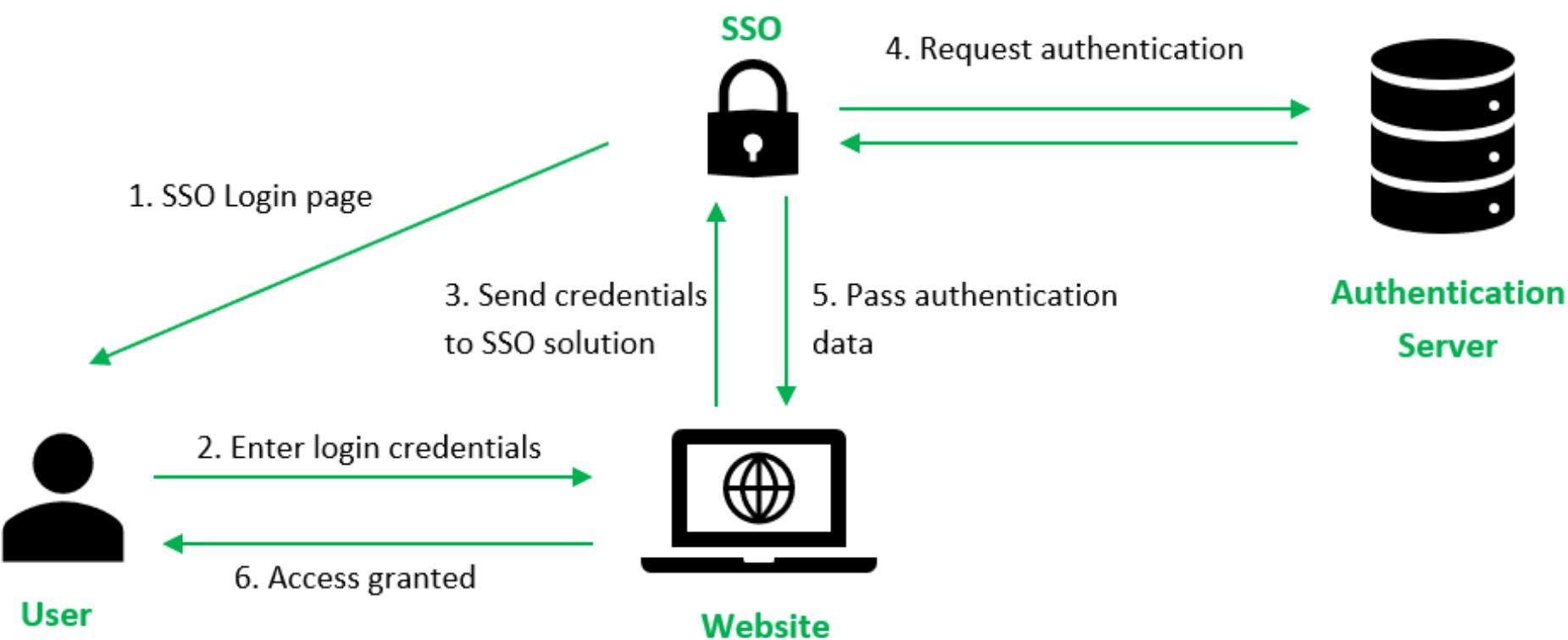
Пароль

Запомнить меня

Войти в личный кабинет

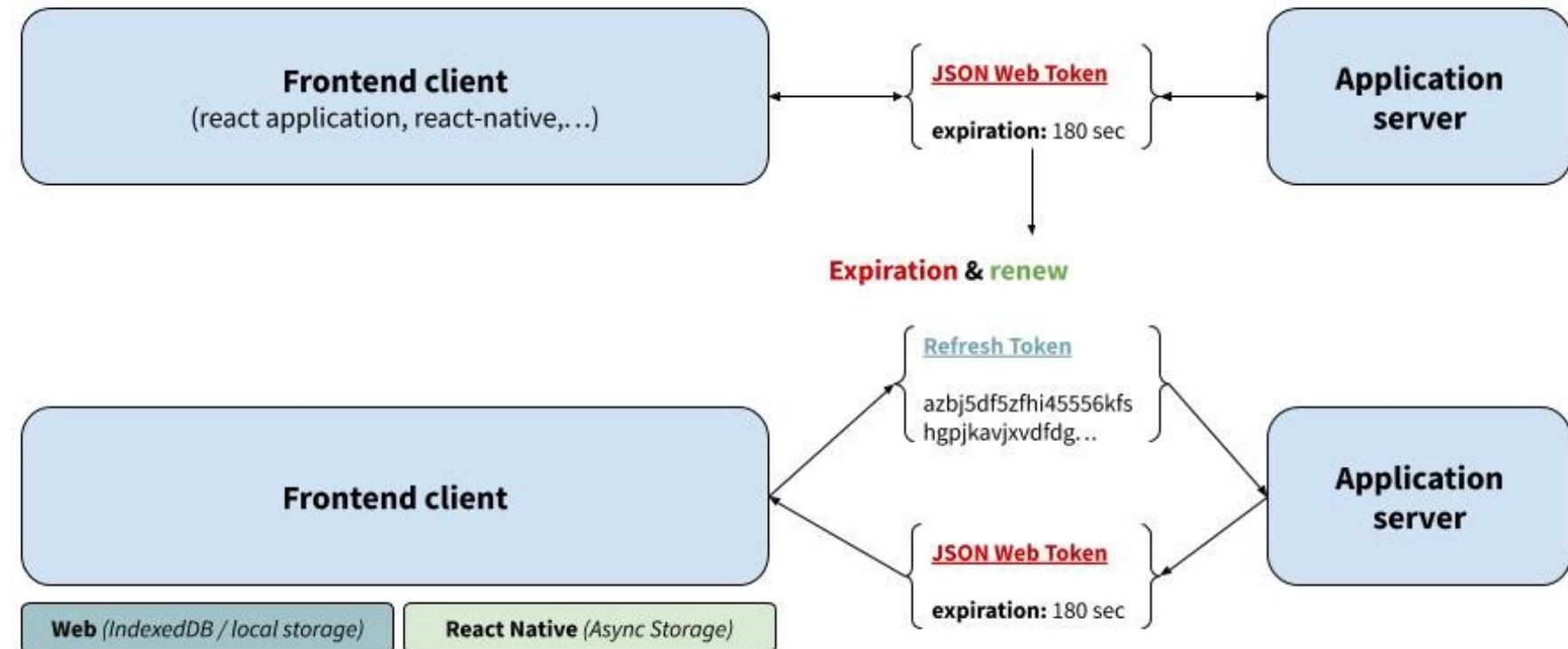
[Регистрация](#)

[Восстановление пароля](#)



Access и Refresh

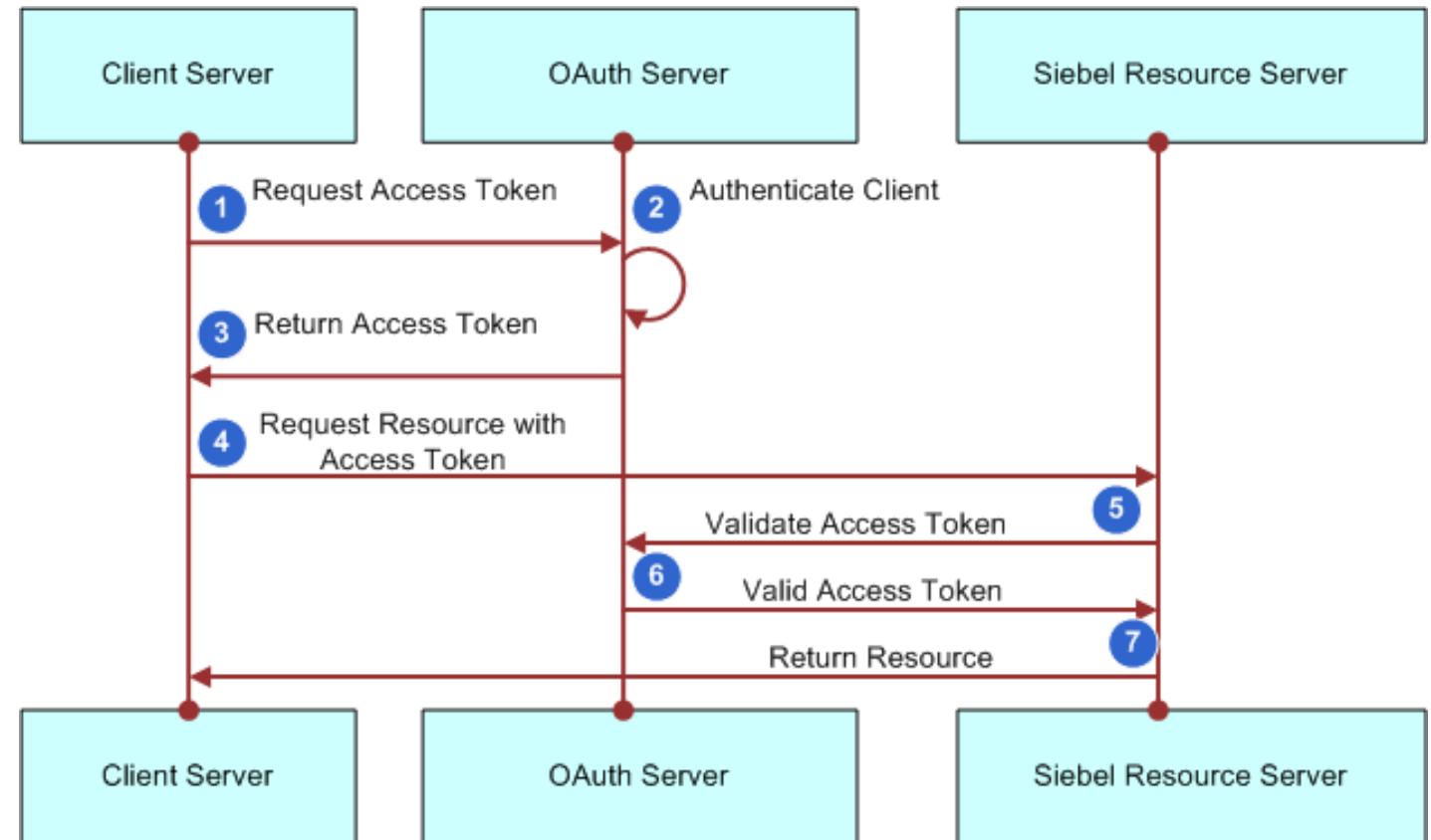
- Давайте усложним схему и сделаем два разных JWT (**OAuth**)
- **Access JWT** используем при авторизации и проверяем во всех наших методах
- **Refresh JWT** выпускаем при аутентификации и каждые 3 минуты отправляем на сервер, чтобы выпустить Access JWT



The credentials are stored here and will be used when required.

OAuth

- OAuth — открытый протокол (схема) авторизации, обеспечивающий предоставление третьей стороне ограниченный доступ к защищённым ресурсам пользователя.
- Без передачи ей (третьей стороне) логина и пароля



Двухфакторная аутентификация

- Двухфакторная аутентификация — это метод идентификации пользователя в каком-либо сервисе (как правило, в Интернете) при помощи запроса аутентификационных данных двух разных типов
- Это обеспечивает двухслойную, а значит, более эффективную защиту аккаунта от несанкционированного проникновения.
- На практике это обычно выглядит так: первый рубеж — это логин и пароль, второй — специальный код, приходящий по SMS или электронной почте.

