

Лекция 7

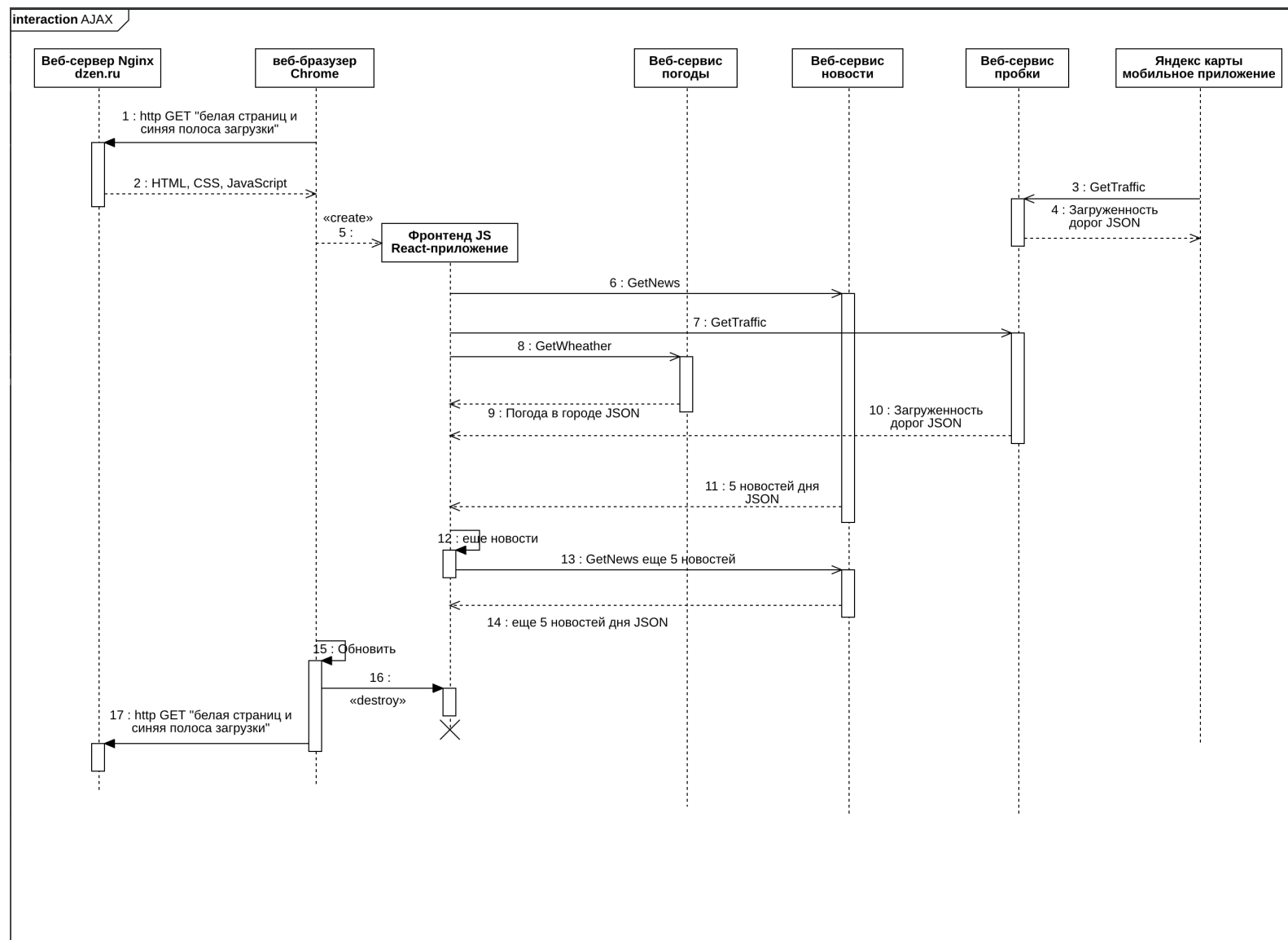
JWT. SSO

Разработка интернет приложений

Канев Антон Игоревич

Диаграмма последовательности

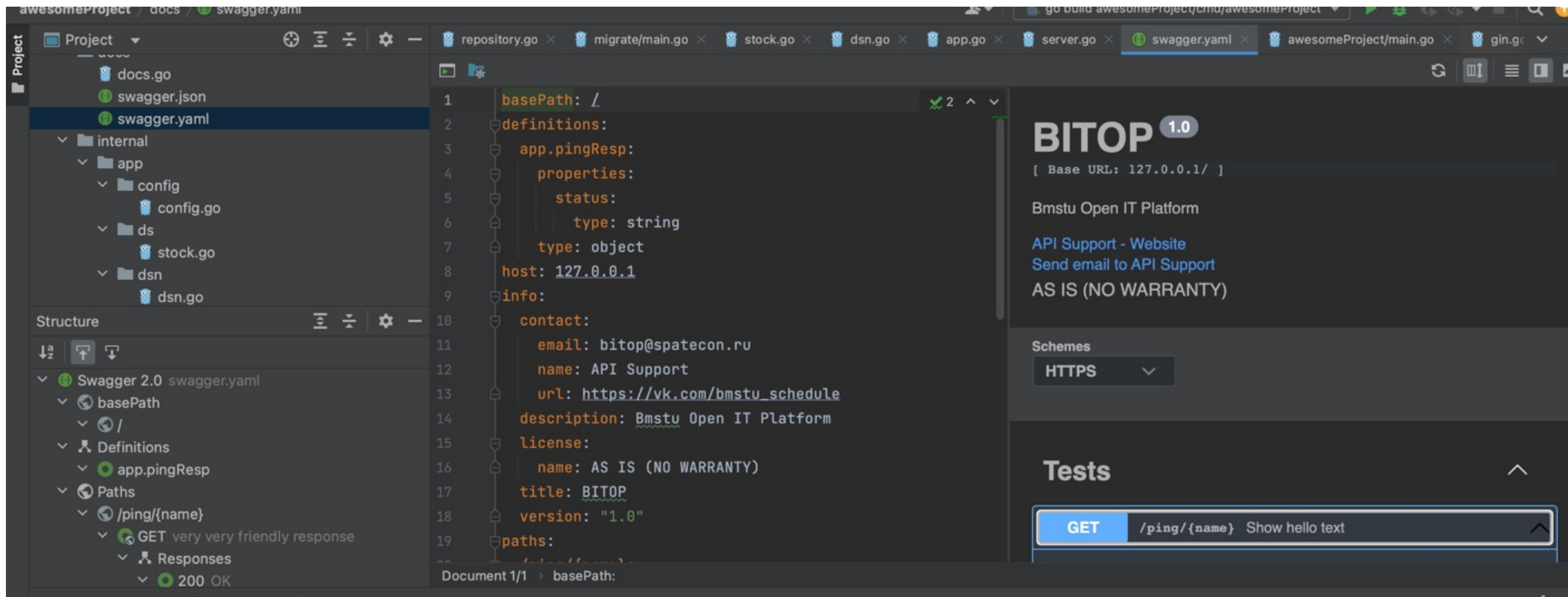
- На этой диаграмме мы хотим описать последовательность действий
- Например HTTP запросы, которые нам нужно выполнить для «ура сценария», например, купить товар.
- Эта последовательность нам пригодится при тестировании сервиса
- Мы укажем какие действия в каком порядке надо выполнить и какие данные мы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

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

<https://editor.swagger.io>

The screenshot displays the Swagger Editor interface. On the left, a code editor shows the OpenAPI 3.0 specification for the Petstore API. The specification includes an 'info' section with a title 'Swagger Petstore - OpenAPI 3.0', a description, and contact information. It also lists several endpoints under the 'paths' section, such as 'pet' (PUT, POST) and 'findByStatus' (GET). On the right, a preview pane shows the rendered version of the specification. It includes a header 'Swagger Petstore - OpenAPI 3.0' with version '1.0.11' and 'OAS 3.0'. Below the header, there is a description of the API and a list of useful links. At the bottom, there is a section for 'Servers' with a dropdown menu showing 'https://petstore3.swagger.io/api/v3'. Below this, there is a section for 'pet' with a 'Find out more' link. The main part of the preview shows a list of endpoints: 'PUT /pet' (Update an existing pet), 'POST /pet' (Add a new pet to the store), and 'GET /pet/findByStatus' (Finds Pets by status). Each endpoint has a corresponding icon and a lock icon.

Аутентификация

Аутентифика́ция (*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;
- проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла.

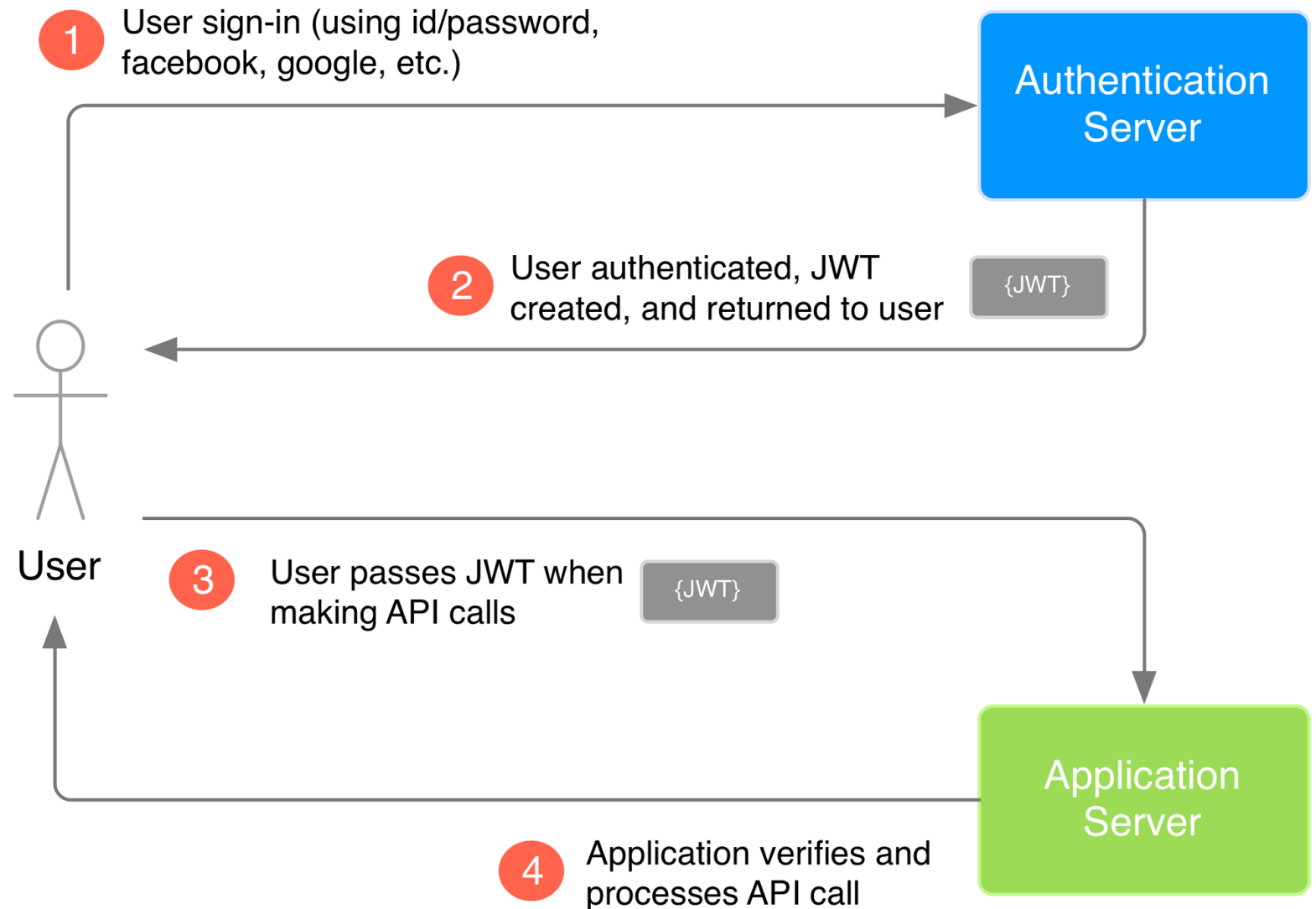
Идентификация — процедура, в результате выполнения которой для субъекта идентификации выявляется его идентификатор, однозначно определяющий этого субъекта в информационной системе.

Авторизация

- **Авториза́ция** (*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.
- 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 и передаются клиенту, который в дальнейшем использует данный токен для подтверждения подлинности аккаунта.



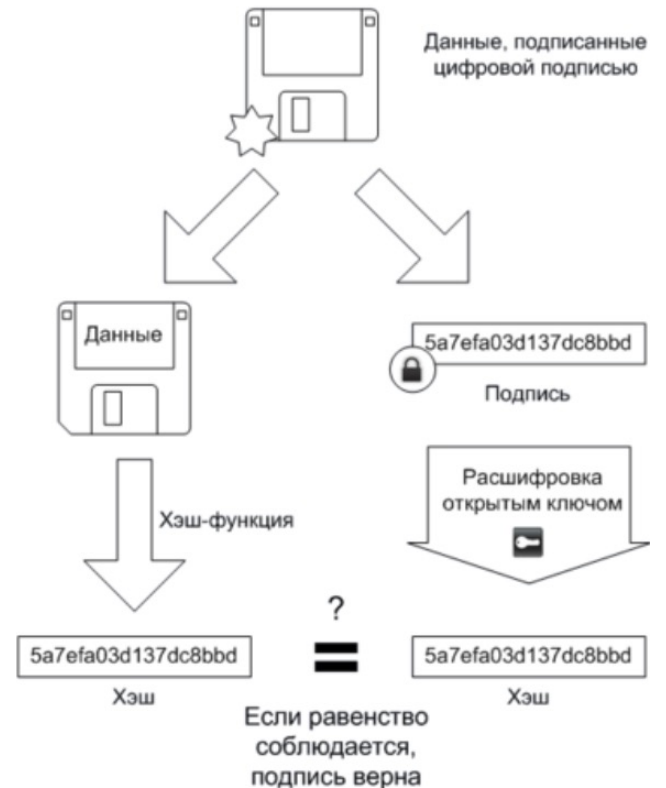
Цифровая подпись

- **Электронная цифровая подпись** позволяет подтвердить авторство электронного документа, будь то реальное лицо или, например, аккаунт в криптовалютной системе.
- Подпись связана как с автором, так и с самим документом с помощью криптографических методов и не может быть подделана с помощью обычного копирования.

Подписывание

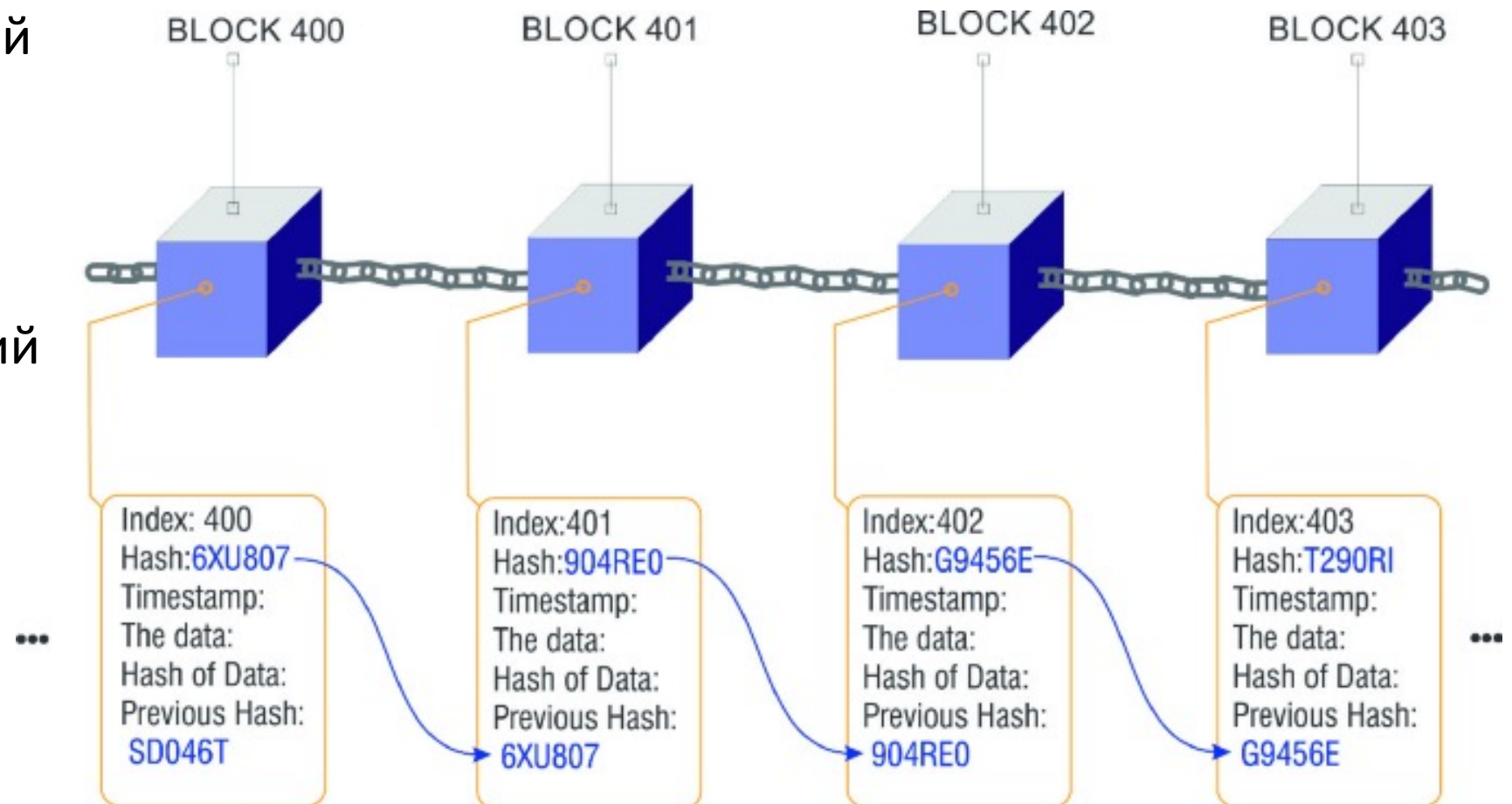


Проверка



Blockchain

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



RSA

$$n = p \cdot q$$

$$\varphi(n) = (p - 1) \cdot (q - 1);$$

$$d \cdot e \equiv 1 \pmod{\varphi(n)}$$

- Первый алгоритм одновременно для ассиметричного шифрования и цифровой подписи
- Для цифровой подписи просто меняем местами
- Открытая экспонента e и модуль n публикуются

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

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

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

- Взять *открытый ключ* (e, n) Алисы
- Взять *открытый текст* m
- Зашифровать сообщение с использованием открытого ключа Алисы:

$$c = E(m) = m^e \bmod n \quad (1)$$

Алгоритм расшифрования:

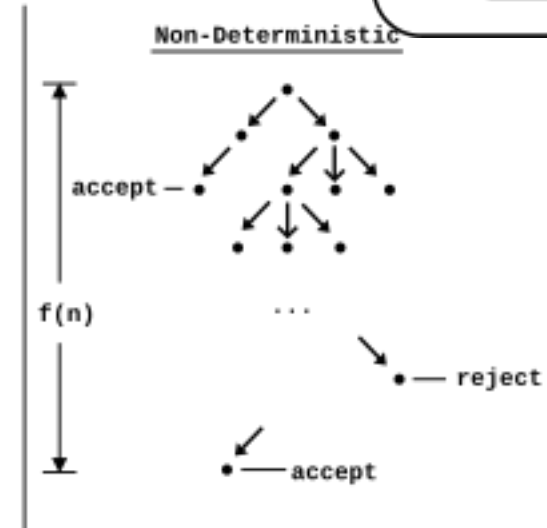
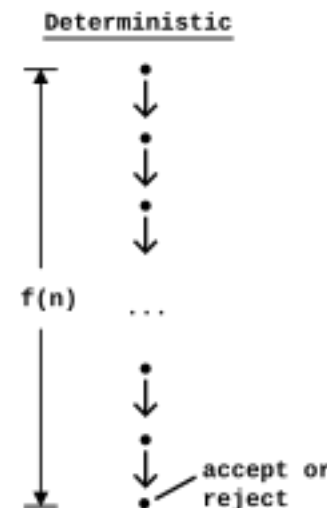
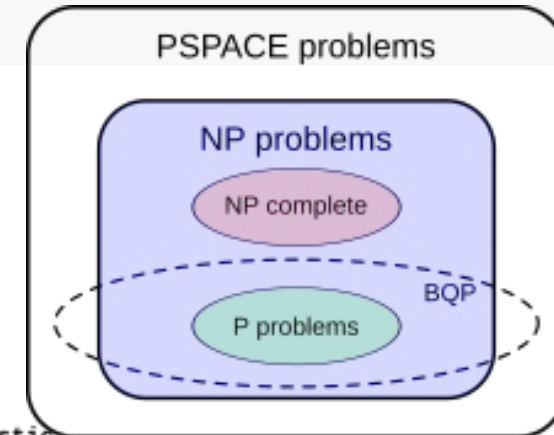
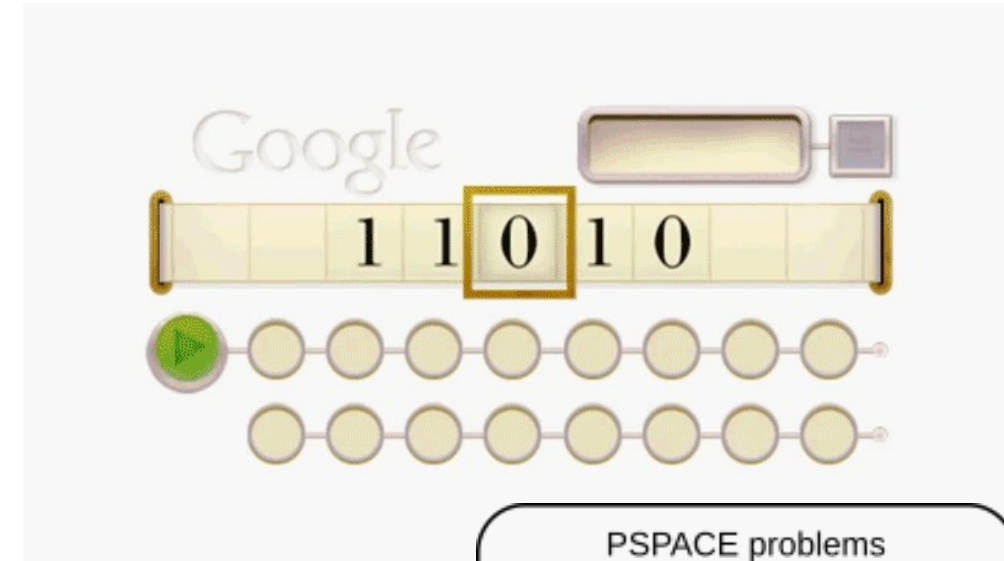
- Принять зашифрованное сообщение c
- Взять свой *закрытый ключ* (d, n)
- Применить закрытый ключ для расшифрования сообщения:

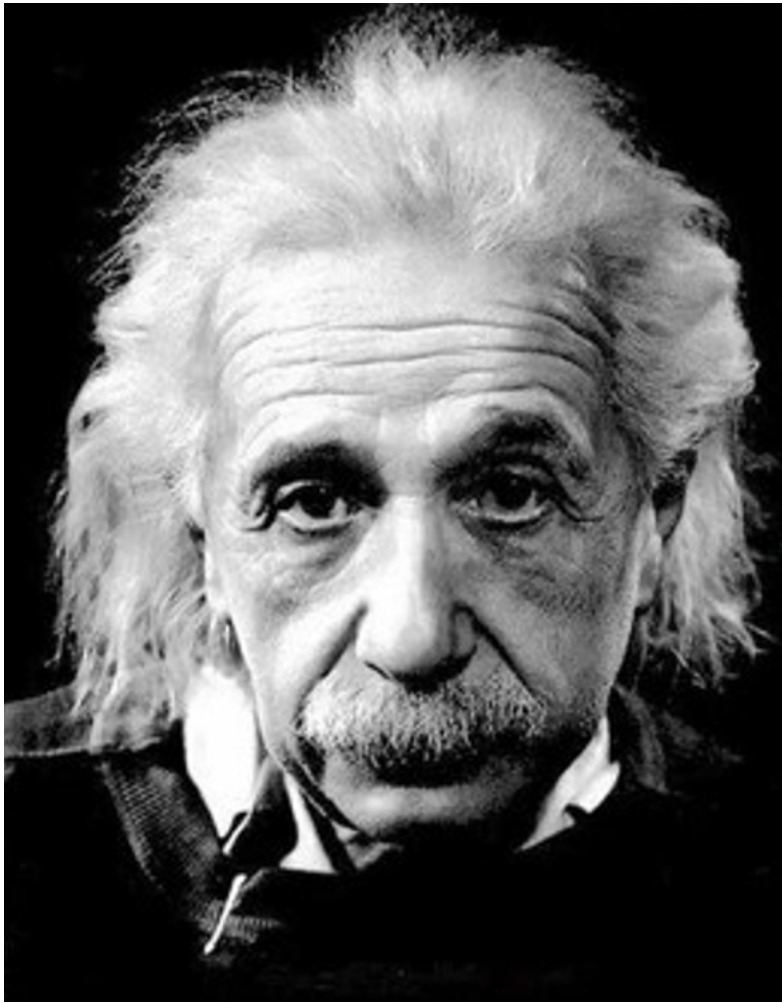
$$m = D(c) = c^d \bmod n \quad (2)$$

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

NP-трудные задачи

- В основе любого современного компьютера лежит машина Тьюринга
- Факторизация обладает экспоненциальной сложностью – растет экспоненциально от количества разрядов числа
- Недерминированная машина может одновременно решать несколько вариантов, «клонироваться»
- NP-трудные задачи выполняются на недерминированной машине Тьюринга за полиномиальное время



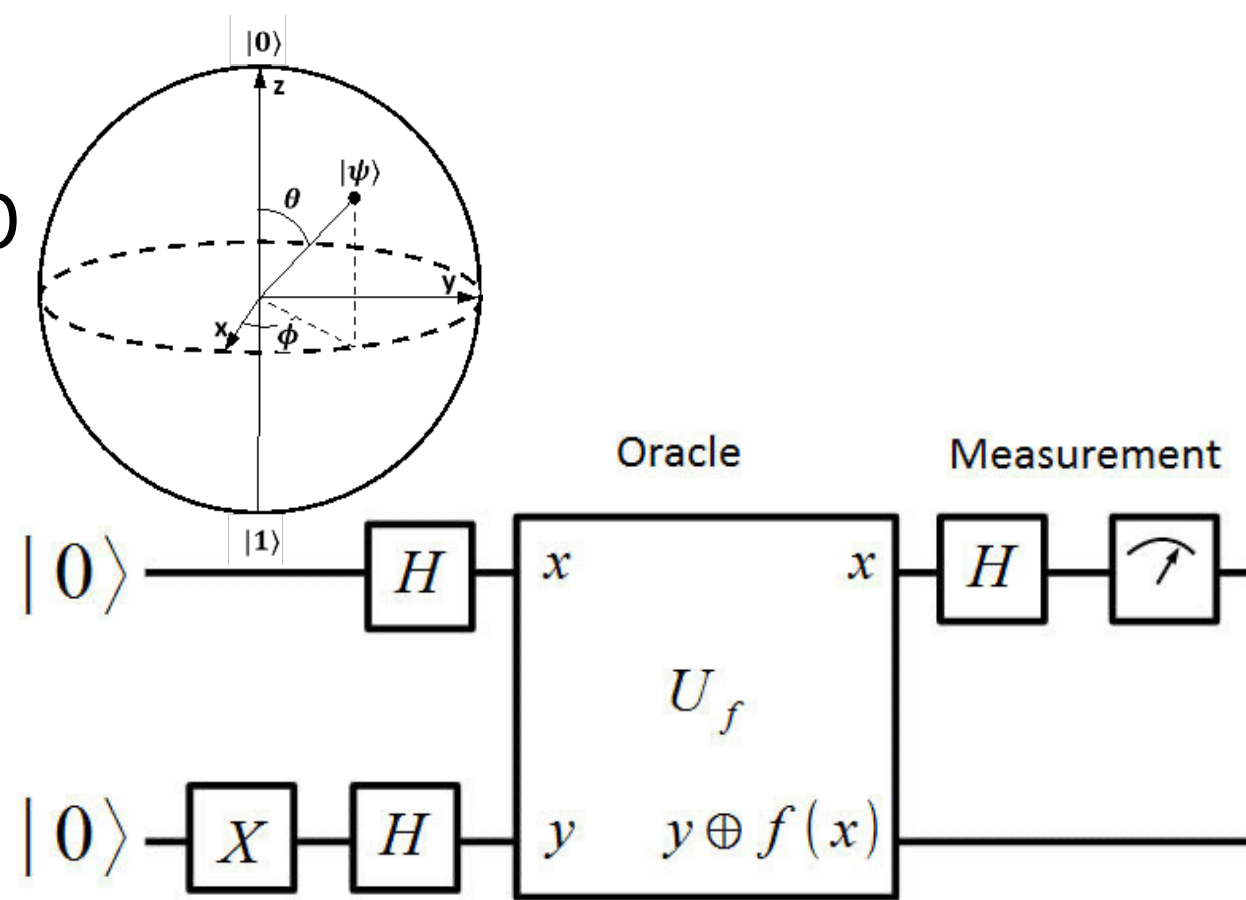
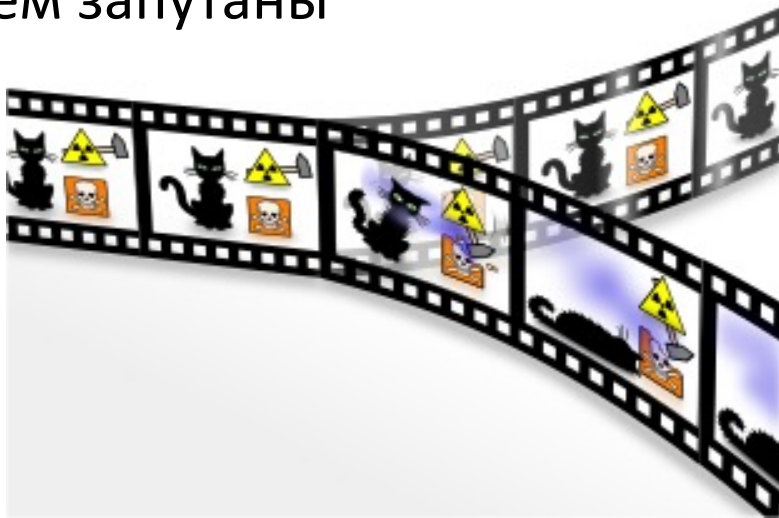


I like to think that the **Moon** is there even if I **am not looking** at it.



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 является many-worlds interpretation
- Наблюдатель как бы расщепляется на несколько версий, мы видим только один вариант – коллапс после измерения

JWT

- Токен JWT состоит из трех частей: заголовок (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 web application interface. At the top, there's a navigation bar with the JWT logo, links for 'Debugger', 'Libraries', 'Ask', and 'Get a T-shirt!', and a user profile 'Ryan'. Below the navigation bar, there's a dropdown menu for 'ALGORITHM' set to 'HS256'. The main content area is divided into two columns: 'Encoded' and 'Decoded'.

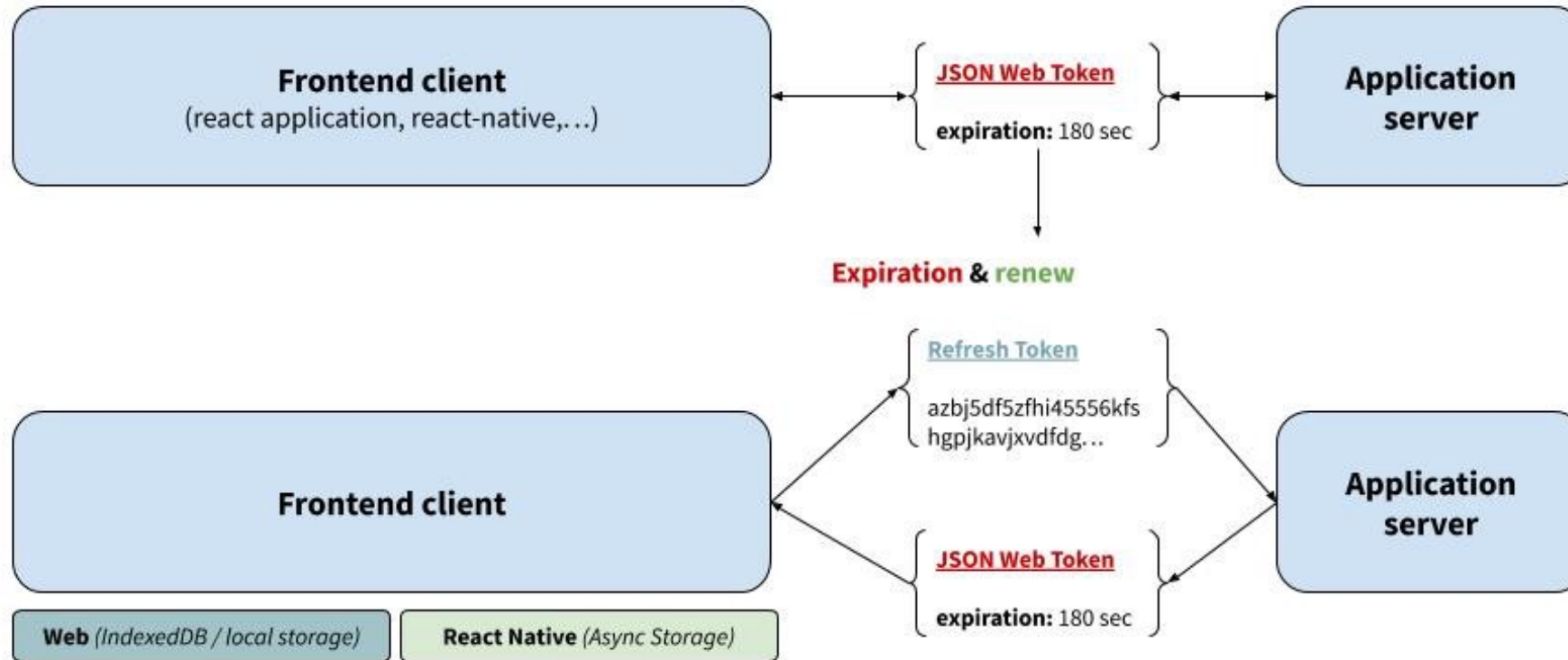
Encoded: A text box containing the encoded JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij0iMTYzNDU2Nzg5OS4iLCJ0b2R5bWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ`

Decoded: This section shows the decoded components of the token:

- HEADER:** A JSON object: `{ "alg": "HS256", "typ": "JWT" }`
- PAYLOAD:** A JSON object: `{ "sub": "1234567890", "name": "John Doe", "admin": true }`
- VERIFY SIGNATURE:** A section showing the signature verification process. It displays the HMACSHA256 function: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)`. There is a text input field for the 'secret' (containing 'secret') and a checkbox for 'secret base64 encoded'.

At the bottom of the interface, there is a large blue button with a checkmark icon and the text 'Signature Verified'.

Access и Refresh



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

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 пароле
- 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) // отдаем 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

<

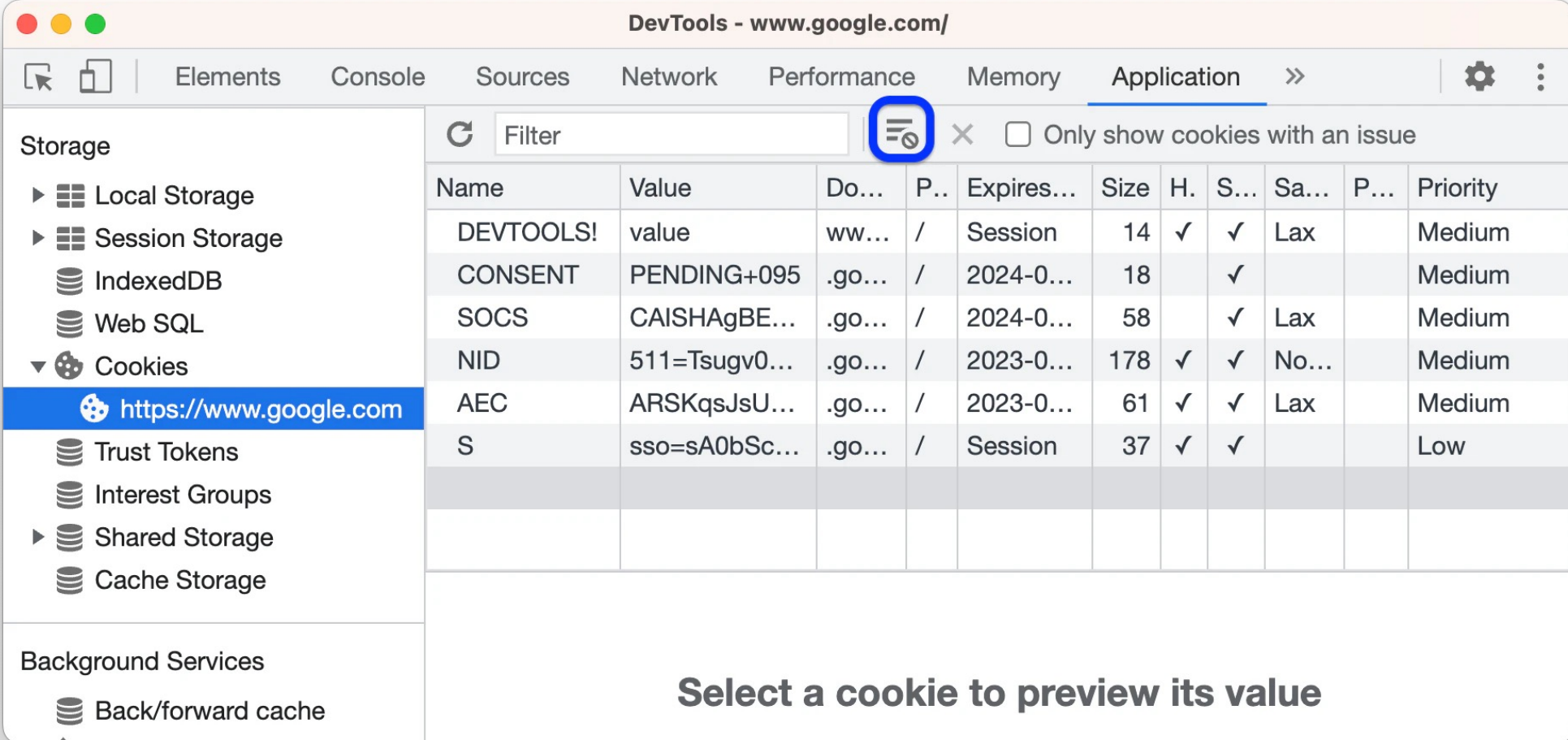
* Connection #0 to host 127.0.0.1 left intact

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": 3600000000000, "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2Njg5NzUwNz
```

Вкладка Application. Cookie



DevTools - www.google.com/


Elements Console Sources Network Performance Memory Application >> ⚙️ ⋮

Storage

- ▶ Local Storage
- ▶ Session Storage
- IndexedDB
- Web SQL
- ▼ Cookies
 - https://www.google.com**
 - Trust Tokens
 - Interest Groups
 - ▶ Shared Storage
 - Cache Storage

Background Services

- Back/forward cache

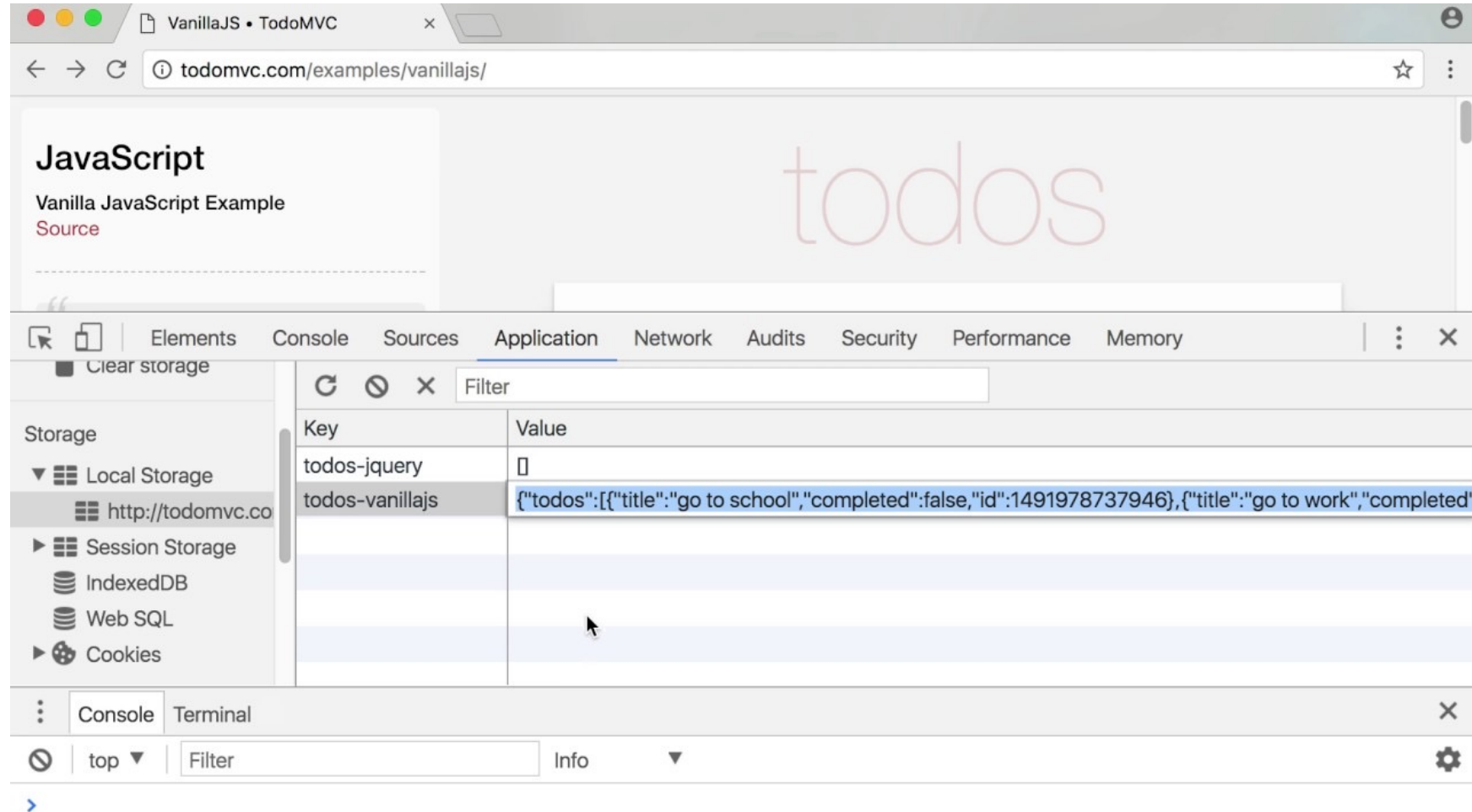
Filter  × ☐ Only show cookies with an issue

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

Select a cookie to preview its value

Вкладка Application. Local Storage

- В варианте с JWT мы используем Local Storage (локальное хранилище) 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, Twitter, 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

SSO

- **Технология единого входа** (Single Sign-On) — технология, при использовании которой пользователь переходит из одного раздела портала в другой, либо из одной системы в другую, не связанную с первой системой, без повторной аутентификации.

Вход



Используйте аккаунт

Телефон или e-mail

Пароль



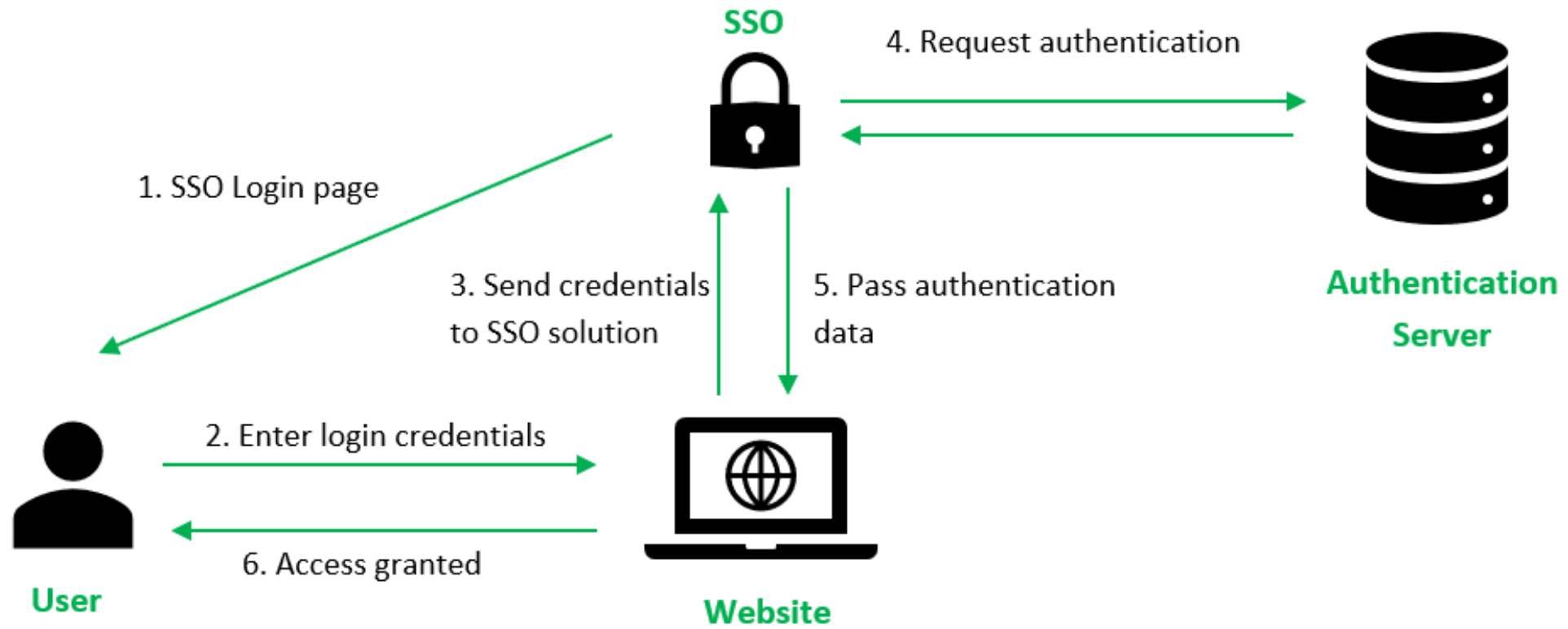
Запомнить меня

Войти в личный кабинет

[Регистрация](#)

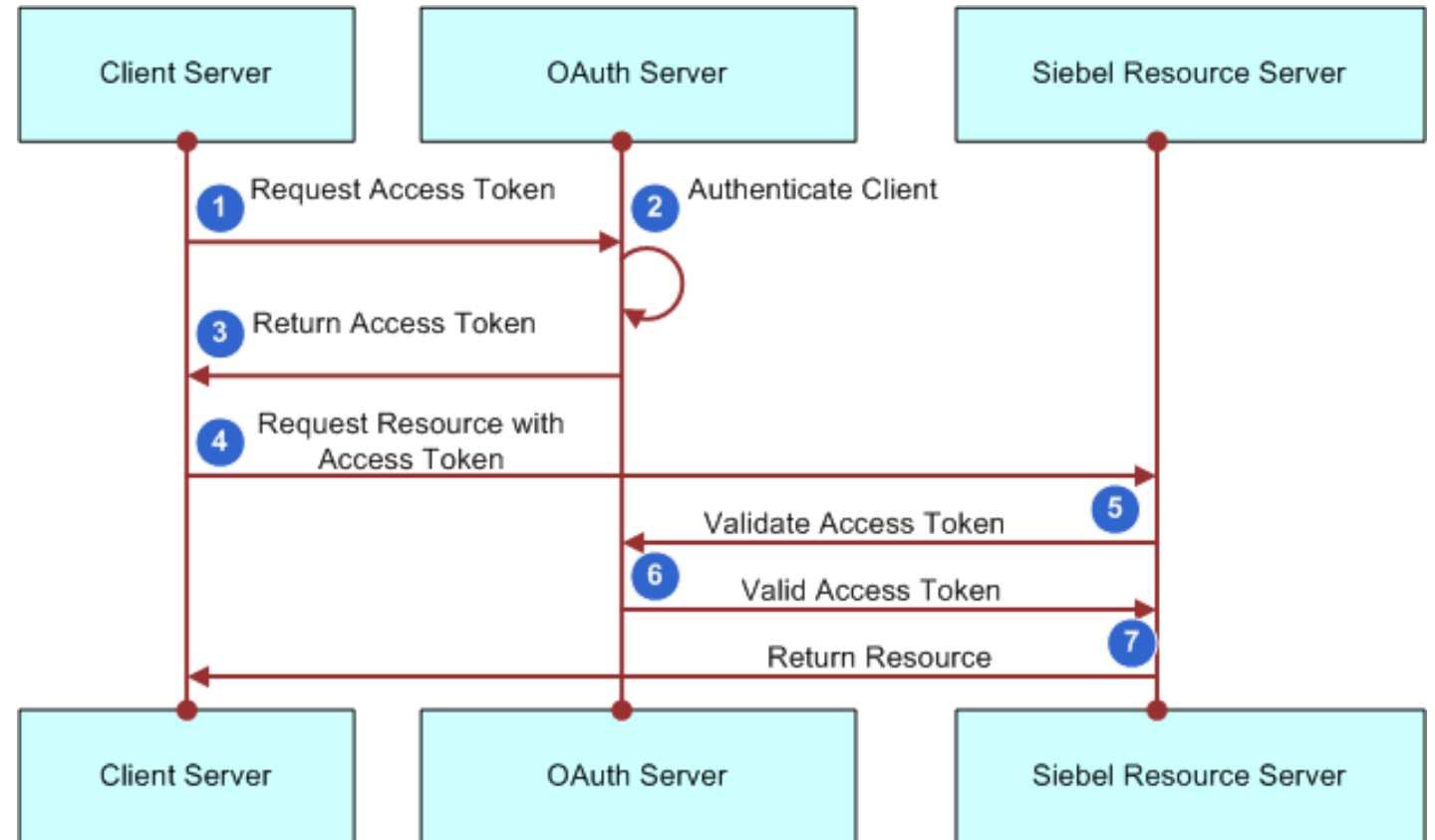
[Восстановление пароля](#)

SSO



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.

