Лекция 12
JWT. SSO

Разработка интернет приложений

Канев Антон Игоревич

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.

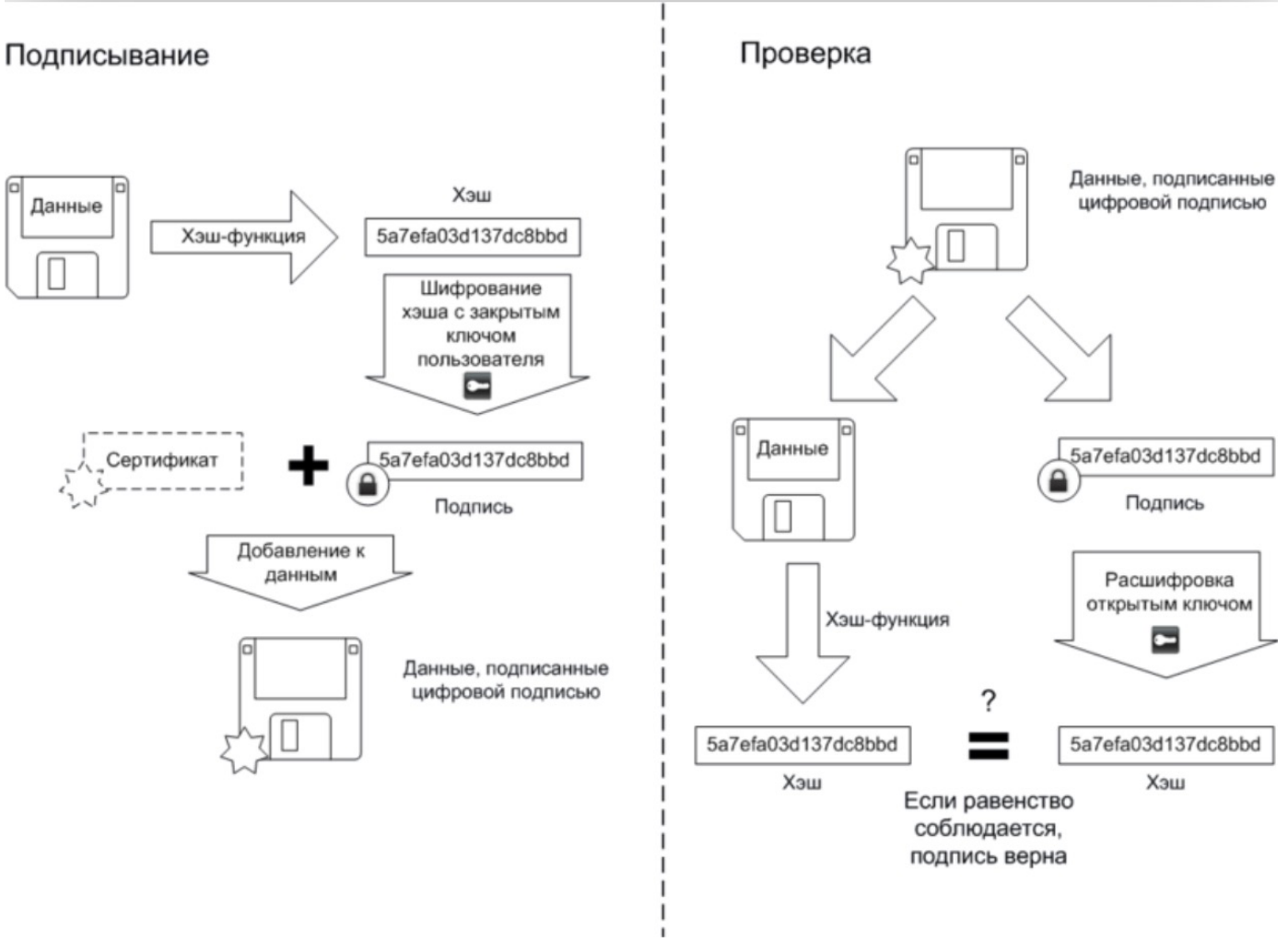
Авторизация

- **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.

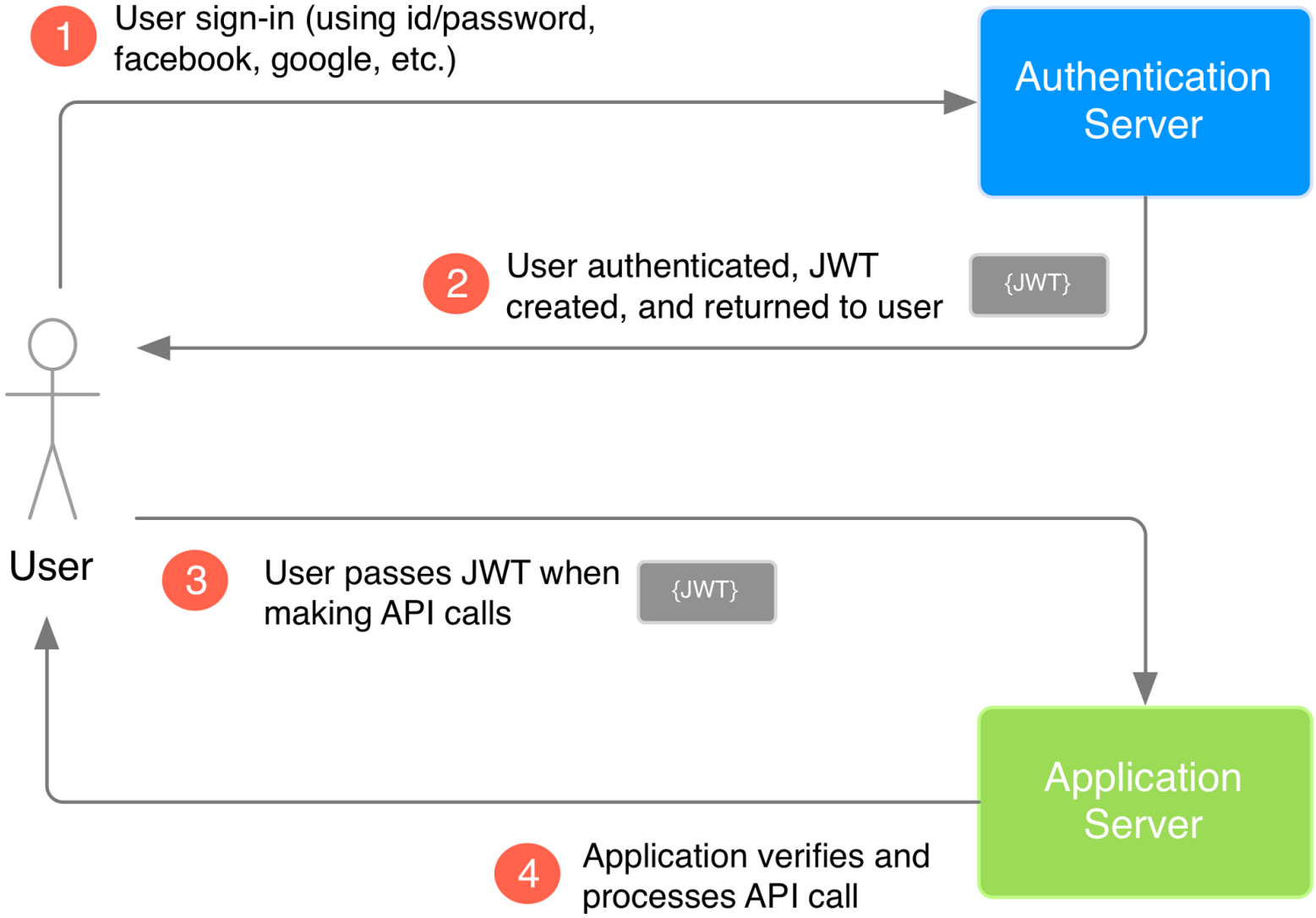
Цифровая подпись

- **Электро́нная цифровая по́дпись** 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.



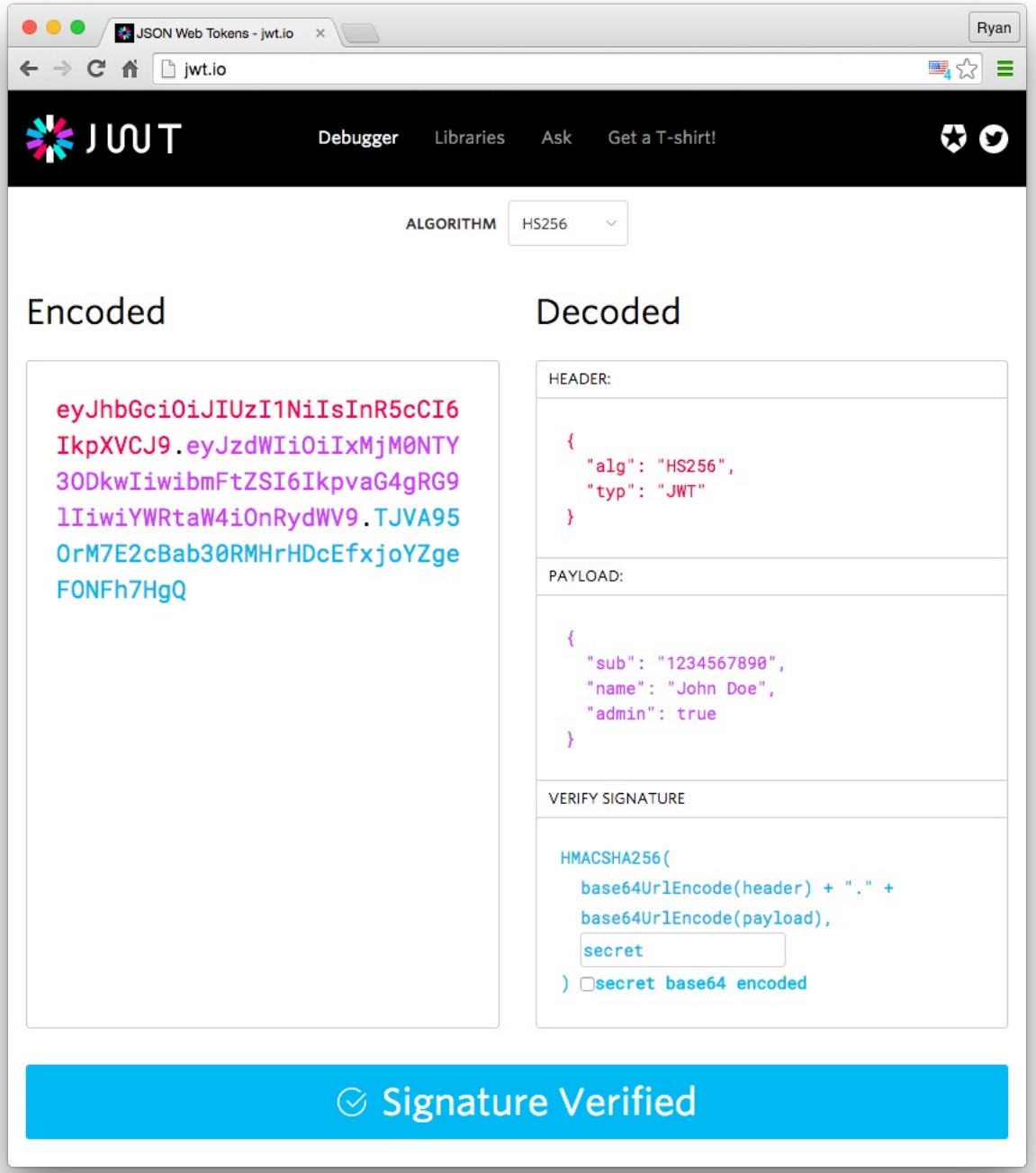
JWT

- **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.

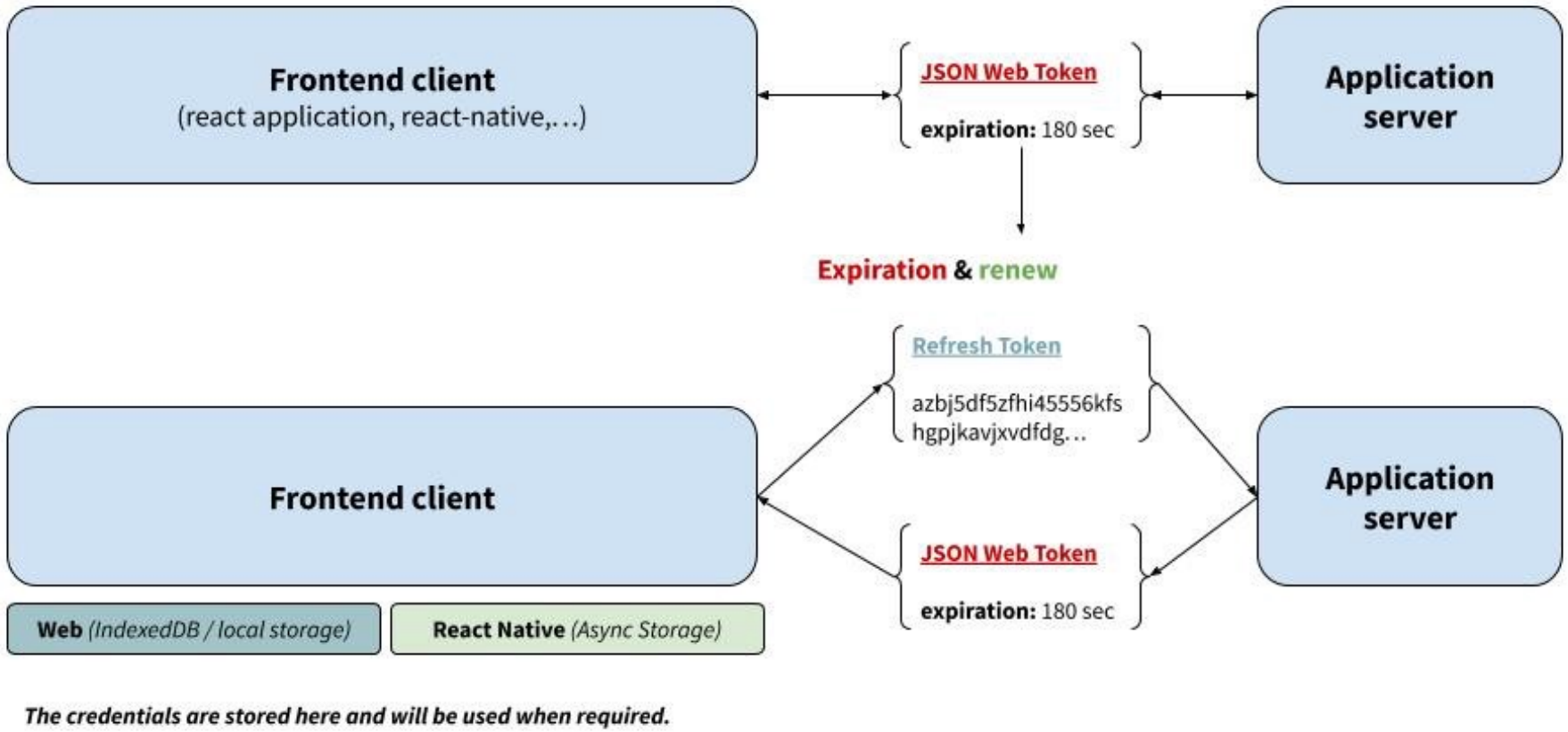


JWT

- 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 («.»).



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

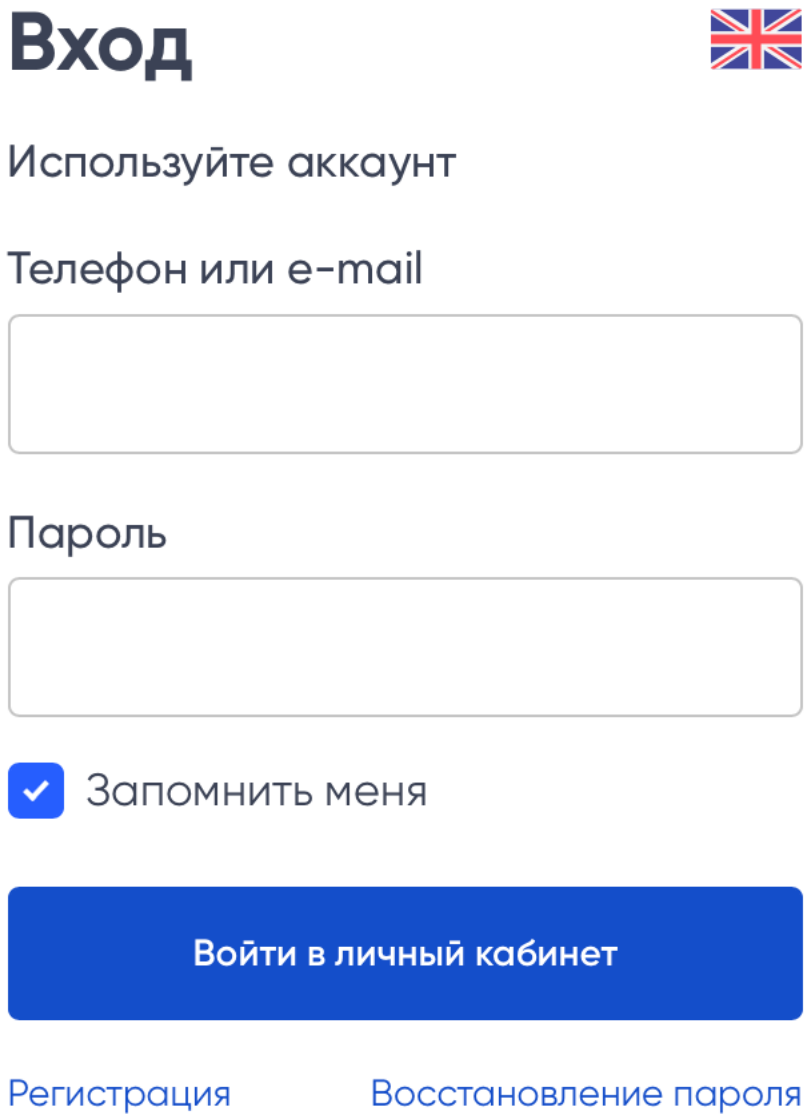
```
$ curl --location --request POST 'http://127.0.0.1:8080/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "login": "login",
    "password": "check123"
}'
{"expires_in":3600000000000,"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2Njg5NzUwNz
```

Проблемы

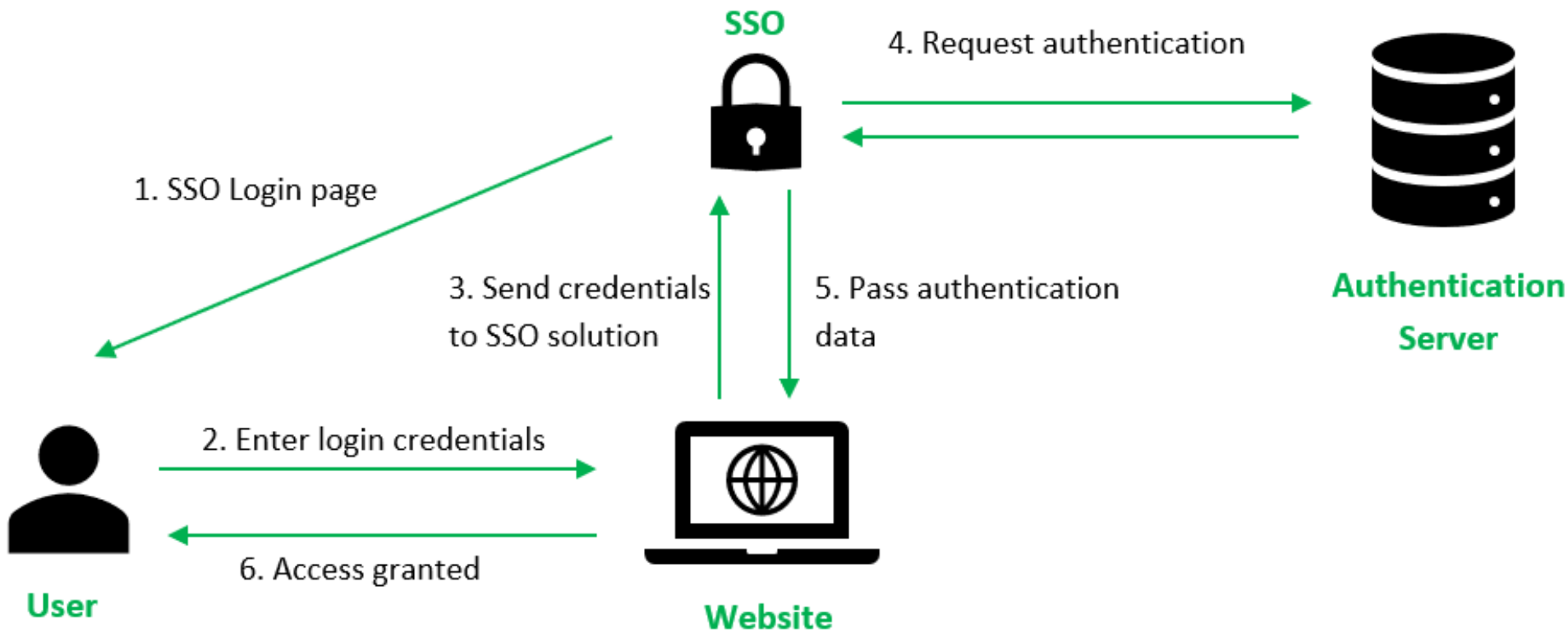
- Одной из проблем аутентификации и информационной безопасности является то, что пользователи, как правило, используют несколько различных сервисов (например, на Google, Twitter, Apple и др.), и, соответственно, несколько учётных записей со своими логинами и паролями.
- Таким образом пользователям требуется 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.

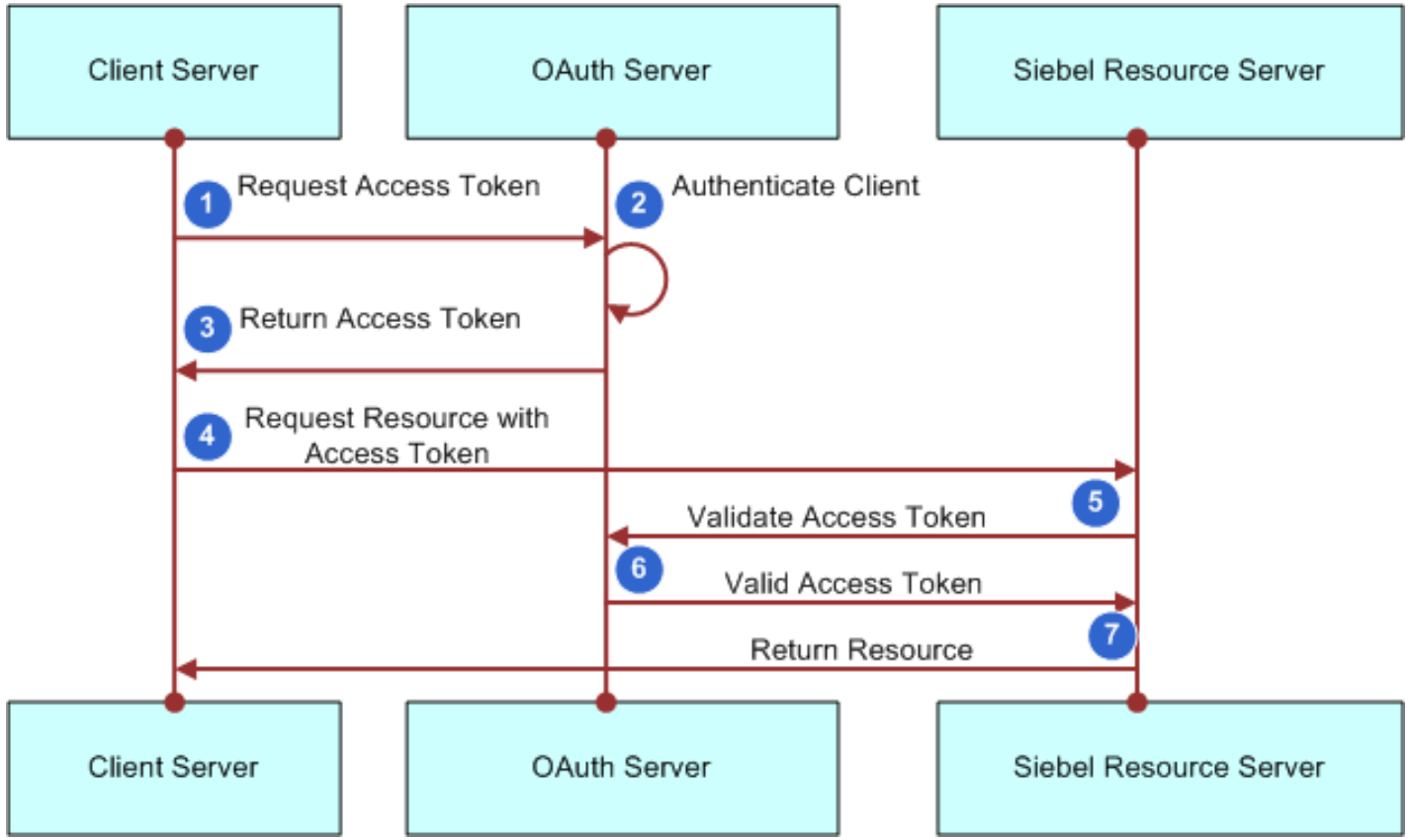


SSO


SSO
4. Request authentication
1. SSO Login page
3. Send credentials to SSO solution
5. Pass authentication data
Authentication Server
2. Enter login credentials
6. Access granted
User
Website

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.