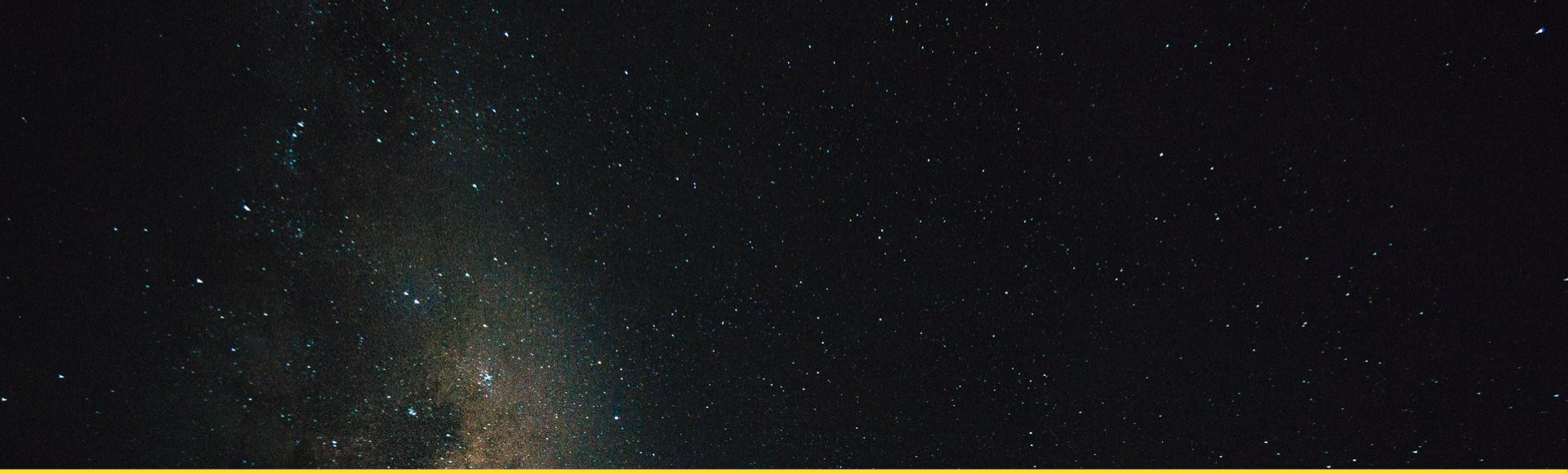


Вебсокеты и шаблоны

Tinkoff.ru

План занятия

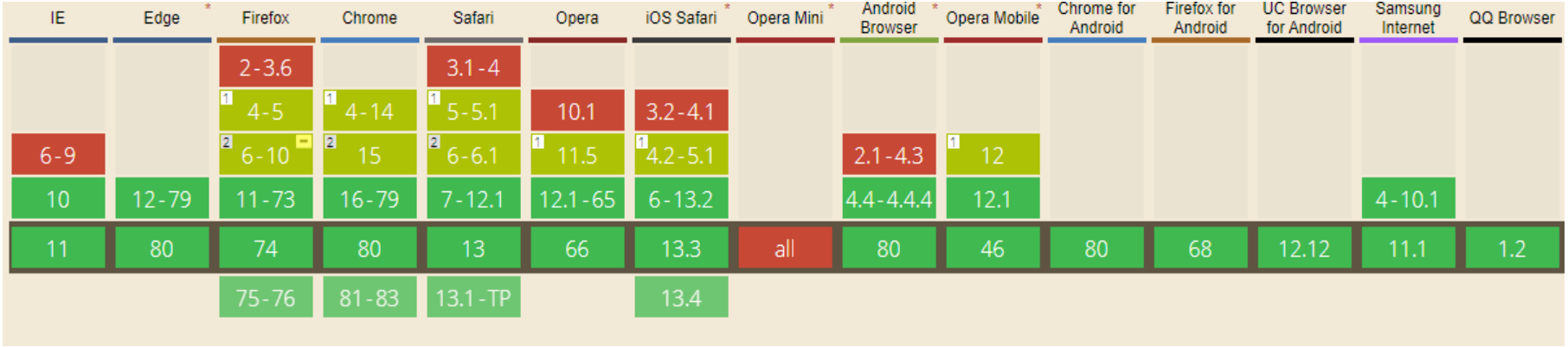
1. Вебсокеты
2. Шаблоны
3. CORS
4. XSS
5. CSRF
6. XSS
7. Пишем чат

Вебсокеты

Вебсокет

Вебсокет – протокол, позволяющий двунаправленный обмен между клиентом и сервером

Стандарт: https://tools.ietf.org/html/rfc6455



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser
		2-3.6		3.1-4										
		1 4-5	1 4-14	1 5-5.1	10.1	3.2-4.1								
6-9		2 6-10	2 15	2 6-6.1	1 11.5	1 4.2-5.1		2.1-4.3	1 12					
10	12-79	11-73	16-79	7-12.1	12.1-65	6-13.2		4.4-4.4.4	12.1				4-10.1	
11	80	74	80	13	66	13.3	all	80	46	80	68	12.12	11.1	1.2
		75-76	81-83	13.1-TP		13.4								

Вебсокет похож на HTTP

Запрос клиента

```
GET /stocks HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: example.com
Origin: http://example.com
WebSocket-Protocol: sample
```

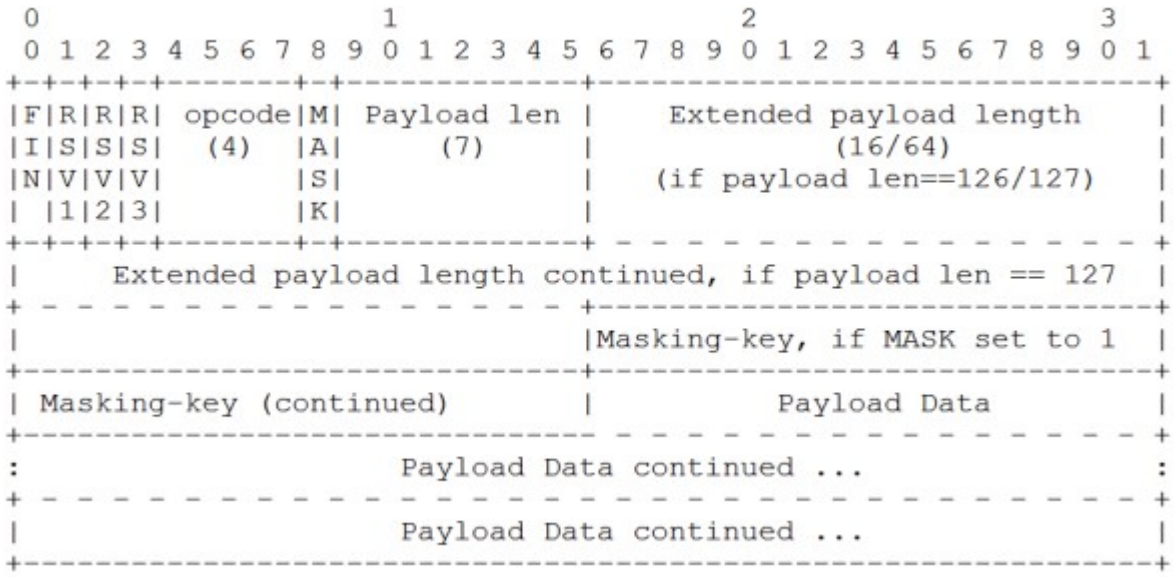
<- Запрос на переключение на WebSocket

Ответ сервера

```
HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://example.com
WebSocket-Location: ws://example.com/stocks
WebSocket-Protocol: sample
```

<- Ответ о возможности переключения

Бинарное сообщение



Типы сообщений в WebSocket

- Ping/Pong сообщение – проверка состояния узлов
- Text/Binary – «полезные» сообщения
- Close сообщения – закрываем соединение
- Мета сообщения

golang.org/x/net/websocket

1) Выполняем команду `go get golang.org/x/net/websocket`

```
1  func WSHandler(ws *websocket.Conn) {
2     fmt.Println("web socket handler function")
3  }
4  func main() {
5     http.Handle("/stocks", websocket.Handler(WSHandler))
6     err := http.ListenAndServe(":5000", nil)
7     if err != nil {
8        log.Fatal(err)
9     }
10 }
```

This package currently lacks some features found in an alternative and more actively maintained WebSocket package:

`https://godoc.org/github.com/gorilla/websocket`

Объект Message

Прочитать данные из сокета

```
// receive text frame
var message string
websocket.Message.Receive(ws, &message)
// receive binary frame
var data []byte
websocket.Message.Receive(ws, &data)
```

Записать данные в сокет

```
// send text frame
message := "hello"
websocket.Message.Send(ws, message)
// send binary frame
data := []byte{0, 1, 2}
websocket.Message.Send(ws, data)
```

Пример сервера

```
1  func Quotes(ws *websocket.Conn) {
2     for n := 0; n < 10; n++ {
3        msg := "price " + string(n+48)
4        fmt.Println("sending to client: " + msg)
5        err := websocket.Message.Send(ws, msg)
6        if err != nil {
7           fmt.Printf("can't send: %s\n", err)
8           break
9        }
10       var reply string
11       err = websocket.Message.Receive(ws, &reply)
12       if err != nil {
13          fmt.Printf("can't receive: %s\n", err)
14          break
15       }
16       fmt.Printf("received from client: %s\n", reply)
17    }
18 }
```

Пример клиента

```
1  conn, err := websocket.Dial(url, "", origin)
2  if err != nil {
3     log.Fatalf("can't connect to server: %s", err)
4  }
5  var msg string
6  for {
7     err := websocket.Message.Receive(conn, &msg)
8     if err != nil {
9        if err == io.EOF {
10          break
11       }
12       fmt.Printf("can't receive: %s\n", err)
13    }
14    fmt.Printf("received from server: %s\n", msg)
15    err = websocket.Message.Send(conn, msg)
16    if err != nil {
17       fmt.Printf("can't send msg: %s", err)
18    }
19 }
```

Обмен JSON объектами

Прочитать объект из сокета

```
// receive json object
var person Person
websocket.JSON.Receive(ws, &person)
```

Записать объект в сокет

```
person := Person{
    Name: "Jan",
    Emails: []string{"ja@newmarch.name", "jan.newmarch@gmail.com"},
}
websocket.JSON.Send(conn, person)
```

Почему Gorilla ?

Gorilla WebSocket compared with other packages

	github.com/gorilla	golang.org/x/net
RFC 6455 Features		
Passes Autobahn Test Suite	Yes	No
Receive fragmented message	Yes	No, see note 1
Send close message	Yes	No
Send pings and receive pongs	Yes	No
Get the type of a received data message	Yes	Yes, see note 2
Other Features		
Compression Extensions	Experimental	No
Read message using io.Reader	Yes	No, see note 3
Write message using io.WriteCloser	Yes	No, see note 3

Пример сервера gorilla/websocket

```
1  func Handler(w http.ResponseWriter, r *http.Request) {
2     conn, err := upgrader.Upgrade(w, r, nil)
3     if err != nil {
4        fmt.Printf("can't upgrade connection: %s\n", err)
5        return
6     }
7
8     for n := 0; n < 10; n++ {
9        msg := "hello  " + string(n+48)
10       fmt.Printf("sending to client: %s\n", msg)
11       err = conn.WriteMessage(websocket.TextMessage, []byte(msg))
12       _, reply, err := conn.ReadMessage()
13       if err != nil {
14          fmt.Printf("can't receive: %s\n", err)
15       }
16       fmt.Printf("received back from client: %s\n", string(reply[:]))
17    }
18    conn.Close()
19 }
```

Handlers

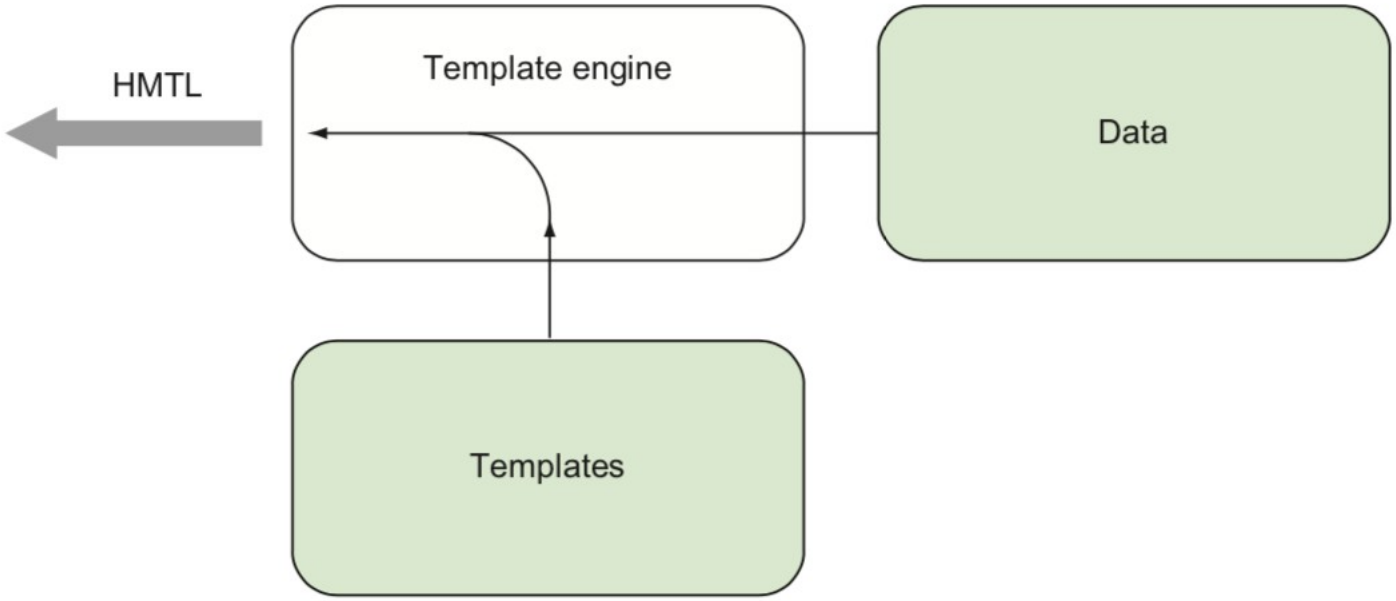
```
1  deadline := time.Now().Add(time.Second*1)
2  conn.SetPingHandler(func (appData string) error {
3       err := conn.WriteControl(websocket.PongMessage, []byte(appData),
4  deadline)
5      if err == websocket.ErrCloseSent {
6            return nil
7      } else if e, ok := err.(net.Error); ok && e.Temporary() {
8            return nil
9      }
10     return err
11  })
12
13
```

Пример клиента gorilla/websocket

```
1  url := "ws://localhost:5000"
2  header := make(http.Header)
3  header.Add("Origin", "http://localhost:5000")
4  conn, _, err := websocket.DefaultDialer.Dial(url, header)
5  for {
6     _, reply, err := conn.ReadMessage()
7     if err != nil {
8        if err == io.EOF {
9           break
10       }
11       if websocket.IsCloseError(err, websocket.CloseAbnormalClosure) {
12          break
13       }
14       fmt.Println("can't receive: %s", err)
15       break
16    }
17    fmt.Printf("received from server: %s\n", string(reply[:]))
18    err = conn.WriteMessage(websocket.TextMessage, reply)
19 }
```

Шаблоны

Шаблоны основная идея



Шаблоны в go

text/template – предназначены для генерации текста

html/template – предназначены для генерации html-контента

Главное отличие – html/template позволяет создавать html-контент защищенный от ряда атак (например XSS)

Наш первый шаблон

```
1  type Candle struct {
2     Name        string
3     O           float64
4     C           float64
5     H           float64
6     L           float64
7  }
8  const tmpl = `Свеча #{{.Name}} - Цена: O: {{.O}} C: {{.C}}`
9  func main() {
10    aapl := Candle{"AAPL", 56.56, 50.5}
11    t := template.New("aapl")
12    t, err := t.Parse(tmpl)
13    if err != nil {
14       log.Fatal("can't parse: ", err)
15    }
16    if err := t.Execute(os.Stdout, aapl); err != nil {
17       log.Fatal("can't execute: ", err)
18    }
19 }
```

```
Свеча #AAPL -  Цена: O: 56.56, C: 50.5
```

Переменные

```
1  const tmpl = `{{$a := 100.0}}{{if eq $a .O}} O is 100{{end}}`
2  func main() {
3     aapl := Candle{"AAPL", 100.0, 50.5}
4     t := template.New("aapl")
5     t, err := t.Parse(tmpl)
6     if err != nil {
7        log.Fatal("can't parse: ", err)
8     }
9     if err := t.Execute(os.Stdout, aapl); err != nil {
10       log.Fatal("can't execute: ", err)
11    }
12 }
```

```
O is 100
```

Пайплайн

```
1  const tmpl = `{{$a := 100.0}}{{eq $a .O|printf "%v"|printf "%s"}} O is 100`
2  func main() {
3     aapl := Candle{"AAPL", 100.0, 50.5}
4     t := template.New("aapl")
5     t, err := t.Parse(tmpl)
6     if err != nil {
7        log.Fatal("can't parse: ", err)
8     }
9     if err := t.Execute(os.Stdout, aapl); err != nil {
10       log.Fatal("can't execute: ", err)
11    }
12 }
```

```
true O is 100
```

html/template – базовый шаблон

```
{{define "base"}}
<html>
<head>{{template "head" .}}</head>
<body>{{template "body" .}}</body>
</html>
{{end}}
```

html/template шаблон для индекс

```
{{define "head"}}<title>Свечи</title>{{end}}
{{define "body"}}
…
        {{range $key,$value := . }}
        <tr>
            <td>{{$value.Name}}</td>
            <td>{{$value.O}}</td>
            <td>{{$value.H}}</td>
            <td>{{$value.L}}</td>
            <td>{{$value.C}}</td>
            <td>
                <a href="market/candles/{{$key}}/buy">Купить</a>
            </td>
        </tr>
        {{end}}
    </table>
</div>
{{end}}
```

Настравиваем роуты

```
r := chi.NewRouter()

r.Route("/market", func(r chi.Router) {
   r.Get("/", GetCandles)
   r.Get("/candles/{id}/buy", GetBuyCandle)
   r.Post("/candles/{id}/buy", BuyCandle)
})

if err := http.ListenAndServe(":5000", r); err != nil {
   log.Fatal(err)
}
```

Инит шаблонов и определяем handlerFunc

```
1  var templates map[string]*template.Template
2  func init() {
3     templates["index"] = template.Must(template.ParseFiles("html/index.html",
   "html/base.html"))
4     templates["buy"] = template.Must(template.ParseFiles("html/buy.html",
   "html/base.html"))
5  }
6  func renderTemplate(w http.ResponseWriter, name string, template string,
   viewModel interface{}) {
7     tmpl, ok := templates[name]
8     if !ok {
9        http.Error(w, "can't find template", http.StatusInternalServerError)
10    }
11    err := tmpl.ExecuteTemplate(w, template, viewModel)
12 }
13 func GetLots(w http.ResponseWriter, r *http.Request) {
14    renderTemplate(w, "index", "base", candles)
15 }
```

Пример


localhost:5000/market/
Свечи
Название
Цена открытия
Максимальная цена
Минимальная цена
Цена закрытия
AAPL 100 100 100 100 Выставить цену
YNDX 150 150 150 150 Выставить цену
AMZN 80 80 80 80 Выставить цену

CORS

CORS

Cross-origin resource sharing – технология, которая позволяет предоставить веб-странице доступ к ресурсам другого домена.

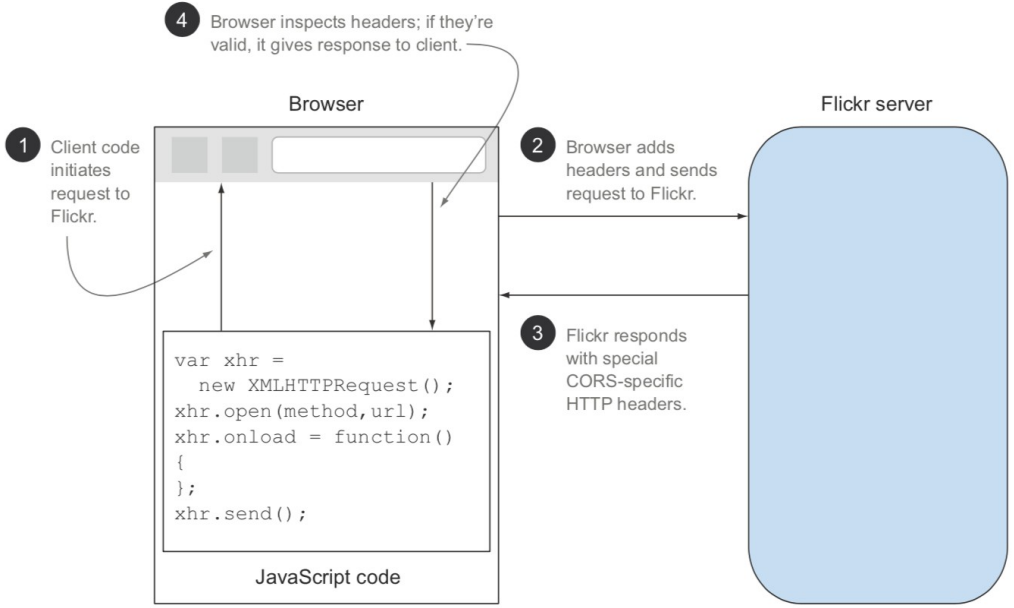


Figure 1.1 End-to-end CORS request flow

Жизненный цикл CORS-запросов

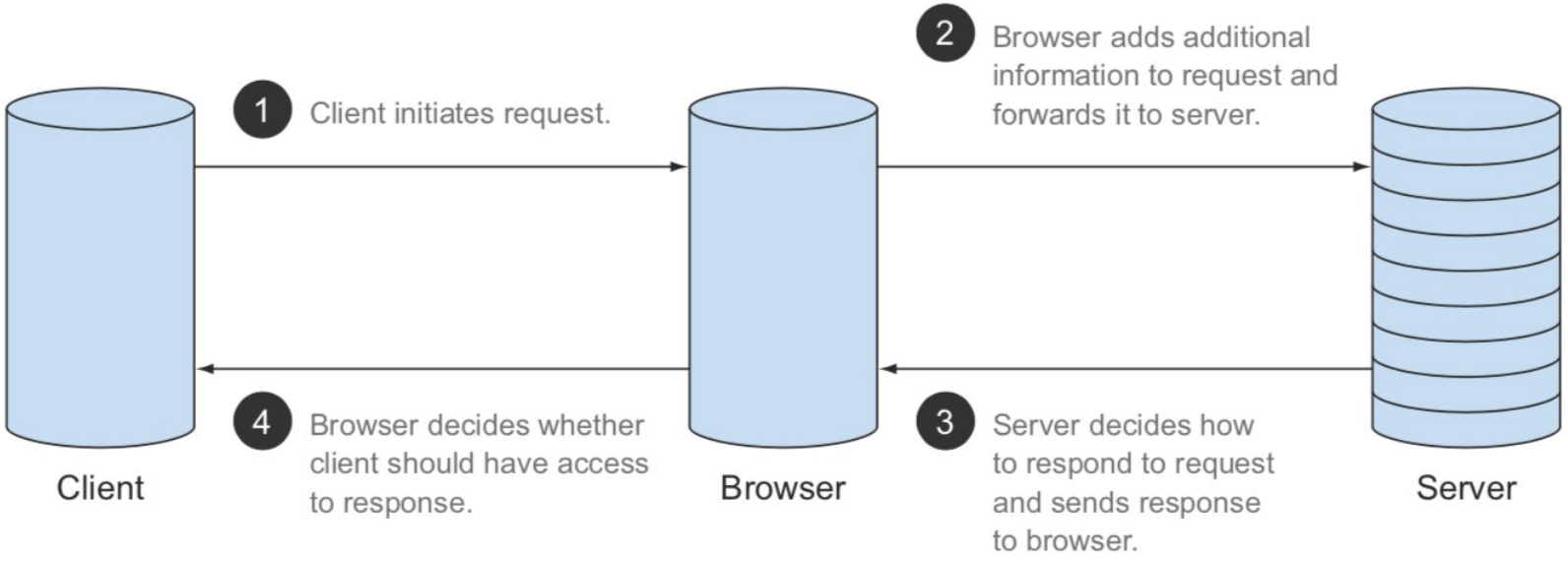


Figure 3.12 Lifecycle of a CORS request

Что такое Origin?

Origin – это http-заголовок, который определяет нахождение клиентского ресурса, его передает браузер бэкенду.

URL	Origin
http://localhost:1111	http://localhost:1111
http://localhost:1111/client.html	http://localhost:1111
https://localhost:1111/client.html	https://localhost:1111
http://localhost/client.html	http://localhost
file:///Users/hossain/ch02/client.html	`null`

Что такое Access-Control-Allow-Origin?

Access-Control-Allow-Origin – это http-заголовок, который указывает браузеру разрешение на доступ клиенту к контенту. Проставляется сервером.

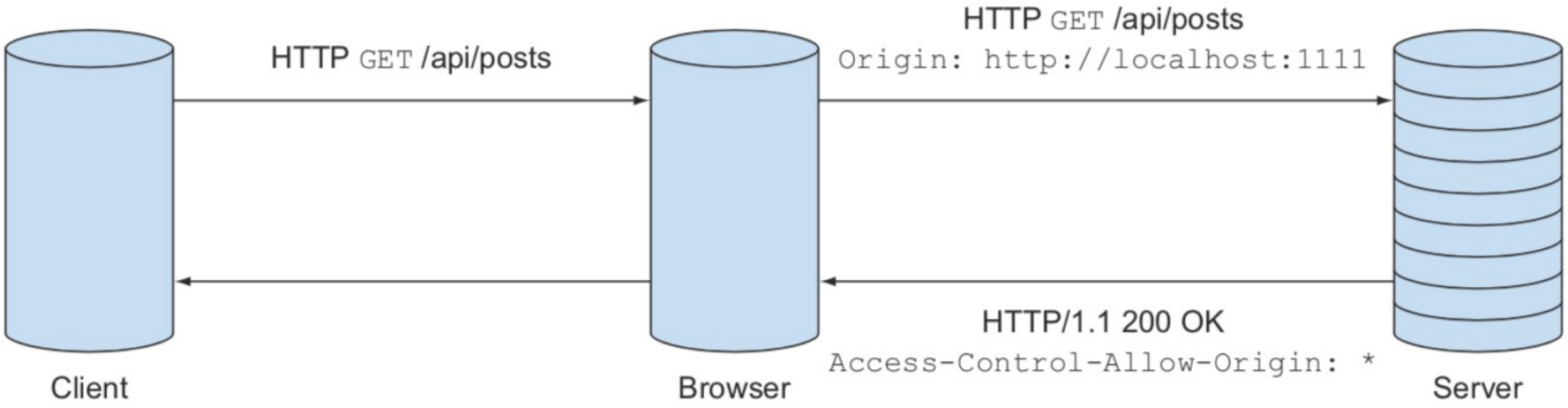
Например:

Access-Control-Allow-Origin: * <- разрешено всем

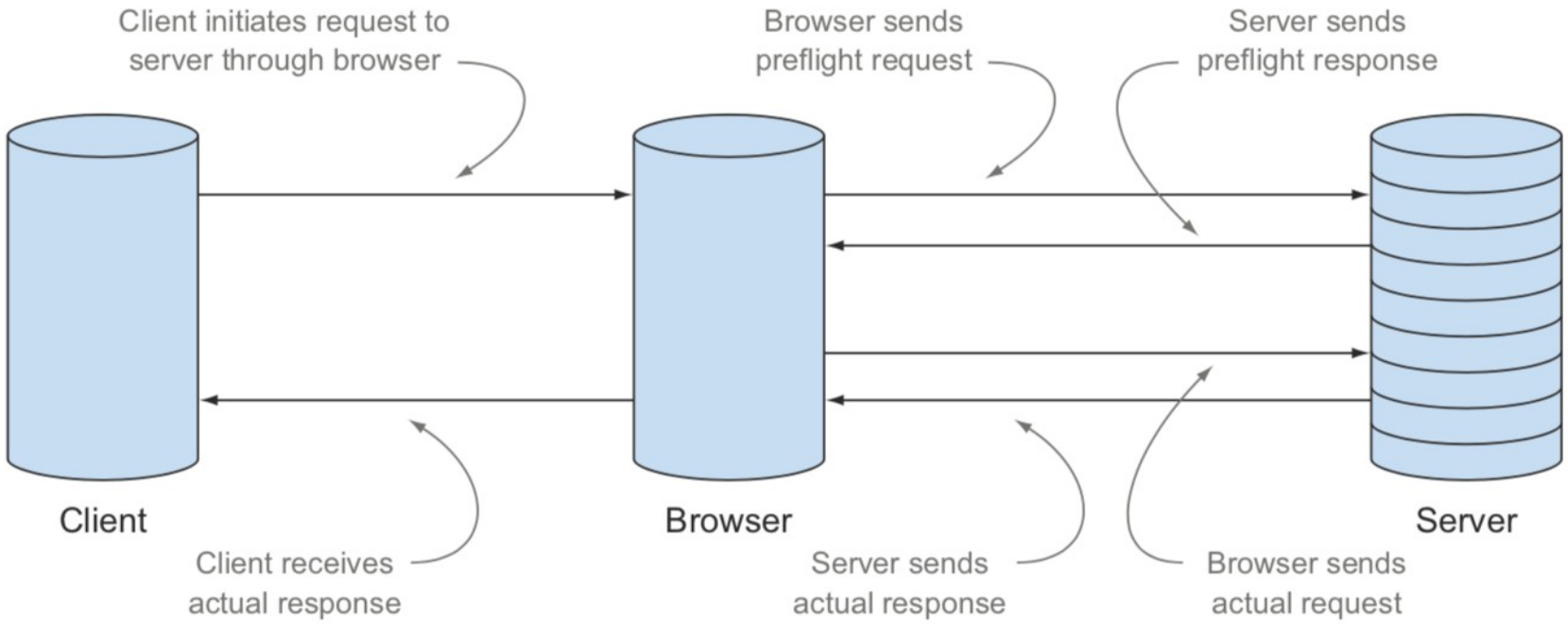
Access-Control-Allow-Origin: https://tinkoff.ru <- разрешено tinkoff.ru

Access-Control-Allow-Origin: null <- запрещено

CORS (метод GET)



Preflight request (Обратная совместимость)



OPTIONS == Preflight Request

Preflight request:

```
OPTIONS /api/posts HTTP/1.1
User-Agent: Chrome
Host: 127.0.0.1:9999
Accept: */*
Origin: http://localhost:1111
Access-Control-Request-Method: GET
Access-Control-Request-Headers: Timezone-Offset, Sample-Source
```

Preflight response:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: http://localhost:1111
Access-Control-Allow-Methods: GET, DELETE
Access-Control-Allow-Headers: Timezone-Offset, Sample-Source
```

Actual request:

```
GET /api/posts HTTP/1.1
User-Agent: Chrome
Host: 127.0.0.1:9999
Accept: */*
Origin: http://localhost:1111
Timezone-Offset: 300
Sample-Source: Cors in Action
```

Не требуют Preflight

HTTP-методы, которые не требуют Preflight запроса

- GET
- HEAD
- POST

Но при условии, что используются только «простые» http-заголовки: Accept, Accept-Language, Content-Language, Content-Type (только определенные значения! application/json в него не входит)

CSRF

Cross-site request forgery

CSRF – вид атаки, когда злоумышленник пытается выполнить тайно запрос на другой сервер (например неавторизованный сайт делает запрос с вашими правами (cookie) на сайт twitter.com).

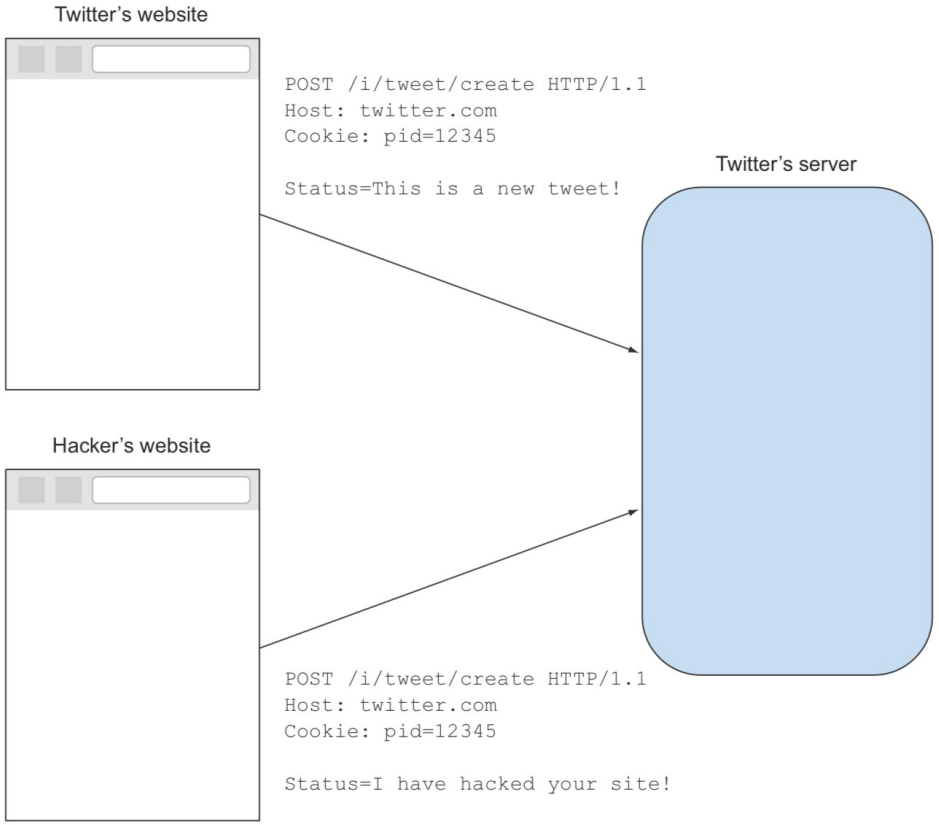


Figure C.1 CSRF exists because cookies are always included on requests, regardless of where the request comes from Luckily Twitter protects itself from CSRF with an authenticity_token..

CSRF-токен

Figure C.3 How Twitter uses the `authenticity_token` field to guard against CSRF

XSS

Cross-site scripting

XSS – вид атаки, когда злоумышленник пытается выполнить запрос на другой сервер с машины клиента путем внедрения вредоносного кода в выдаваемую клиенту страницу.

Этого можно добиться, когда вредоносный код попадает на сервер (например в базу данных) и возвращается пользователям интернет-ресурса

Санитаризация:

https://rawgit.com/mikesamuel/sanitized-jquery-templates/trunk/safetemplate.html#problem_definition

Пример

```
1  const tmpl = `Лот #{{.ID}} - Описание: {{.Title}}, Цена: {{.StartPrice}}`
2  const ex = `
3  <script>
4     window.location='http://attacker/'
5  </script>
6  `
7  func main() {
8     note := Lot{1, ex, 100}
9     t := template.New("note")
10    t, err := t.Parse(tmpl)
11    if err != nil {
12       log.Fatal("can't parse: ", err)
13       return
14    }
15    if err := t.Execute(os.Stdout, note); err != nil {
16       log.Fatal("can't execute: ", err)
17       return
18    }
19 }
```

Сравниваем text vs html templates

text/template

```
Лот #1 - Описание:
<script>
   window.location='http://attacker/'
</script>
, Цена: 100
```

html/template

```
 Лот #1 - Описание:
&lt;script&gt;
   window.location=&#39;http://attacker/&#39;
&lt;/script&gt;
, Цена: 100
```

Пишем чат

Обратная связь

Спасибо за внимание

Tinkoff.ru