WEB-SOCKET, TEMPLATES

Гаратуев Шамиль

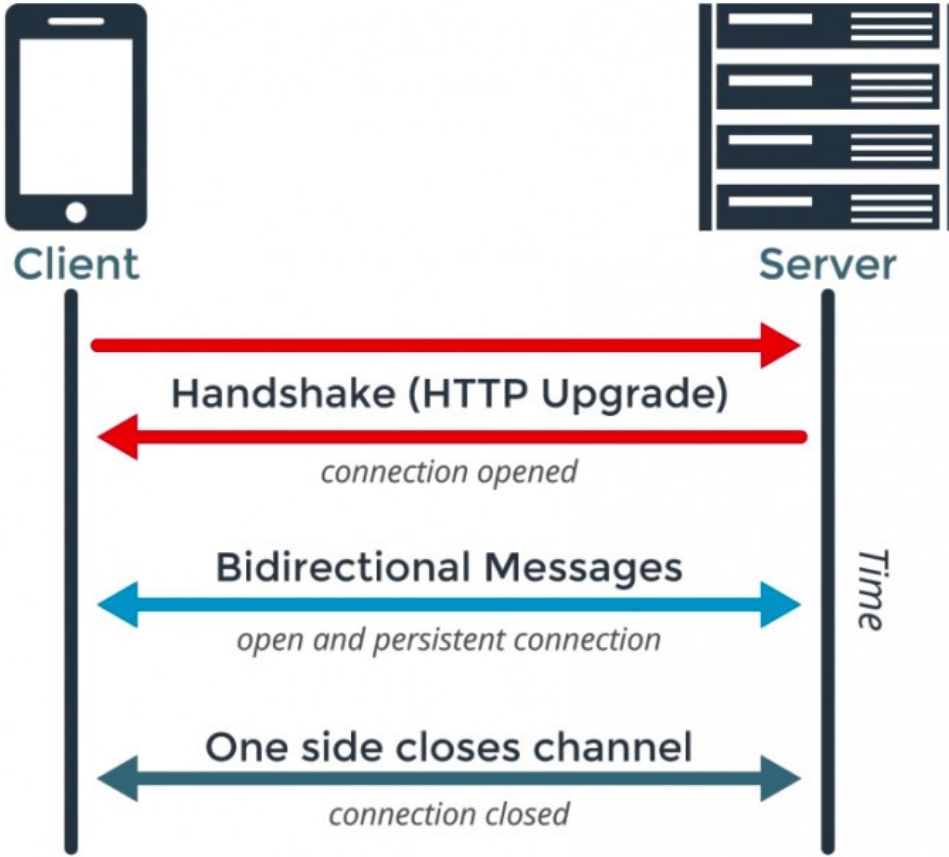
План занятия

- websocket
- templates
- cors, csrf, xss

websocket

протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени

RFC 6455



websocket handshake

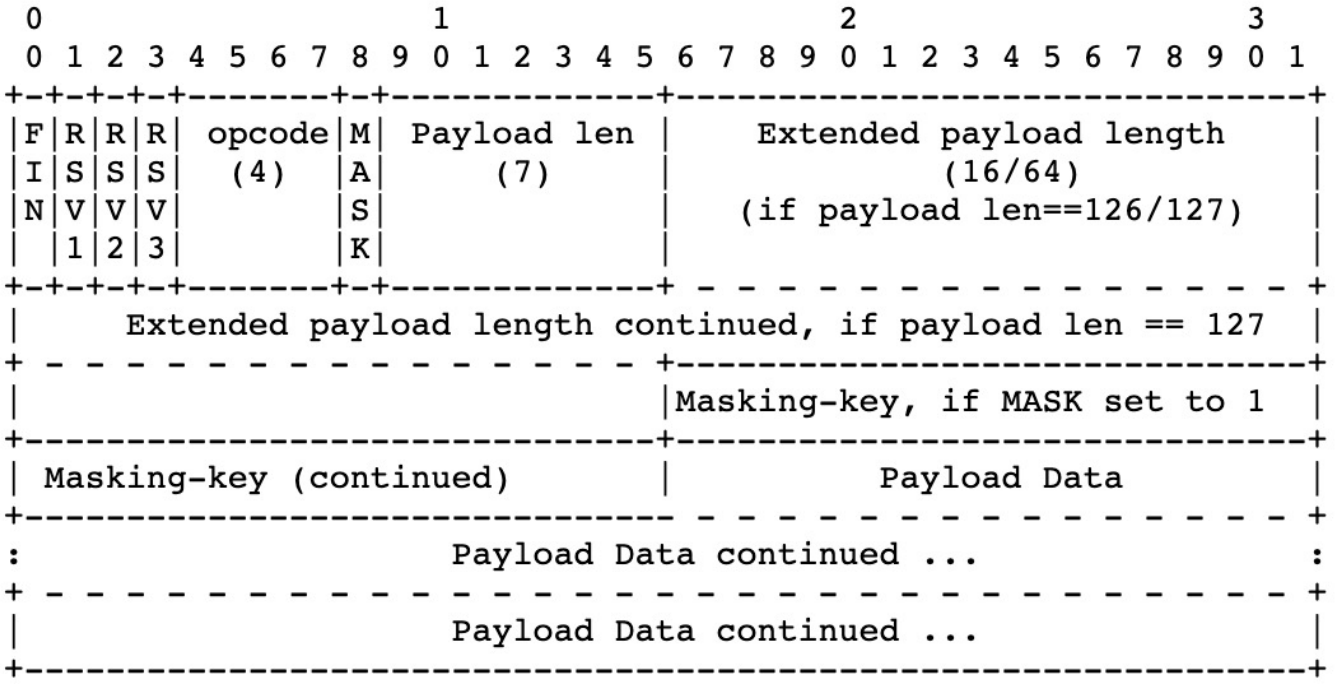
Request:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

websocket frames



websocket opcodes

- %x0 denotes a continuation frame
- %x1 denotes a text frame
- %x2 denotes a binary frame
- %x3-7 are reserved for further non-control frames
- %x8 denotes a connection close
- %x9 denotes a ping
- %xA denotes a pong
- %xB-F are reserved for further control frames

golang.org/x/net/websocket *Conn

```go
import "golang.org/x/net/websocket"

var ws *websocket.Conn

var s string
// works only with string and []byte
err := websocket.Message.Receive(ws, &s)

var b = []byte(s)
// works only with string and []byte
err = websocket.Message.Send(ws, b)

// unmarshals json text
var m  = Message{Text: "welcome"}
err = websocket.JSON.Receive(ws, &m)

// marshals json text
err = websocket.JSON.Send(ws, m)

// *ws.Conn implements io.Reader
_, err = io.ReadAll(ws)

// *ws.Conn implements io.Writer
err = json.NewEncoder(ws).Encode(m)
```

- поддерживается отправка простых текстовых или бинарных сообщений
- поддерживается JSON Codec
- имплементирует io.ReadWriter*

websocket client/server

```go
import "golang.org/x/net/websocket"

// client
websocket.Dial(":5000", "chat", "/")

// server
http.Handle("/stocks", websocket.Handler(WSHandler))
err := http.ListenAndServe(":5000", nil)
if err != nil {
    log.Fatal(err)
}

func WSHandler(ws *websocket.Conn) {...}
```

Недостатки стандартной библиотеки

- client должен сам отправлять/обрабатывать ping/pong
- самостоятельно отправлять/обрабатывать close
- io.ReadWriter имплементированы так себе, потому что работают в рамках frame

https://pkg.go.dev/golang.org/x/net/websocket

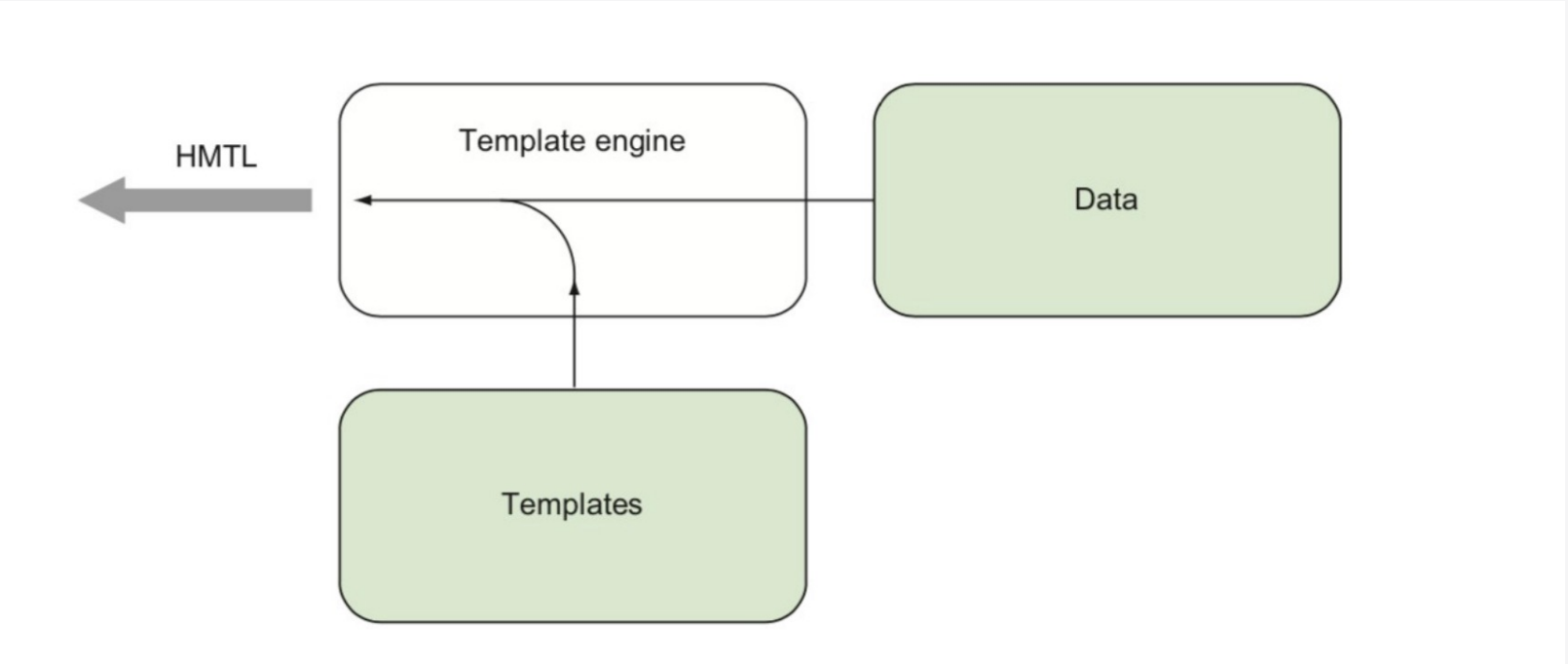
(!) This package currently lacks some features found in alternative and more actively maintained WebSocket packages:

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

https://godoc.org/nhooyr.io/websocket

Templates

Templates



text/template

```
1 import "text/template"
2
3 const easyT = "Hello, {{.Name}}!"
4
5 t := template.Must(template.New("easy").Parse(easyT))
6
7 err := d.Execute(os.Stdout, Person{Name: "Arnold"})
8
9 // result:
10 Hello, Arnold!
```

text/template

Поддерживаются:

- Доступ к данным
- Функции
- Пайплайны
- Переменные
- Условия
- Циклы
- и многое другое…

Документация тут: pkg.go.dev/text/template

ParseFiles creates a new Template and parses the template definitions from the
the base name and parsed contents of the first file. There must be at least one f
*Template is nil.

When parsing multiple files with the same name in different directories, the last
instance, ParseFiles("a/foo", "b/foo") stores "b/foo" as the template named "foo

func ParseGlob

```
func ParseGlob(pattern string) (*Template, error)
```

ParseGlob creates a new Template and parses the template definitions from the
according to the semantics of filepath.Match, and the pattern must match at lea
(base) name and (parsed) contents of the first file matched by the pattern. Pars
of files matched by the pattern.

When parsing multiple files with the same name in different directories, the last

func (*Template) AddParseTree

```
func (t *Template) AddParseTree(name string, tree *parse.Tree) (*Te
```

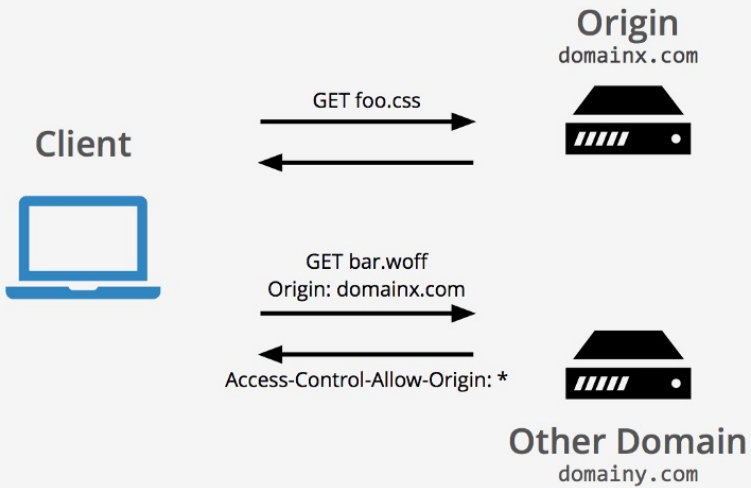
AddParseTree associates the argument parse tree with the template t, giving it t
defined, this tree becomes its definition. If it has been defined and already has t
otherwise a new template is created, defined, and returned.

html/template

```
1 tmpl, err :=template.ParseFiles("users.tmpl")
2 if err != nil {
3     panic(err)
4 }
5
6 http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
7     tmpl.Execute(w, data)
8 })
```

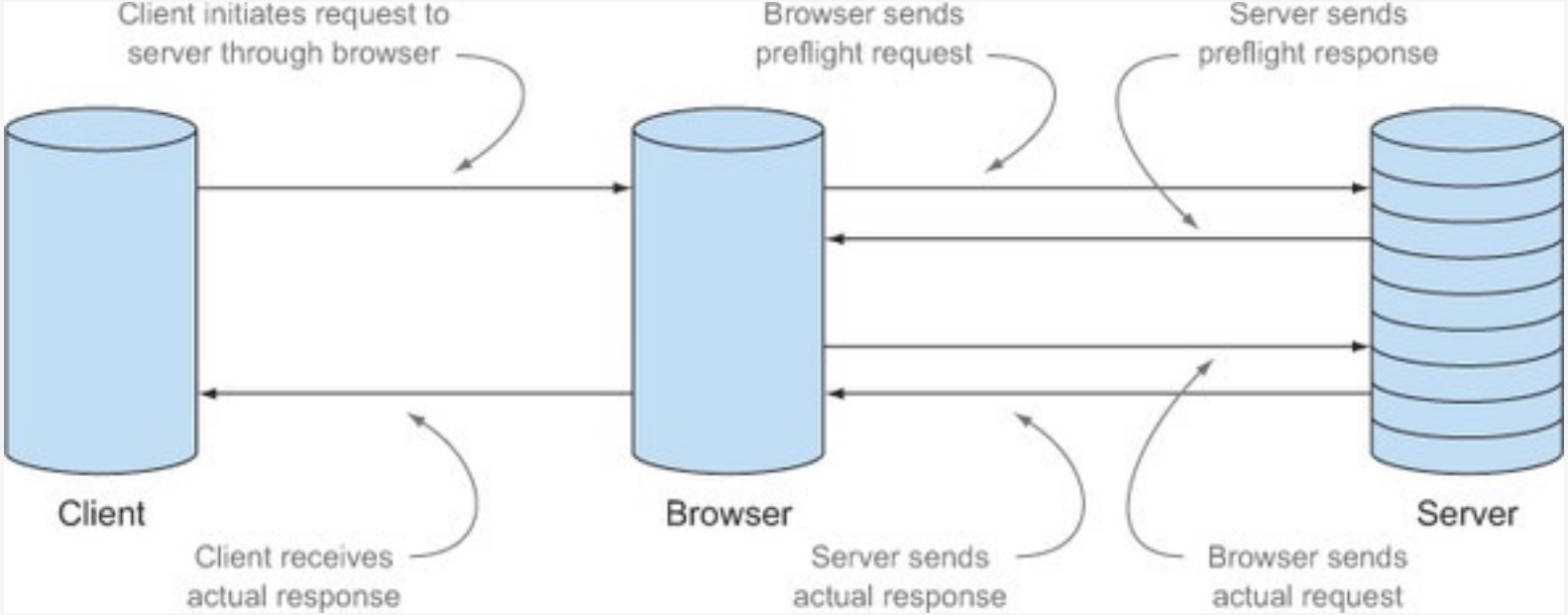
- корректно экранирует html-символы
- можно хостить статику (server-side render, wow!)

cross-origin resource sharing (CORS)



CORS

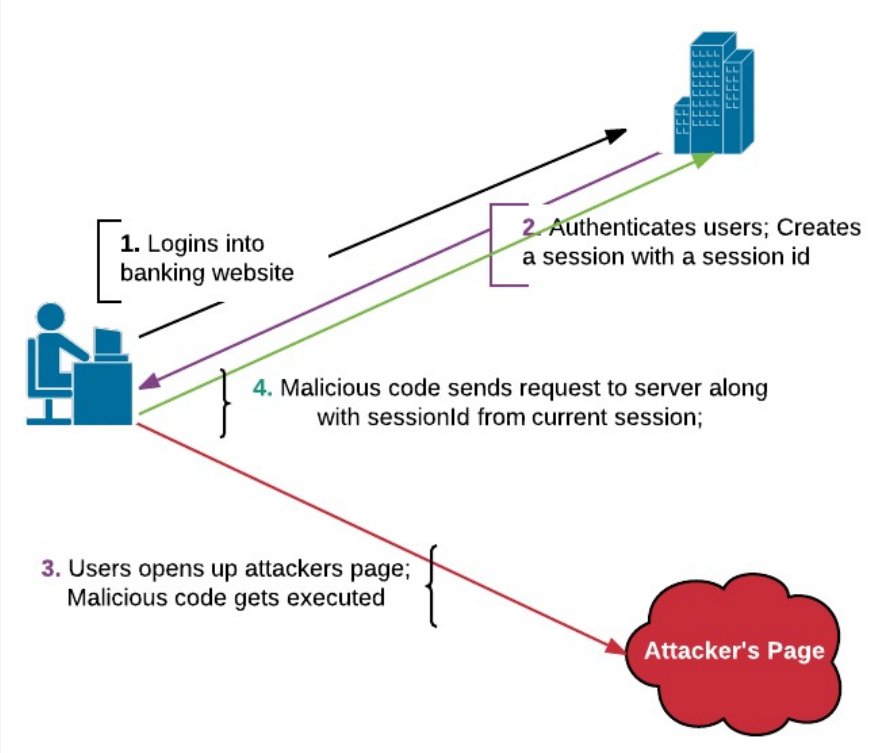
CORS


Client initiates request to server through browser
Browser sends preflight request
Server sends preflight response
Client
Browser
Server
Client receives actual response
Server sends actual response
Browser sends actual request

CORS в go-chi

```go
package main

import (
    "net/http"

    "github.com/go-chi/chi/v5"
    "github.com/go-chi/cors"
)

func main() {
    r := chi.NewRouter()
    r.Use(cors.Handler(cors.Options{
        AllowedOrigins:     nil,
        AllowOriginFunc:    nil,
        AllowedMethods:     nil,
        AllowedHeaders:     nil,
        ExposedHeaders:     nil,
        AllowCredentials:   false,
        MaxAge:             0,
        OptionsPassthrough: false,
        Debug:              false,
    }))

    http.ListenAndServe(":5000", r)
}
```

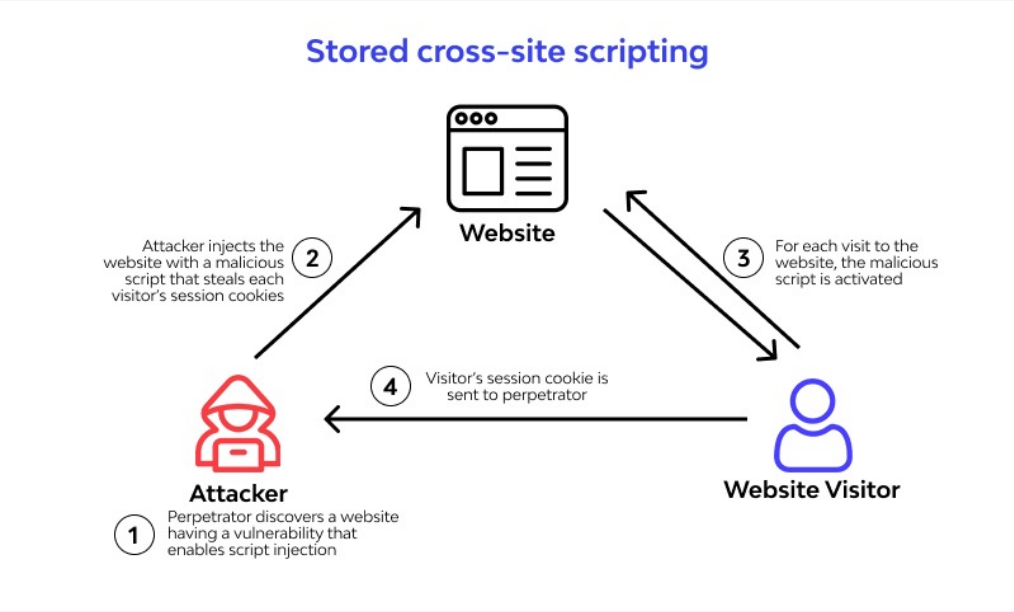
Cross-site request forgery CSRF (XSRF)



Как спастись:

- Не надеяться только на CORS
- csrf-tokens

Cross-site scripting (XSS)

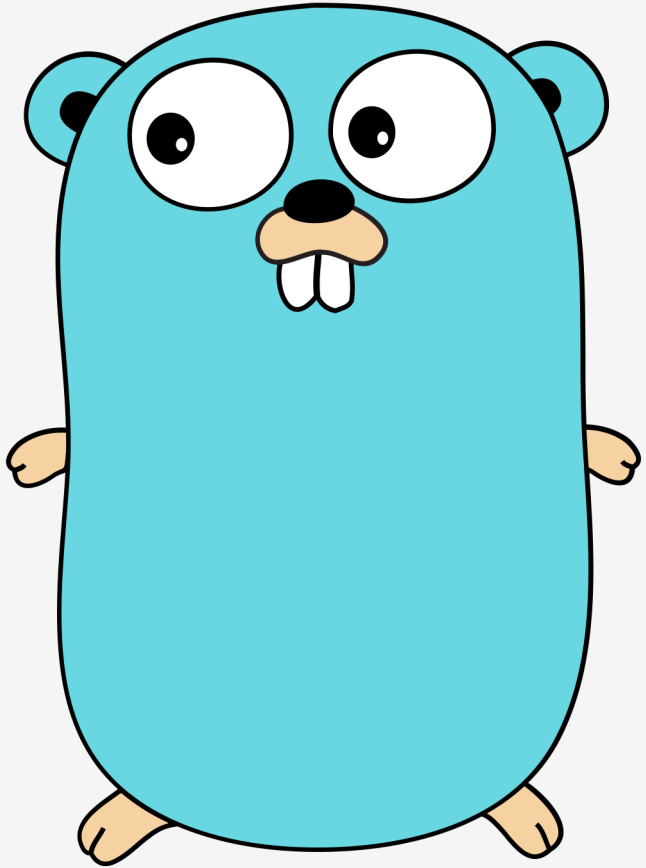


Как спастись? Экранируй все:

- HTML (html/template)
- JS, CSS
- SQL

Итоги

- websocket
- templates
- cors, csrf, xss



WEB-SOCKET, TEMPLATES

Гаратуев Шамиль
s.garatuev@tinkoff.ru
@SquareLemon