Websockets in Go

Hamza Ali

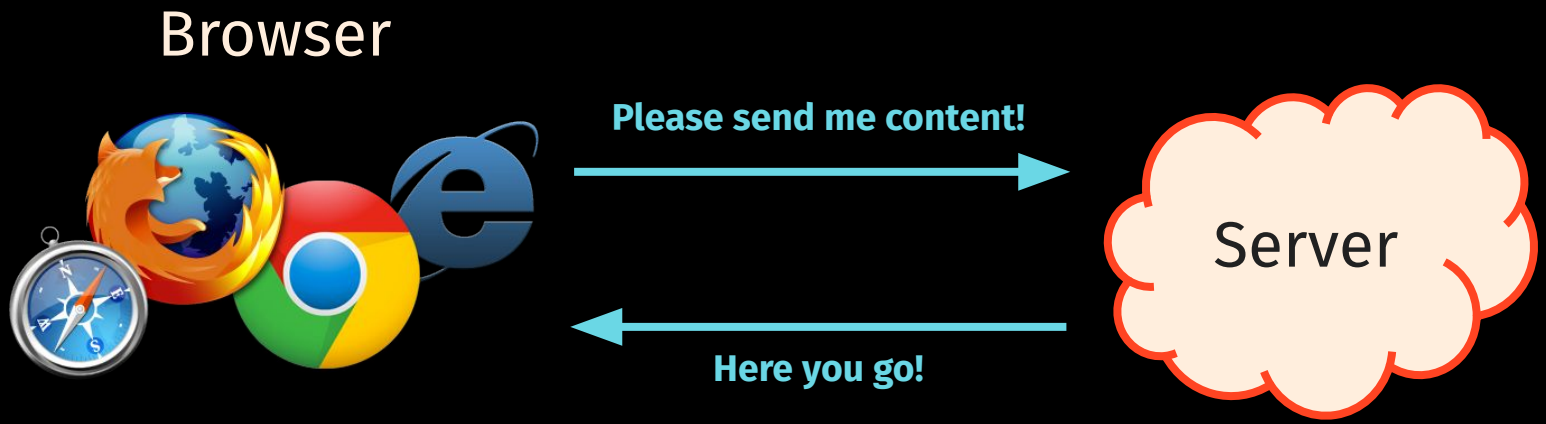
About Me

Hamza Ali is a high school student studying in Jakarta. Started using Go in early 2018 as a language to replace Java for backend projects. Currently a Junior (Grade 11 student) in Jakarta Intercultural School. Hamza has been tinkering, creating, (and breaking) web applications for almost 4 years now, and has learned a lot about how they work.

Why Websockets?

A Primer on HTTP.

How Does HTTP Work?



HTTP: Data Upon Request

- The server has no way of telling the client there is new information.
- The client must initiate all communication
- The server can only respond once for every request.

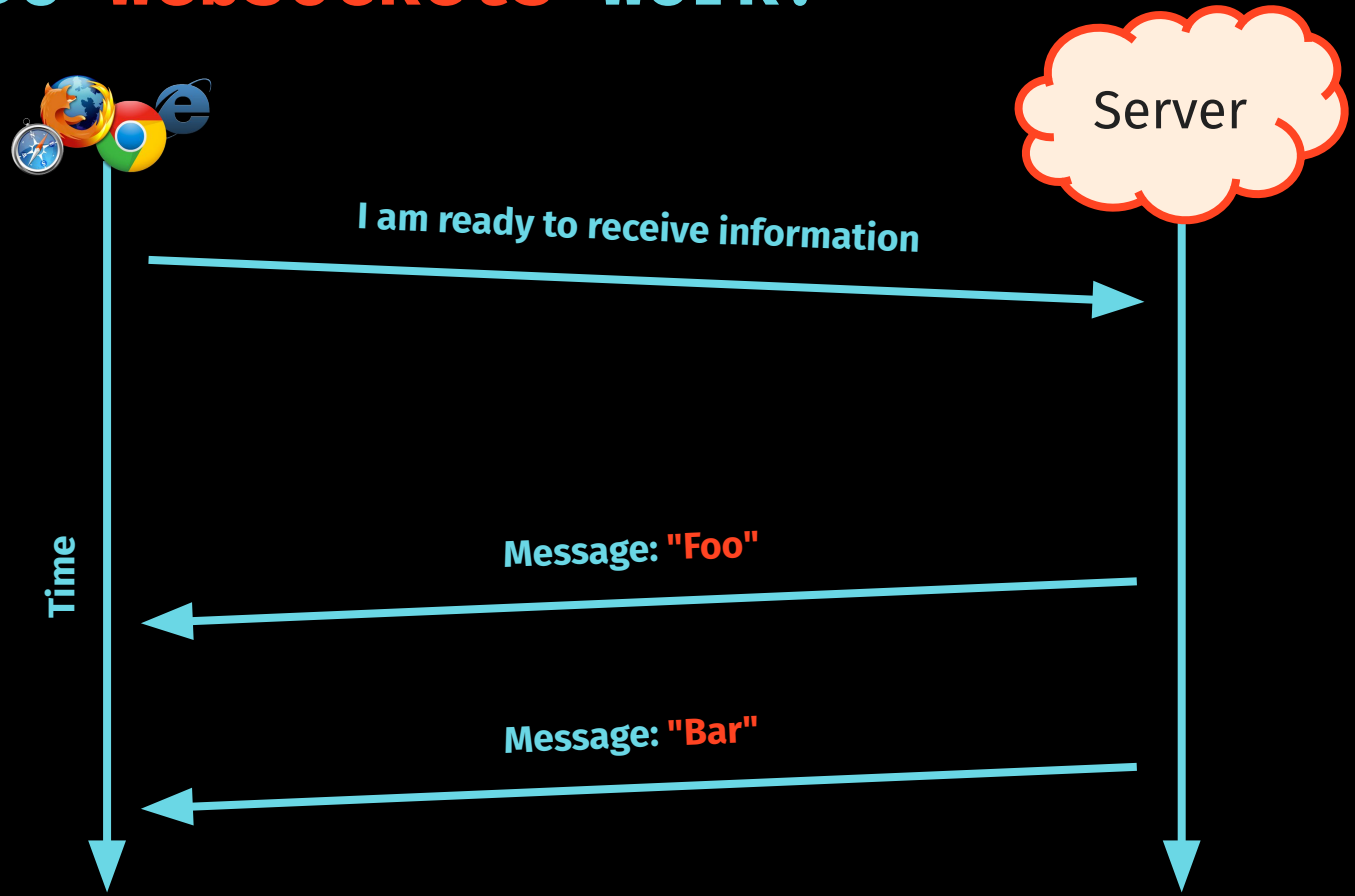
But Then How Would I Make...

- Social Feeds
- Chatting Applications
- Collaboration Suites
- Data Streams
- Online Games
- Financial Software

What are WebSockets?

A protocol for real-time communication between the browser and server.

How Do Websockets Work?



Under the hood

- Sends a regular HTTP Request with an `Upgrade` header
- Keeps the underlying TCP connection alive
- Uses the `ws(s)://` protocol instead of `http(s)://`
- Data is transferred through a WebSocket as messages

Websockets Alternatives

Server Side events

No support for data from the client

No disconnection detection

Single duplex system.

HTTP Long Polling

Possible unreliable message ordering

Increased bandwidth usage

Intensive on server

Server-Side Implementation

```go
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello World")
}

func main() {
    http.Handle("/verify", indexHandler)

    err := http.ListenAndServe(":8080", nil)
}
```


→ curl http://localhost:8080/verify


→ curl http://localhost:8080/verify
Hello World

Handling Websocket Requests

Gorilla Websocket Library

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

Notes:

1. Large messages are fragmented in Chrome's new WebSocket implementation.
2. The application can get the type of a received data message by implementing a Codec marshal function.
3. The go.net io.Reader and io.Writer operate across WebSocket frame boundaries. Read returns when the input buffer is full or a frame boundary is encountered. Each call to Write sends a single frame message. The Gorilla io.Reader and io.WriteCloser operate on a single WebSocket message.

github.com/gorilla/websocket

```go
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }
    // ...
}
```

```go
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }
    // ...
}
```

```go
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }
    // ...
}
```

```
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }
    // ...
}
```

```go
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }
    // ...
}
```

```go
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint("WS connection error: ", err)
        return
    }
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

```go
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

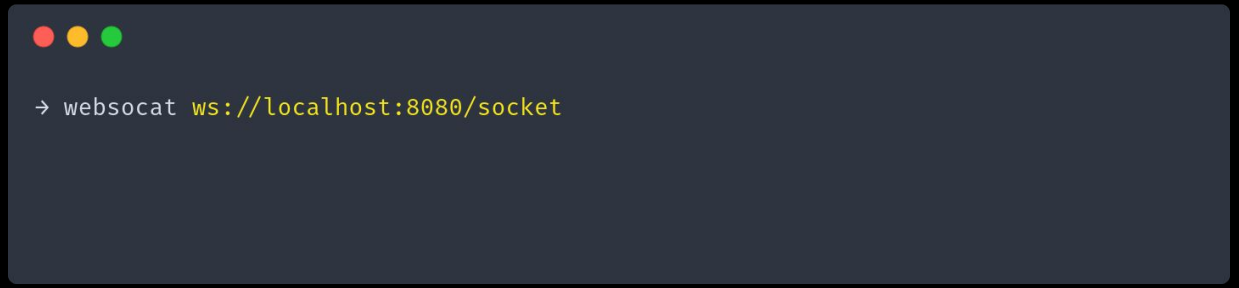
```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    // ...
    defer ws.close()

    for {
        msgType, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            handleDisconnection(ws)
            break
        }

        msg := string(bytes)
        handleIncomingMessage(ws, msg)
    }
}
```

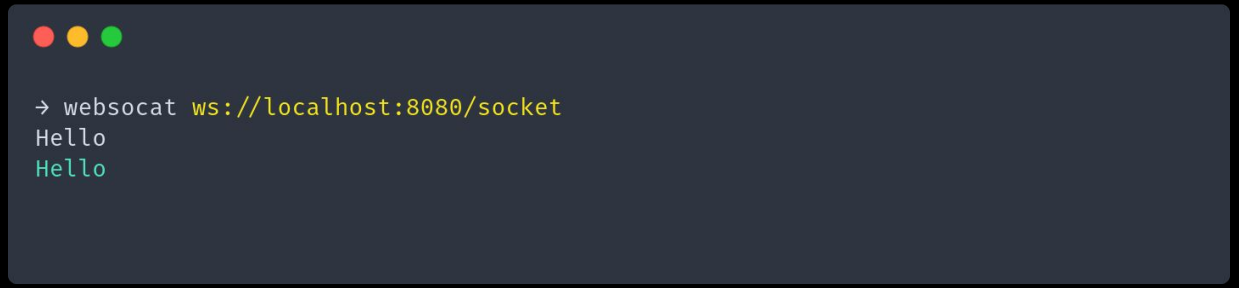
A Simple Echo Server

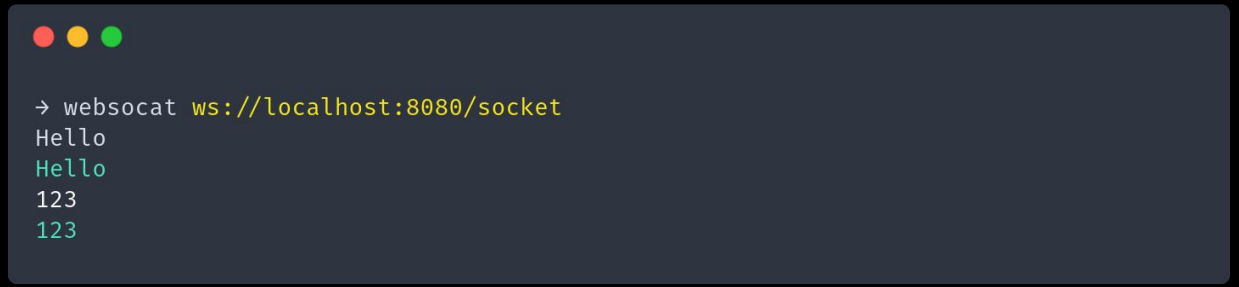
```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }

    defer ws.Close()

    for {
        _, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            break
        }

        ws.WriteMessage(websocket.TextMessage, bytes)
    }
}
```

A Simple Echo Server

```
func socketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        _, _ = fmt.Fprint(w, "WS connection error: ", err)
        return
    }

    defer ws.Close()

    for {
        _, bytes, err := ws.ReadMessage()
        if err ≠ nil {
            break
        }

        ws.WriteMessage(websocket.TextMessage, bytes)
    }
}
```


→ websocat ws://localhost:8080/socket


→ websocat ws://localhost:8080/socket
Hello
Hello


→ websocat ws://localhost:8080/socket
Hello
Hello
123
123

The Client Side of Things

```
let socket = new WebSocket("ws://localhost:8080/socket");

socket.onopen = function(event) {

};

socket.onmessage = function(event) {

};

socket.onclose = function(event) {

};
```

```
let socket = new WebSocket("ws://localhost:8080/socket");

socket.onopen = function(event) {

};

socket.onmessage = function(event) {

};

socket.onclose = function(event) {

};
```

```
let socket = new WebSocket("ws://localhost:8080/socket");

socket.onopen = function(event) {

};

socket.onmessage = function(event) {

};

socket.onclose = function(event) {

};
```

```
let socket = new WebSocket("ws://localhost:8080/socket");

socket.onopen = function(event) {

};

socket.onmessage = function(event) {

};

socket.onclose = function(event) {

};
```

```
let socket = new WebSocket("ws://localhost:8080/socket");

socket.onopen = function(event) {

};

socket.onmessage = function(event) {

};

socket.onclose = function(event) {

};
```

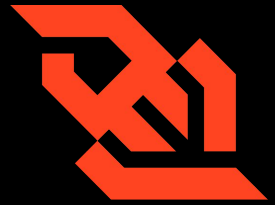
```
let socket = new WebSocket("ws://localhost:8080/socket");

socket.onopen = function(event) {

};

socket.onmessage = function(event) {

};

socket.onclose = function(event) {

};

socket.send("...");
```

Demo

Extending The Project

To the Next Level

- Populate with message history (past 50 messages, infinite scrolling)
- Allow user authentication (username + password)
- Support editing and deleting messages



Websockets

- Server-Side gorilla / websocket 
- Client-Side http://javascript.info/websocket
- Me me@hamzantal.pw
- Slides/Src
  http://hamzantal.pw/websockets

Websockets in Go

Hamza Ali