Web Development (Frontend)

In a nutshell

Jonathan Eberle - September 2024

BACKEND DEVELOPER
CSS
flip.com

Agenda

- Network and the Internet, HTTP(S) and REST
- HTML (Formatting, Semantics, SEO)
- HTML Forms
- Selectors
- CSS (Box Model, Flex and Grid, Media Queries, Pseudo Elements)
- JS (DOM interaction, events, module, objects, typescript)
- Frameworks

How to use this

- This document is a presentation i held in 42Heilbronn from a student to students in my free time and requires the knowledge at the knowledge that 42 students in the last third of the core curriculum should have
- It does not claim to be complete and flawless
- Use it with this repo: https://github.com/Ebejay95/webdev_in_a_nutshell_42_tutorium
- You can find demo HTML so show some stuff (marked by &demo.html)
    - Download from GitHub and open locally in browser
- A C based quiz engine to test knowledge (marked by &quiz.txt)
    - Follow the README Guide on how to run the c quiz with the given question collection
    - Sometimes there are additional advanced question packs quiz-advanced.txt
- Some Tasks which require go and frontend knowledge as well as a little c (marked by &t)

Network and the Internet

A Computer

- Cable or Antenna
- Networkcards (MAC Addresses) XX:XX:XX:XX:XX:XX
- IP Address XXX.XXX.XXX.XXX
- Ports (offer a single service)
- e.g. :25 SMTP, :80 HTTP, :21 FTP, :53 DNS, :443 HTTPS :3306 MySQL &ports.txt &services.txt
- Well Known ports 0 -1023 (reserved to specific service)
- Registered ports 1024 - 49151 (not reserved to specific service)
- Dynamic ports or private ports 49152 - 65535 (use behind the scenes)
- https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Network and the Internet

URLs &url.txt

- protocol://host:port/path/.../.../?param=value&otherparam=othervalue
- protocol://host:port/path/.../.../#anchor
- https://mydomain.com:8888/app/users/?sort=ASC&search=admin
- https://mydomain.com:8888/app/home/#about-us

Network and the Internet

HTTP - The Content of a Request

```
Request Line: Contains the HTTP method (e.g., GET, POST), the resource path, and the HTTP
version (e.g., HTTP/1.1).

Headers: Provide metadata about the request, such as Host, User-Agent, and Content-Type.

Empty Line: Separates headers from the body, indicating the end of the header section.

Request Body (optional): Includes data sent to the server, used in methods like POST or PUT.
According on the content type strings in here are interpreted differently.

HTTP Methods: Defines the action, with common methods being GET, POST, PUT, and DELETE.

Query Parameters (optional): Pass extra data in the URL, usually after a ?, like /search?
q=example.
```

Network and the Internet

HTTP - Response and status

`Content Type defines the way the response is interpreted`

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

REST - Representational State Transfer &rest.txt

• GET:	Retreive data from a server	app/users/
• GET:	Retreive data from a server	app/users/123
• POST:	Create new data on a server	app/users/add
• PUT:	Fully update a existing ressource	app/users/edit/123
• PATCH:	Partially update of a given ressource	app/users/changepw/123
• DELETE:	Remove data from a server	app/users/remove/123

REST - Representational State Transfer

Action on serverside - example

```
Go api example and showcase req res files
https://github.com/Ebejay95/go_api
```

JSON - Data Representation of Data

Action on serverside - example

```go
type Event struct {
    ID int64
    Name string `binding: required`
    Description string `binding: required`
    Location string `binding: required`
    DateTime time.Time `binding: required`
    UserID int
}
```

```sql
    CREATE TABLE IF NOT EXISTS events (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        description TEXT NOT NULL,
        location TEXT NOT NULL,
        dateTime DATETIME NOT NULL,
        user_id INTEGER
    )
```

Request Content Types

```
text/plain                          - Raw text files
text/html                           - HTML web pages
text/css                            - CSS stylesheets
text/javascript                     - (Legacy) JavaScript files
text/csv                            - CSV data files
text/xml                            - XML data (though application/xml is preferred)
```

```
Copyapplication/json                - JSON data
application/xml                     - XML data
application/javascript              - JavaScript files (modern)
application/pdf                     - PDF documents
application/zip                     - ZIP archives
application/x-www-form-urlencoded   - Form data
application/octet-stream            - Binary files/downloads
```

```
Copyimage/jpeg                      - JPEG images
image/png                           - PNG images
image/gif                           - GIF images
image/svg+xml                       - SVG images
image/webp                          - WebP images
image/avif                          - AVIF images
```

```
Copyaudio/mpeg                      - MP3 audio
audio/wav                           - WAV audio
video/mp4                           - MP4 video
video/webm                          - WebM video
```

```
Copymultipart/form-data             - File uploads/form data with files
multipart/mixed                     - Email with attachments
```

JSON - Data Representation of Data

Action on serverside - example

```go
type Event struct {
    ID int64
    Name string `binding: required`
    Description string `binding: required`
    Location string `binding: required`
    DateTime time.Time `binding: required`
    UserID int
}
```

```json
 {

    "id": 1,
    "name": "Test event",
    "description": "A test event",
    "location": "A test location",
    „date_time": "2024-01-01T15:30:00Z",
    „user_id": 1
 }
```

JSON - Data Representation of Data

Action on serverside - example

```
[

  {

    "id": 1,
    "name": "Test event",
    "description": "A test event",
    "location": "A test location",
    „date_time": "2024-01-01T15:30:00Z",
    „user_id": 1
  },
  {

    "id": 2,
    "name": "Test event",
    "description": "A test event",
    "location": "A test location",
    "date_time": "2024-01-01T15:30:00Z",
    "user_id": 1
  }
]
```

REST - Representational State Transfer

Action on serverside - the server

```go
package main

import (
    "example.com/main/db"
    "example.com/main/routes"
    "github.com/gin-gonic/gin"
)

func main () {
    db.InitDB()
    server := gin.Default()
    routes.RegisterRoutes(server)
    server.Run(":8080")
}
```

REST - Representational State Transfer

Action on serverside - the routes

```go
package routes

import "github.com/gin-gonic/gin"

func RegisterRoutes(server *gin.Engine) {
    server.GET("/events", getEvents)
    server.GET("/events/:id", getEvent)
    server.POST("/events", createEvent)
    server.PUT("/events/:id", updateEvent)
    server.DELETE("/events/:id", deleteEvent)
}
```

REST - Representational State Transfer

Action on serverside - getEvents - Request

```
GET http://localhost:8080/events/
```

REST - Representational State Transfer

Action on serverside - getEvents

```
server.GET("/events", getEvents)
```

```
func getEvents(context *gin.Context) {
    events, err := models.GetAllEvents()
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not fetch events"})
        return
    }
    context.JSON(http.StatusOK, events)
}
```

REST - Representational State Transfer

Action on serverside - getEvents - Database Interaction

```go
func GetAllEvents() ([]Event, error ){
    query := `SELECT * FROM events`
    rows, err := db.DB.Query(query)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var events []Event
    for rows.Next() {
        var event Event
        err := rows.Scan(&event.ID, &event.Name, &event.Description, &event.Location, &event.DateTime,
&event.UserID)
        if err != nil {
            return nil, err
        }
        events = append(events, event)
    }
    return events, nil;
}
```

REST - Representational State Transfer

Action on serverside - getEvent - Request

```
GET http://localhost:8080/events/1
```

REST - Representational State Transfer

Action on serverside - getEvent

```go
server.GET("/events/:id", getEvent)

func getEvent(context *gin.Context) {
    id, err := strconv.ParseInt(context.Param("id"), 10, 64)
    if err != nil {
        context.JSON(http.StatusBadRequest, gin.H{"message":"id wrong formatted"})
        return
    }
    event, err := models.GetEvent(id)
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not fetch event"})
        return
    }
    context.JSON(http.StatusOK, event)
}
```

REST - Representational State Transfer

Action on serverside - getEvent - Database Interaction

```
func GetEvent(id int64) (*Event, error ){
    query := `SELECT * FROM events WHERE id = ?`
    row := db.DB.QueryRow(query, id)
    var event Event
    err := row.Scan(&event.ID, &event.Name, &event.Description, &event.Location, &event.DateTime,  &event.UserID)
    if err != nil {
        return nil, err
    }
    return &event, nil;
}
```

REST - Representational State Transfer

Action on serverside - createEvent - Request

```
POST http://localhost:8080/events
content-type: application/json

{
    "name":"Test event",
    "description":"A test event",
    "location":"A test location",
    "dateTime":"2024-01-01T15:30:00.000Z"
}
```

REST - Representational State Transfer

Action on serverside - createEvent

```go
server.POST("/events", createEvent)

func createEvent(context *gin.Context) {
    var event models.Event
    err := context.ShouldBindJSON(&event)
    if err != nil {
        context.JSON(http.StatusBadRequest, gin.H{"message":"could not parse request data"})
        return
    }
    event.ID = 1
    event.UserID = 1
    err = event.Save()
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not create event"})
        return
    }
    context.JSON(http.StatusCreated, gin.H{"message":"Event created!", "event": event})
}
```

REST - Representational State Transfer

Action on serverside - createEvent - Database Interaction

```go
func (e *Event) Save() error {
    query := `
    INSERT INTO events(name, description, location, dateTime, user_id)
    VALUES (?,?,?,?,?)
    `
    stmt, err := db.DB.Prepare(query)
    if err != nil {
        return err
    }
    defer stmt.Close()
    result, err := stmt.Exec(e.Name, e.Description, e.Location, e.DateTime, e.UserID)
    if err != nil {
        return err
    }
    id, err := result.LastInsertId()
    if err != nil {
        return err
    }
    e.ID = id
    return nil
}
```

REST - Representational State Transfer

Action on serverside - updateEvent - Request

```
PUT http://localhost:8080/events/1
content-type: application/json

{
    "name":"Updated! event",
    "description":"A test event",
    "location":"A test location (Updated!)",
    "dateTime":"2024-01-01T15:30:00.000Z"
}
```

REST - Representational State Transfer

Action on serverside - updateEvent

```
server.PUT("/events/:id", updateEvent)
```

```
func updateEvent(context *gin.Context) {
    id, err := strconv.ParseInt(context.Param("id"), 10, 64)
    if err != nil {
        context.JSON(http.StatusBadRequest, gin.H{"message":"id wrong formatted"})
        return
    }
    _ , err = models.GetEvent(id)
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not fetch event"})
        return
    }

    var updatedEvent models.Event
    err = context.ShouldBindJSON(&updatedEvent)
    if err != nil {
        context.JSON(http.StatusBadRequest, gin.H{"message":"could not parse request data"})
        return
    }
    updatedEvent.ID = id
    err = updatedEvent.Update()
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not update event"})
        return
    }
    context.JSON(http.StatusCreated, gin.H{"message":"Event updated!", "event": updatedEvent})
}
```

REST - Representational State Transfer

Action on serverside - updateEvent - Database Interaction

```go
func (e *Event) Update() error {
    query := `
    UPDATE events
    SET name = ?, description = ?, location = ?, dateTime = ?
    WHERE id = ?
    `
    stmt, err := db.DB.Prepare(query)
    if err != nil {
        return err
    }
    defer stmt.Close()
    _, err = stmt.Exec(e.Name, e.Description, e.Location, e.DateTime, e.ID)
    return err
}
```

REST - Representational State Transfer

Action on serverside - deleteEvent - Request

```
DELETE http://localhost:8080/events/1
```

REST - Representational State Transfer

Action on serverside - deleteEvent

```go
server.DELETE("/events/:id", deleteEvent)
```

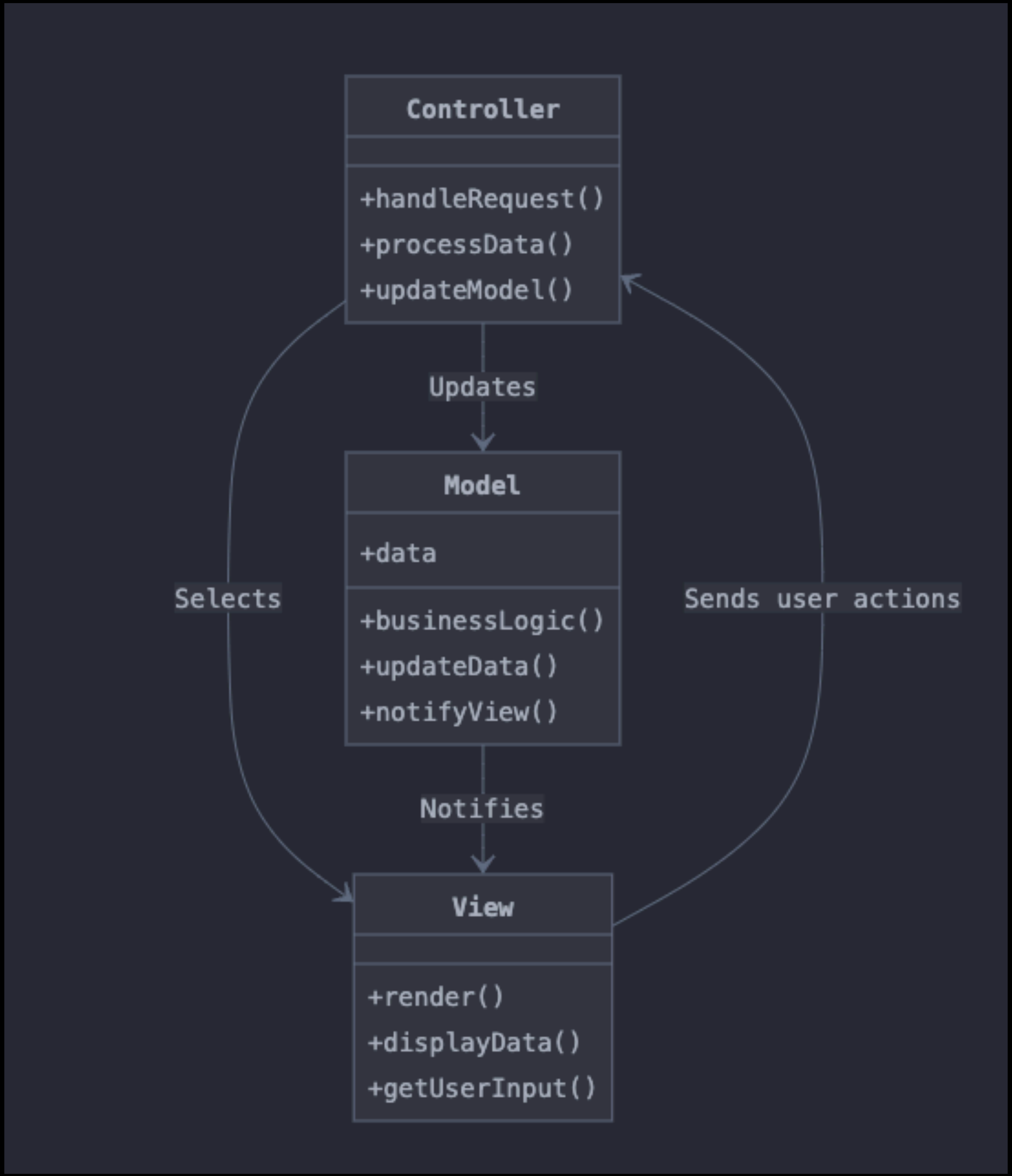
```go
func deleteEvent(context *gin.Context) {
    id, err := strconv.ParseInt(context.Param("id"), 10, 64)
    if err != nil {
        context.JSON(http.StatusBadRequest, gin.H{"message":"id wrong formatted"})
        return
    }
    event, err := models.GetEvent(id)
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not fetch event"})
        return
    }
    err = event.Delete()
    if err != nil {
        context.JSON(http.StatusInternalServerError, gin.H{"message":"could not delete event"})
        return
    }
    context.JSON(http.StatusCreated, gin.H{"message":"Event deleted!"})
}
```

REST - Representational State Transfer

Action on serverside - deleteEvent - Database Interaction

```go
func (e *Event) Delete() error {
    query := `DELETE FROM events WHERE id = ?`
    stmt, err := db.DB.Prepare(query)
    if err != nil {
        return err
    }
    defer stmt.Close()
    _, err = stmt.Exec(e.ID)
    return err
}
```

MVC Structure



I want UI … colours and animations

Well lets call it a introduction to formatting…
And make it interactive - best with our REST API

Jonathan Eberle - September 2024

Another Backend Example

HTML Files are parsed as text/html as HTTP Responses

```go
func renderIndexHTML(c *gin.Context) {
  c.HTML(http.StatusOK, "index.html", gin.H{
    "title": "Welcome to My Website",
    "content": gin.H{
      "message": "Hello from Gin!",
    },
  })
}

func RegisterRoutes(server *gin.Engine) {
    server.GET("/", renderIndexHTML)
}


func main() {
    db.InitDB()
    server := gin.Default()
    server.LoadHTMLGlob("templates/*")
    server.Static("/static", "./static")
    routes.RegisterRoutes(server)
    server.Run(":8080")
}
```

HTML Formatting &html-basics.txt &html-basics.html

A simple Document

```
<!DOCTYPE html> <!-- define html –>
<html lang="en"> <!-- define language –>
<head> <!-- invisible: holds meta data –>
    <meta charset="UTF-8"> <!-- defines charset –>
    <meta name="viewport" content="width=device-width, initial-scale=1.0“>
     <!-- defines charset –>
    <title>Document</title> <!-- define title in Browser Tab or Google Search–>
    <meta name="viewport" content="width=device-width, initial-scale=1.0“>
    <!-- viewport keeps responsive behaviour in any screensize–>
</head>
<body>
    <!-- vidible document: holds content –>
</body>
</html>
```

HTML Formatting

include Assets

```
<head>
    <!—— Everything else needed in your head ->
    <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="https://ultimate-cdn/external.css">
</head>
<body>
    <!—— Your page content ->
    <script src=„./index.js"></script>
    <script src="https://ultimate-cdn/external.js"></script>
</body>
</html>
```

```
A CDN (Content Delivery Network) might provide text/plain resources
```

https://getbootstrap.com/docs/3.3/getting-started/

`https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap-theme.min.css`

HTML Formatting

Tags - give it semantic meanings

- Headlines
- Paragraphs
- Lists ul ol > li
- Articles
- Header Footer Aside Main
- Nav

HTML Formatting

Sources for more semantic tags

https://pwskills.com/blog/list-of-all-html-tags-in-2024/

https://www.w3schools.com/html/

HTML Formatting

Attributes

```
    <nav class="navbar">
        <a href="#home" class="active">Home</a>
        <a href="#services">Services</a>
        <a href="#about">About</a>
        <a href="#contact">Contact</a>
    </nav>
    <main id="content">

    </main>
```

HTML Formatting

Atributes

- id: single element selector
- class: multiple element selector - use for JS interactivity by toggle
- data-*: store data for frontend logic (strings, numbers (as strings), stringified JSON)
- special attributes required by HTML functionality or definition e.g. href, src, value, type, name, autocomplete, controls

HTML Formatting
Atributes

```html
<nav class="navbar">
    <a href="#home" class="active" data-page="home" data-analytics="track-click">Home</a>
    <a href="#services" data-page="services" data-analytics="track-click">Services</a>
    <a href="#about" data-page="about" data-analytics="track-click">About</a>
    <a href="#contact" data-page="contact" data-analytics="track-click">Contact</a>
</nav>
```

```
html Attributes can be modified easily and their use for security relevant behaviours is not
recommended
```

```html
<a href="https://example.com">Visit Example</a>
<img src="image.jpg" alt="Example Image">
<input type="text" value="Default Text">
<input type="password">
<input type="text" name="username">
<input type="email" autocomplete="on">
<video controls>
   <source src="video.mp4" type="video/mp4">
</video>
```

HTML Formatting

aria-labels

```html
<button class="accordion" aria-expanded="false" aria-controls="section1-content" id="section1">Section 1</button>
<div class="accordion-content" id="section1-content" aria-labelledby="section1">
    <p>This is the content of Section 1. You can include any HTML elements here.</p>
</div>

<button class="accordion" aria-expanded="false" aria-controls="section2-content" id="section2">Section 2</button>
<div class="accordion-content" id="section2-content" aria-labelledby="section2">
    <p>This is the content of Section 2. Expand to read more information.</p>
</div>

<button class="accordion" aria-expanded="false" aria-controls="section3-content" id="section3">Section 3</button>
<div class="accordion-content" id="section3-content" aria-labelledby="section3">
    <p>This is the content of Section 3. Accordion elements are great for organizing content.</p>
</div>
```

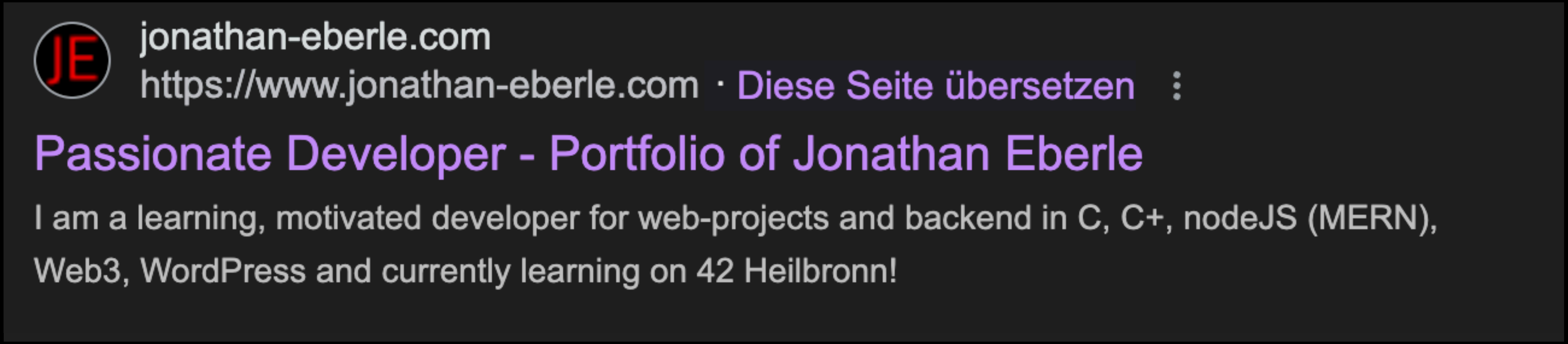
HTML Formatting

SEO aspects

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sample Page</title>
    <meta name="description" content="This is a simple HTML document with an H1 tag, referencing a 
sitemap and robots.txt file.">
    <link rel="sitemap" type="application/xml" title="Sitemap" href="/sitemap.xml">
    <meta name="robots" content="index, follow">
</head>
<body>

    <img src="path/to/me.png" alt="me.png" title="a picture of me">

    <h1>Welcome to the Sample Page</h1>

</body>
</html>
```

HTML Formatting

SEO aspects

- 

jonathan-eberle.com
https://www.jonathan-eberle.com · Diese Seite übersetzen

Passionate Developer - Portfolio of Jonathan Eberle

I am a learning, motivated developer for web-projects and backend in C, C+, nodeJS (MERN), Web3, WordPress and currently learning on 42 Heilbronn!

REST and HTML Forms - Backend Connect

• GET:	Retreive data from a server	app/users/
• GET:	Retreive data from a server	app/users/123
• POST:	Create new data on a server	app/users/add
• PUT:	Fully update a existing ressource	app/users/edit/123
• PATCH:	Partially update of a given ressource	app/users/changepw/123
• DELETE:	Remove data from a server	app/users/remove/123

- method
- action
- Identifier

REST and HTML Forms - Backend Connect

How use that now? &html-forms.txt &forms.html

```
<form method="post" action="/action/url">
    <!-- FORM FIELDS -->
</form>
```

```
method="put"
method="patch"
method="delete“
```

```
action="/action/url"
```

```
action is the service that needs to be provided by your serverside
application routes
```

REST and HTML Forms - Form Fields

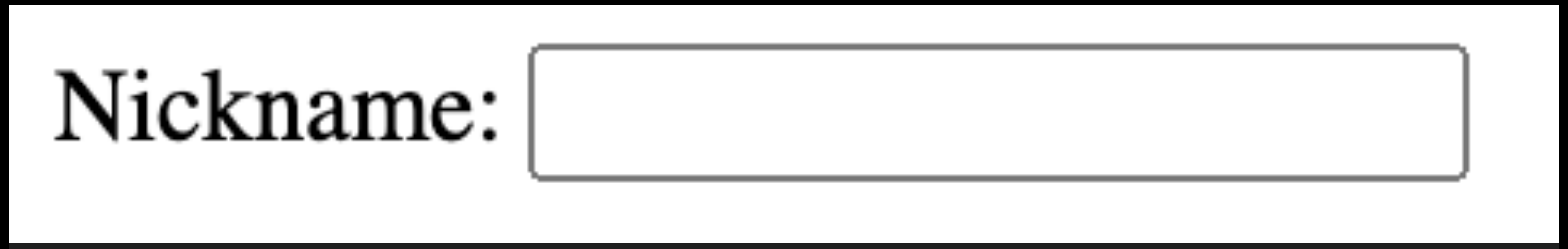
Provide relevant data

```
<form method="post" action="/app/users/edit/<?php echo $user->id ?>">
    <!-- Nickname -->
    <label for="nickname">Nickname:</label>
    <input type="text" id="nickname" name="nickname" value="<?php echo $user->nickname ?>" required>
    <br><br>

    <!-- Email -->
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" value="<?php echo $user->email ?>" required>
    <br><br>

    <!-- Geschlecht (Select) -->
    <label for="gender">Geschlecht:</label>
    <select id="gender" name="gender">
        <option value="male" <?php echo ($user->gender == 'male') ? 'selected' : '' ?>>Männlich</option>
        <option value="female" <?php echo ($user->gender == 'female') ? 'selected' : '' ?>>Weiblich</option>
        <option value="other" <?php echo ($user->gender == 'other') ? 'selected' : '' ?>>Andere</option>
    </select>
    <br><br>

    <!-- Role (Radio) -->
    <label>Rolle:</label>
    <input type="radio" id="admin" name="role" value="admin" <?php echo ($user->role == 'admin') ? 'checked' : '' ?>>
    <label for="admin">Admin</label>
    <input type="radio" id="user" name="role" value="user" <?php echo ($user->role == 'user') ? 'checked' : '' ?>>
    <label for="user">User</label>
    <br><br>

    <!-- Example Checkbox -->
    <label for="newsletter">Newsletter abonnieren:</label>
    <input type="checkbox" id="newsletter" name="newsletter" value="yes" <?php echo ($user->newsletter == 'yes') ? 'checked' : '' ?>>
    <br><br>

    <!-- Range Slider -->
    <label for="age">Alter (Range):</label>
```

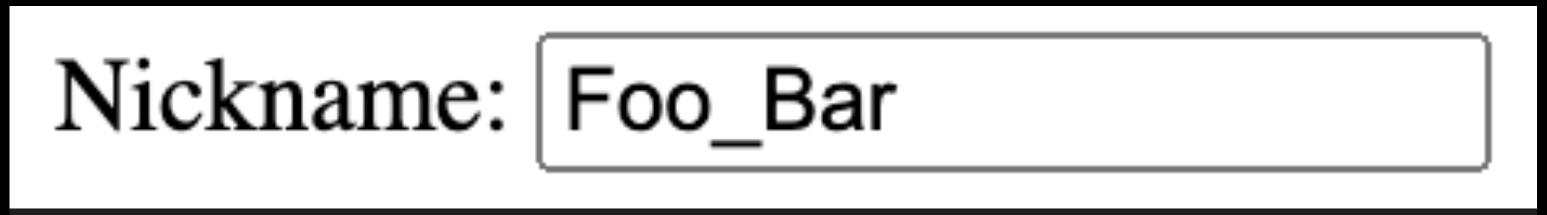
REST and HTML Forms - Form Fields

Provide relevant data - Text Field

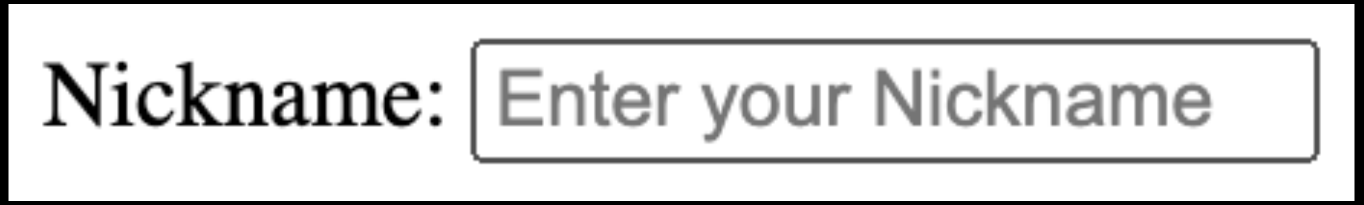
```
    <!-- Nickname -->
    <label for="nickname">Nickname:</label>
    <input type="text" id="nickname" name="nickname" value="<?php echo $user->nickname ?>" required>
    <br><br>
```


Nickname:

```
without a value
```


Nickname: Foo_Bar

```
value entered or retrieved from server
```

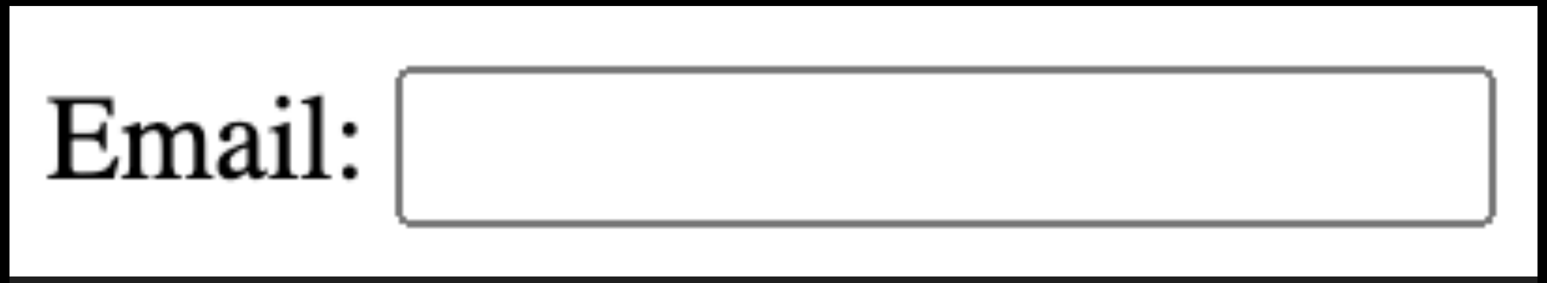

Nickname: Enter your Nickname

```
placeholder="Enter your Nickname"
```

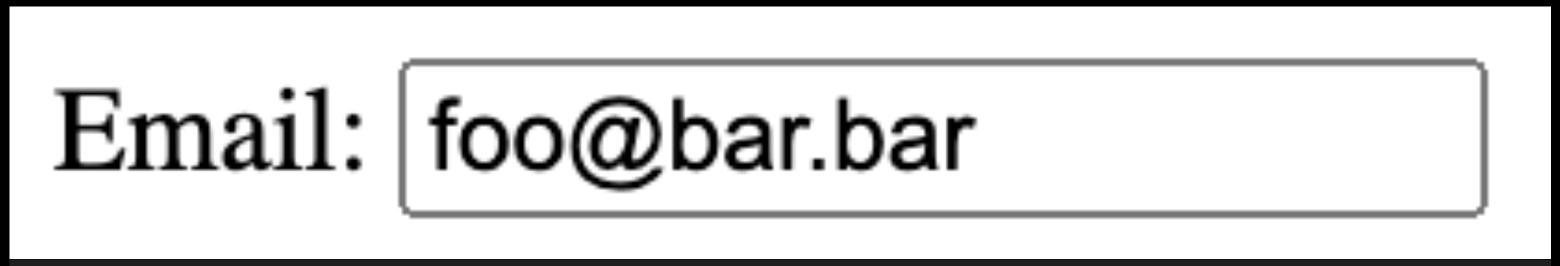
REST and HTML Forms - Form Fields

Provide relevant data - Email Field

```
    <!-- Email -->
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" value="<?php echo $user->email ?>" required>
    <br><br>
```



```
without a value
```



```
value entered or retrieved from server
```

```
placeholder="Enter your E-Mail"
autocomplete=„on"
and many more!!!!!
A text-like field is also available for password - check for your app which level of security for password changing
ist required.
```

REST and HTML Forms - Form Fields

Provide relevant data - Other Text-Like fields

```
<input type="text">
A simple text field for single-line text input.

<input type="email">
A text field specifically for email addresses, supporting validation for the email format.

<input type="password">
A text field for password input. The entered text is masked with dots or stars.

<input type="search">
A search field that, in some browsers, offers special search functionalities like a clear button ("x").

<input type="url">
A text field for URL input that validates the entry as a proper URL.

<input type="tel">
A text field for phone numbers that focuses on number input (e.g., it shows a phone keypad on mobile
devices).

<input type="number">
A text field for entering numbers, with arrows for increasing or decreasing the value.
```

REST and HTML Forms - Form Fields

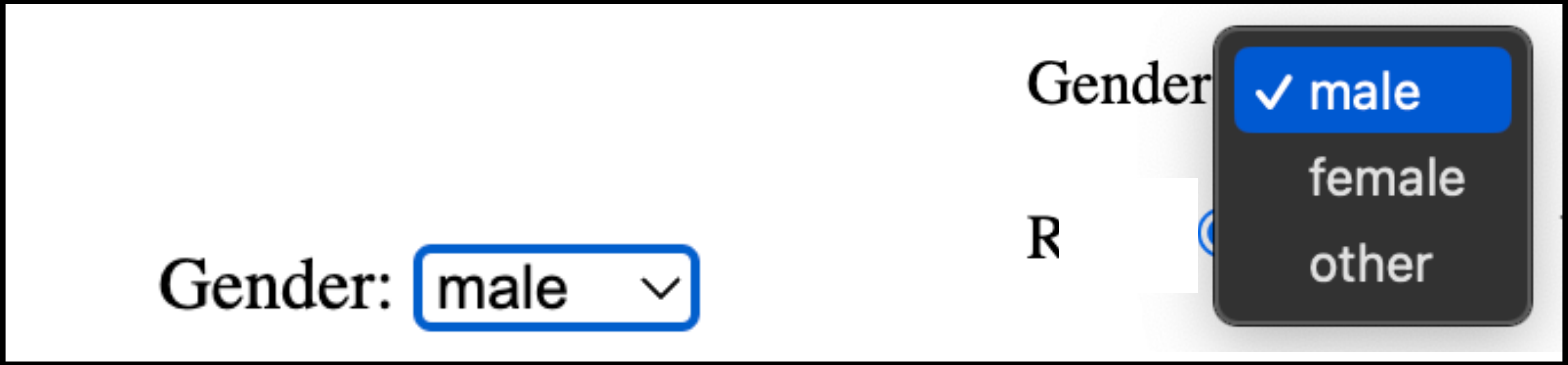
Provide relevant data - Other Text-Like fields

```
<input type="date">
A field for entering dates, offering a date picker in supported browsers.

<input type="datetime-local">
A field for entering both date and time without a timezone.

<input type="month">
A field for selecting a month and year.

<input type="week">
A field for selecting a specific week of a year.

<input type="time">
A field for entering a time (hours and minutes).

<input type="color">
A field for selecting a color with a color picker widget.

<input type="hidden">
A hidden input field that is not visible in the browser but can store values for server submission.
```

REST and HTML Forms - Form Fields

Provide relevant data - Select Field

```
    <!-- Gender (Select) -->
    <label for="gender">Gender:</label>
    <select id="gender" name="gender">
        <option value="male" <?php echo ($user->gender == 'male') ? 'selected' : '' ?>>male</option>
        <option value="female" <?php echo ($user->gender == 'female') ? 'selected' : '' ?>>female</option>
        <option value="other" <?php echo ($user->gender == 'other') ? 'selected' : '' ?>>other</option>
    </select>
    <br><br>
```



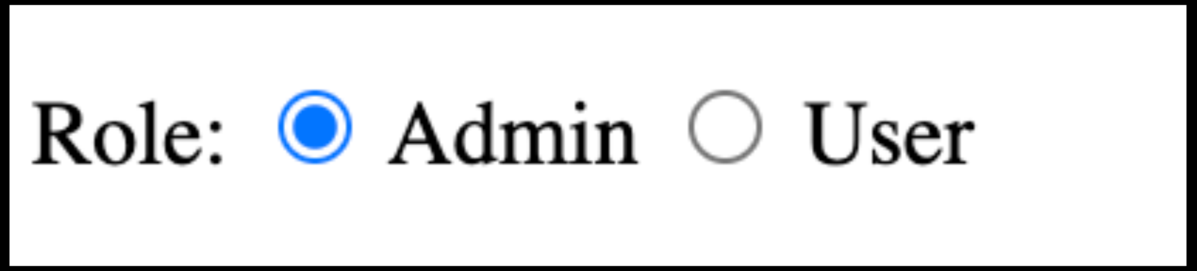
```
<option value="male" selected>male</option>
```

```
 points at the option that is selected currently
```

REST and HTML Forms - Form Fields

Provide relevant data - Radio Field

```
    <!-- Role (Radio) -->
    <label>Role:</label>
    <input type="radio" id="radioadmin" name="role" value="admin" <?php echo ($user->role == 'admin') ? 'checked' : '' ?>>
    <label for="radioadmin">Admin</label>
    <input type="radio" id="radiouser" name="role" value="user" <?php echo ($user->role == 'user') ? 'checked' : '' ?>>
    <label for="radiouser">User</label>
    <br><br>
```

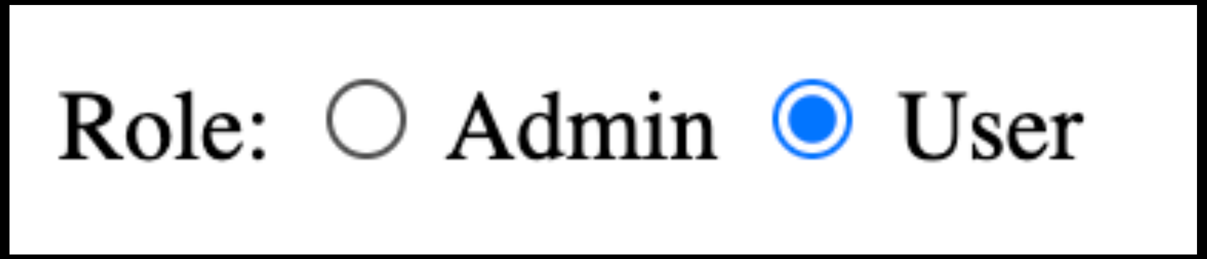


Role: [✓ Admin] User

```
    <input type="radio" id="user" name="role" value="user" checked>
    <label for="user">User</label>
```

```
points at the option that is checked currently
```

```
for=„user" => id=„user"
```

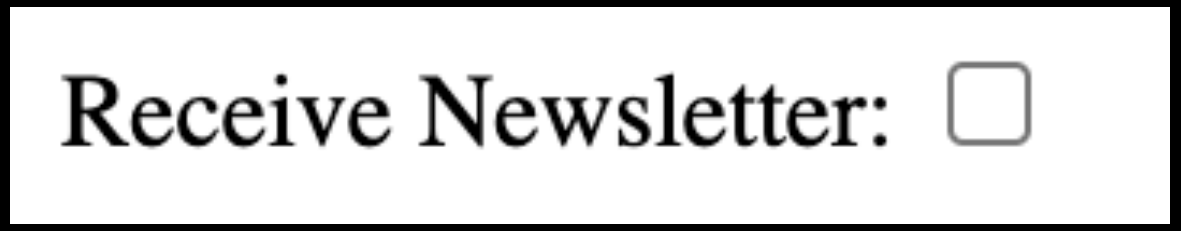


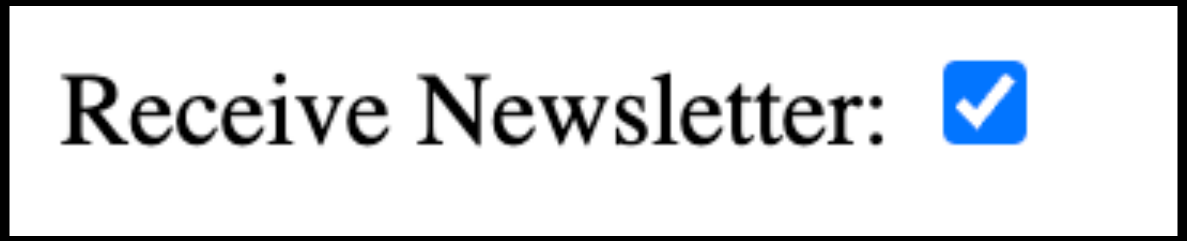
Role: Admin [✓ User]

REST and HTML Forms - Form Fields

Provide relevant data - Checkbox Field

```
    <!-- Checkbox -->
    <label for="newsletter">Receive Newsletter:</label>
    <input type="checkbox" id="newsletter" name="newsletter" value="yes" <?php echo ($user->newsletter == 'yes') ? 'checked' : '' ?>>
    <br><br>
```

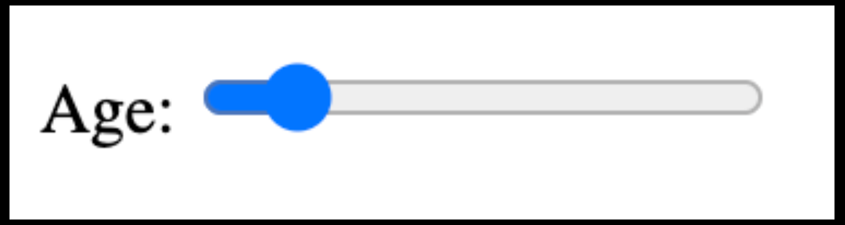

Receive Newsletter: ☐


Receive Newsletter: ☑

REST and HTML Forms - Form Fields

Provide relevant data - Range Field

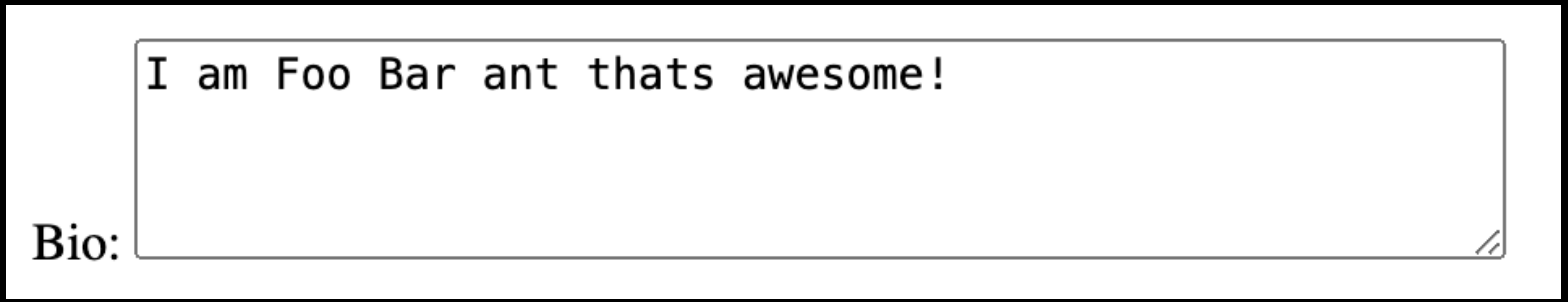
```
    <!-- Range Slider -->
    <label for="age">Age:</label>
    <input type="range" id="age" name="age" min="18" max="100" value="<?php echo $user->age ?>">
    <br><br>
```



REST and HTML Forms - Form Fields

Provide relevant data - Texture Field

```
  <!-- Textarea -->
    <label for="bio">Bio:</label>
    <textarea id="bio" name="bio" rows="4" cols="50"><?php echo $user->bio ?></textarea>
    <br><br>
```


Bio: I am Foo Bar ant thats awesome!

```
similar attributes as in text fields are supported
```

REST and HTML Forms - Form Fields

Validate

- e.g. the Nickname should only contain lowercase or digit chars. (check by JS)
- e.g. the Nickname should be unique in the DB (check server side - use AJAX)
- e.g. the Request should be only allowed in a session or by role (check auth token)
- e.g. the Request should not contain harming code (script od SQL Injections) - apply cleaning mechanisms like htmlspecialchars, regexes or others….
- e.g. the number should be positive and <= 100
- e.g. a field is mandatory (HTML set required attribute)

REST and HTML Forms - Regex &regex.txt

A little view at patterns…. so go validate it….

- Regex: Short for Regular Expressions.
- Pattern matching: Used to search, match, or manipulate text.
- Special characters: Symbols like *, +, . to define patterns.
- Validation: Often used to validate input (e.g., email, phone numbers).
- Search and replace: Can find and modify text based on patterns.
- Text processing: Common in parsing, filtering, and extracting data
- https://regexone.com/ - interactive Learning :D

Where is the nice modern interaction…

And it still looks like crap…

Enough of REST? Do Frontend Stuff…

Jonathan Eberle - September 2024

HTML Formatting

include Assets

```
<head>
    <!—— Everything else needed in your head ->
    <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="https://ultimate-cdn/external.css">
</head>
<body>
    <!—— Your page content ->
    <script src=„./index.js"></script>
    <script src="https://ultimate-cdn/external.js"></script>
</body>
</html>
```

```
A CDN (Content Delivery Network) might provide text/plain resources
https://getbootstrap.com/docs/3.3/getting-started/
```

https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap-theme.min.css

CSS

Make it look nice…

- Include in .css files by link red stylesheet
- or do it inline (in the html file itself

```
<!DOCTYPE html>
<html>
<head>
    <title>Inline JavaScript Example</title>
</head>
<body>
    <h1>Welcome to CSS</h1>

    <style>
        hi {
           font-size: 24px;
        }
       body {
           background-colour: beige;
        }
    </style>
</body>
</html>
```

Selectors &selectors.txt

Pick it!

```
<div class="container"></div>       div.container    or      .container

<p id="intro"></p>                  p#intro          or      #intro

<ul>                                ul > li.item  or  ul li.item   li.item        or         .item
    <li class="item">Item 1</li>
    <li class="item">Item 2</li>
</ul>

<button type="button">Click me</button>   button[type=button]        or      button

<input type="text" name="firstname">   input[type=text]       or     input[name=firstname] or

                                          input[type=text][name=firstname]
```

CSS &css.txt &css.html

Cascading....

```
/* General style rules for all paragraphs */
p {
    font-family: Arial, sans-serif;
    font-size: 16px;
    color: #333;
  }

  /* More specific rule for paragraphs within an article */
  article p {
    line-height: 1.6;
    margin-bottom: 15px;
  }

  /* Even more specific rule for the first paragraph in an article */
  article p:first-child {
    font-weight: bold;
    font-size: 18px;
  }

  /* Inline style (highest specificity) */
  <p style="color: red;">This text will be red.</p>
```

CSS

Oh that looks nice!

```
<link rel="stylesheet" href="./style.css">
```

```
selector {
    attribute1: value;
    attribute2: another-value;
}

p.warning {
    color: red;
    background-color: #ffdddd;
    padding: 10px;
    border: red solid 2px;
}
```

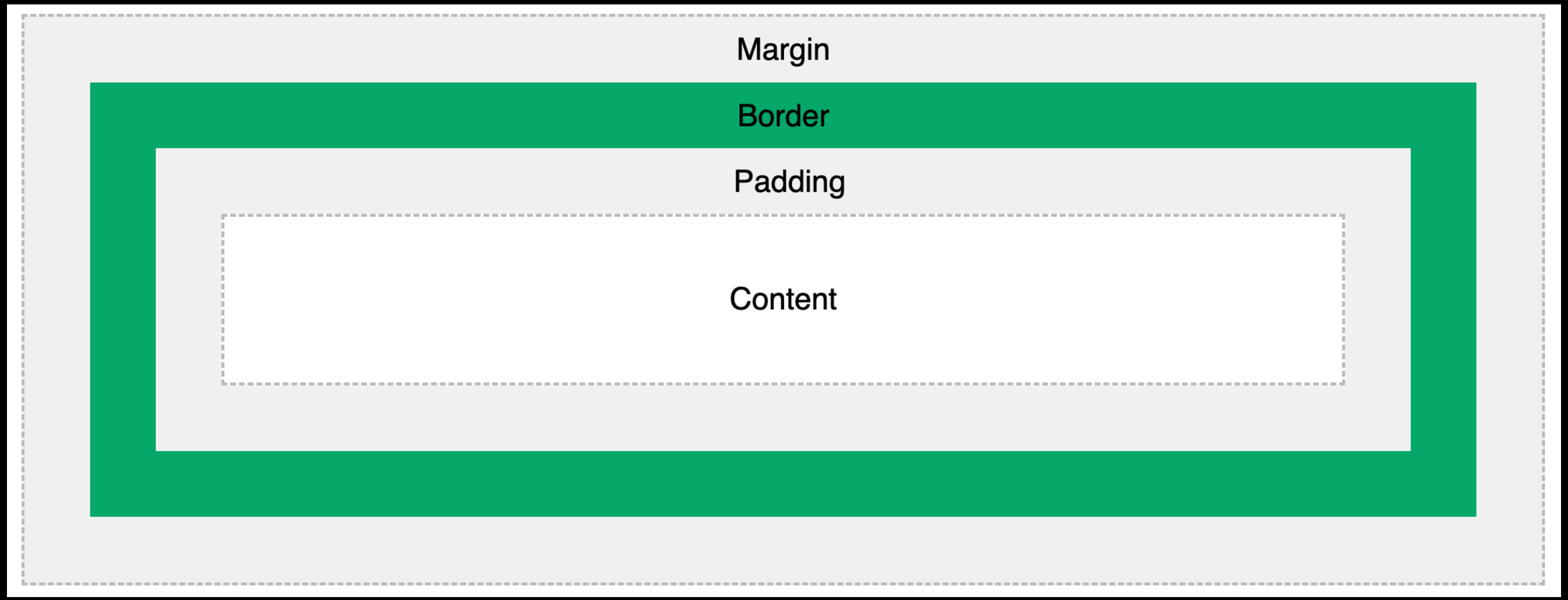
CSS

Oh that looks nice!

```
The :root selector applies on the Root of the document <html> and all its contents.
So variables declared here are available for styling in all contents. So do any declared variables for any of
their child elements recursively

:root {
    --main-color: #3498db;
    --secondary-color: #2ecc71;
    --font-size: 16px;
}

body {
    font-size: var(--font-size);
    color: var(--main-color);
}

.button {
    background-color: var(--secondary-color);
    padding: 10px 20px;
    border: none;
    color: #fff;
}
```

CSS
The Box Model &box-model.html

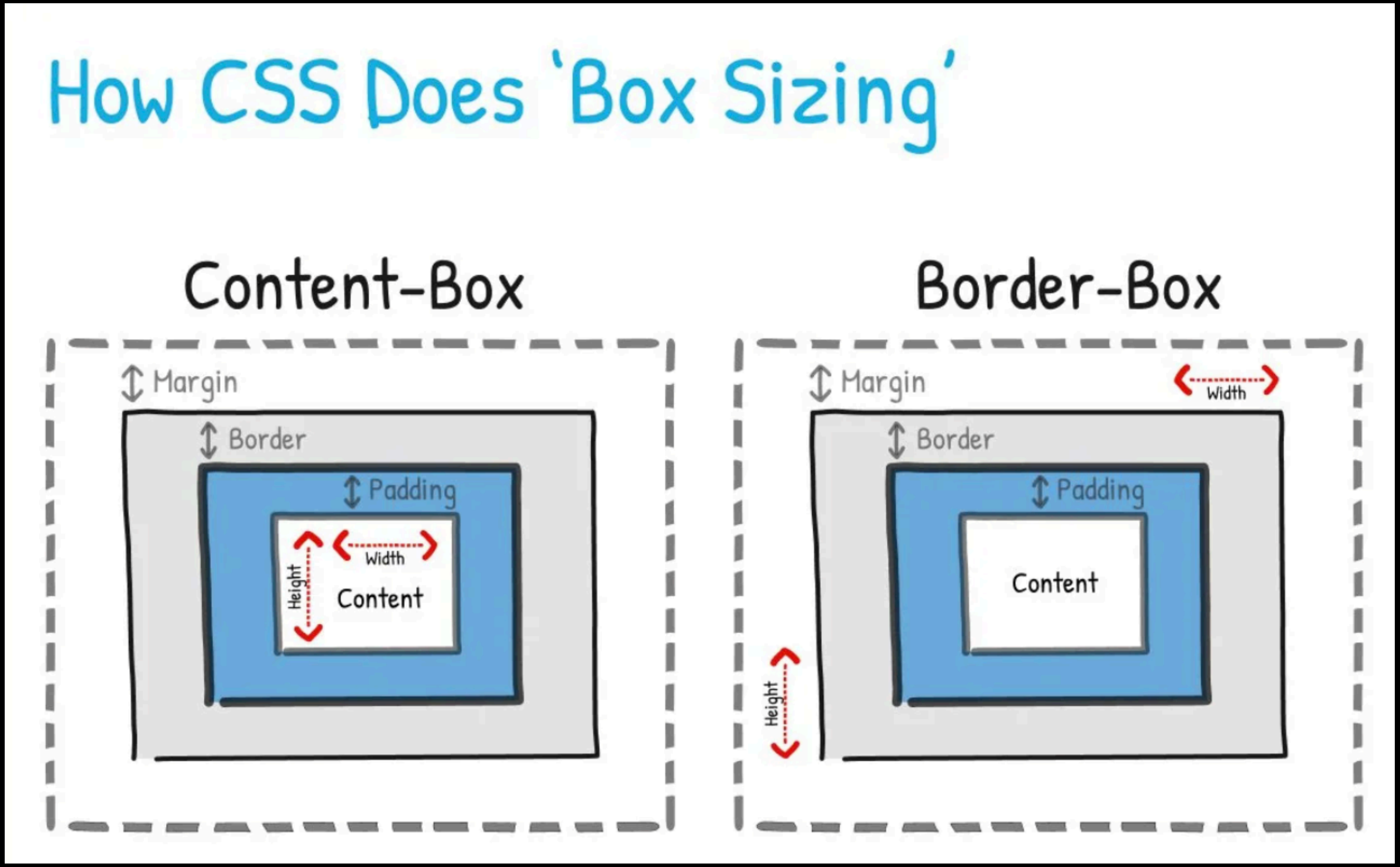
- Everything is a rectangular box
- The box has a content
- The box has padding
- The box has a border
- The box has margin
- boxes can be inside of boxes
- https://www.w3schools.com/css/css_boxmodel.asp



CSS

Box Model - Details

- The attribute of „box-sizing“

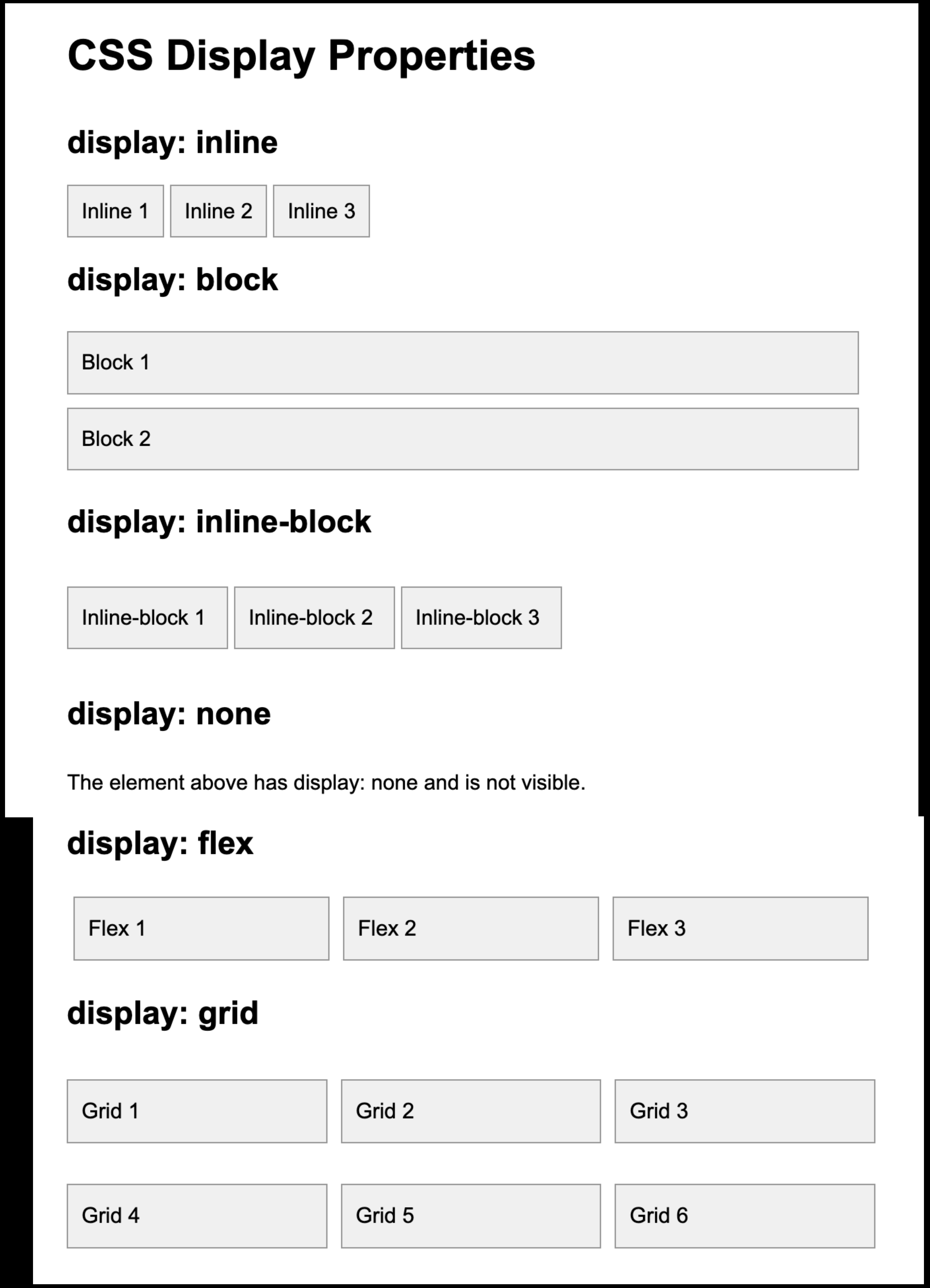


- https://www.reddit.com/r/css/comments/plj16h/what_is_boxsizing_in_css_how_does_it_work/

CSS

Display &display.html

- display: inline: Elements are displayed inline, allowing other elements to sit beside them.
- display: block: Elements take up the full width available and start on a new line.
- display: inline-block: Elements are inline but can have width and height properties.
- display: none: The element is completely removed from the layout.
- display: flex: Creates a flexible container for laying out child elements.
- display: grid: Creates a grid container for a two-dimensional layout system.



CSS Display Properties

display: inline

Inline 1 Inline 2 Inline 3

display: block

Block 1

Block 2

display: inline-block

Inline-block 1 Inline-block 2 Inline-block 3

display: none

The element above has display: none and is not visible.

display: flex

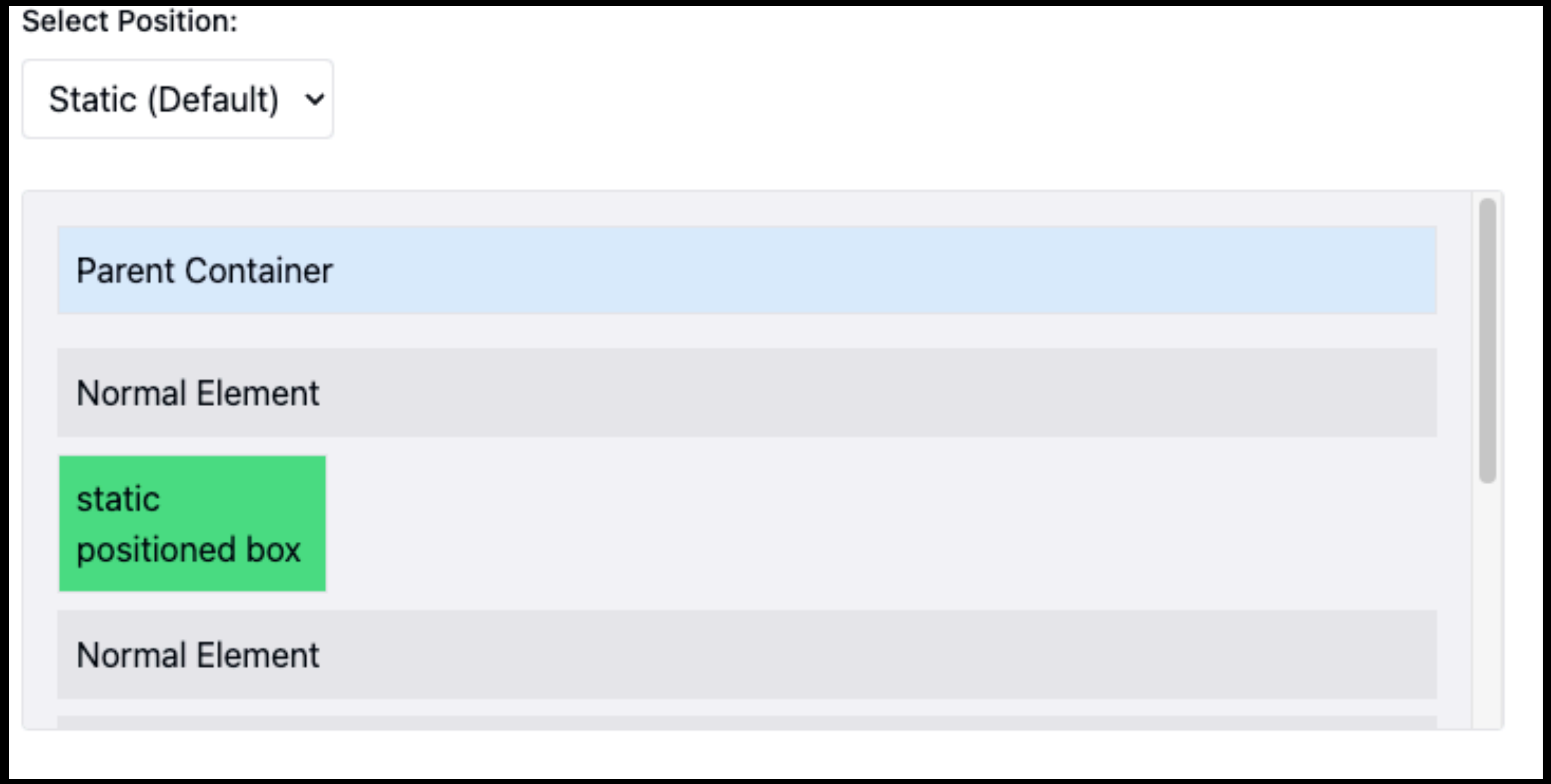
Flex 1 Flex 2 Flex 3

display: grid

Grid 1	Grid 2	Grid 3
Grid 4	Grid 5	Grid 6

CSS

lets talk about positions baby!!! &positions.html



Select Position:

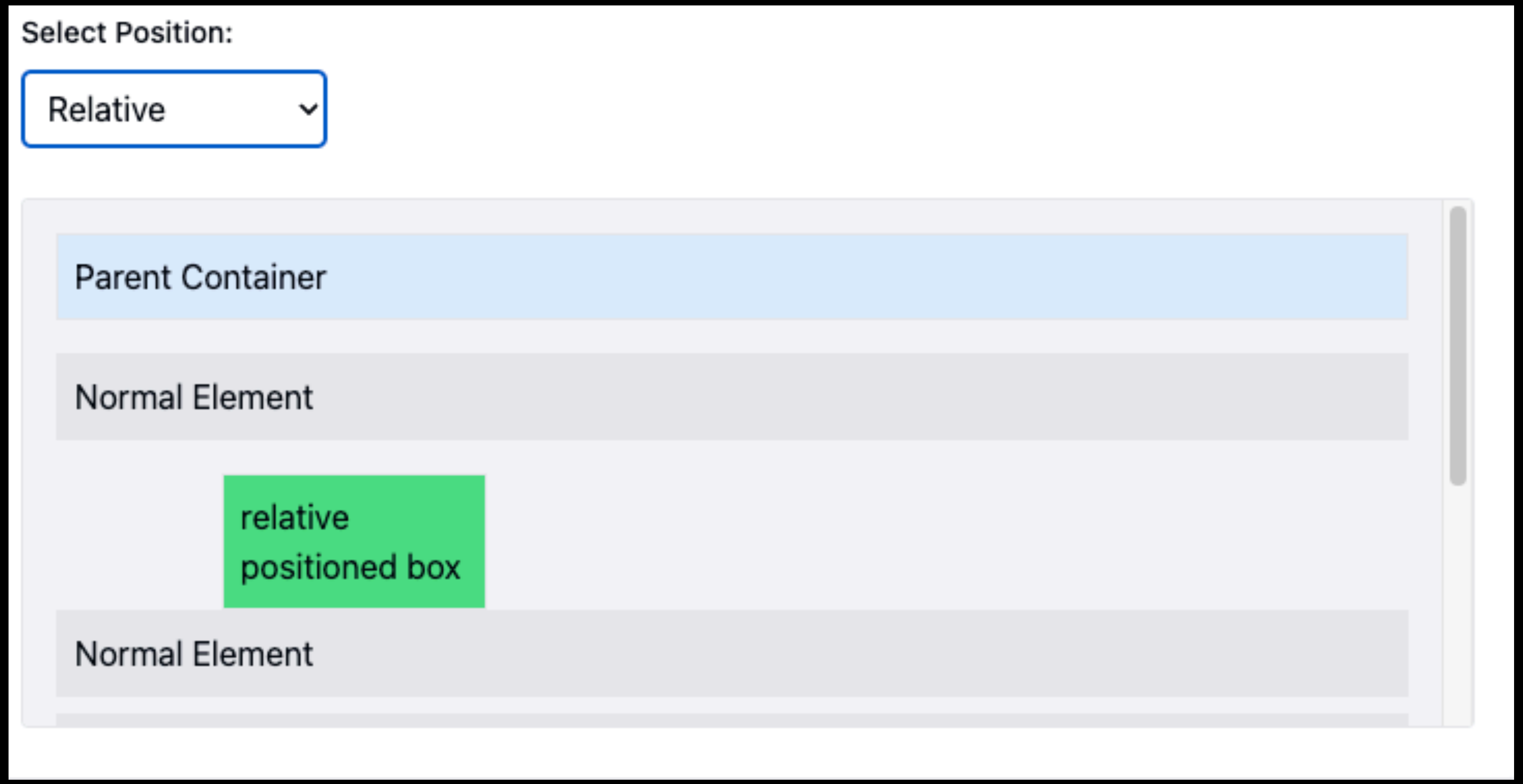
Static (Default)

Parent Container

Normal Element

static positioned box

Normal Element



Select Position:

Relative

Parent Container

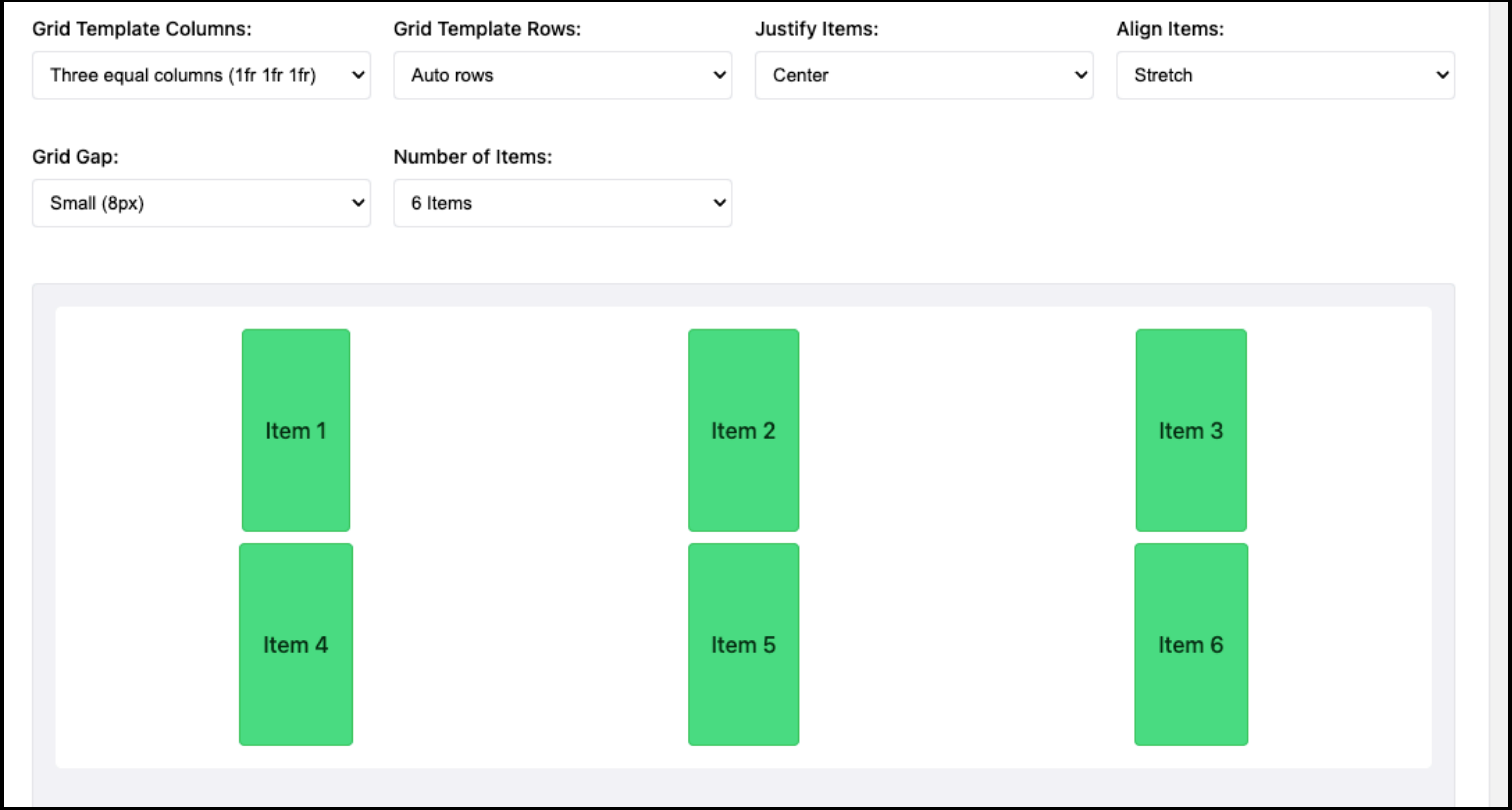
Normal Element

relative positioned box

Normal Element

CSS Layout Tools

Grids &grid.html



Grid Template Columns: Three equal columns (1fr 1fr 1fr)

Grid Template Rows: Auto rows

Justify Items: Center

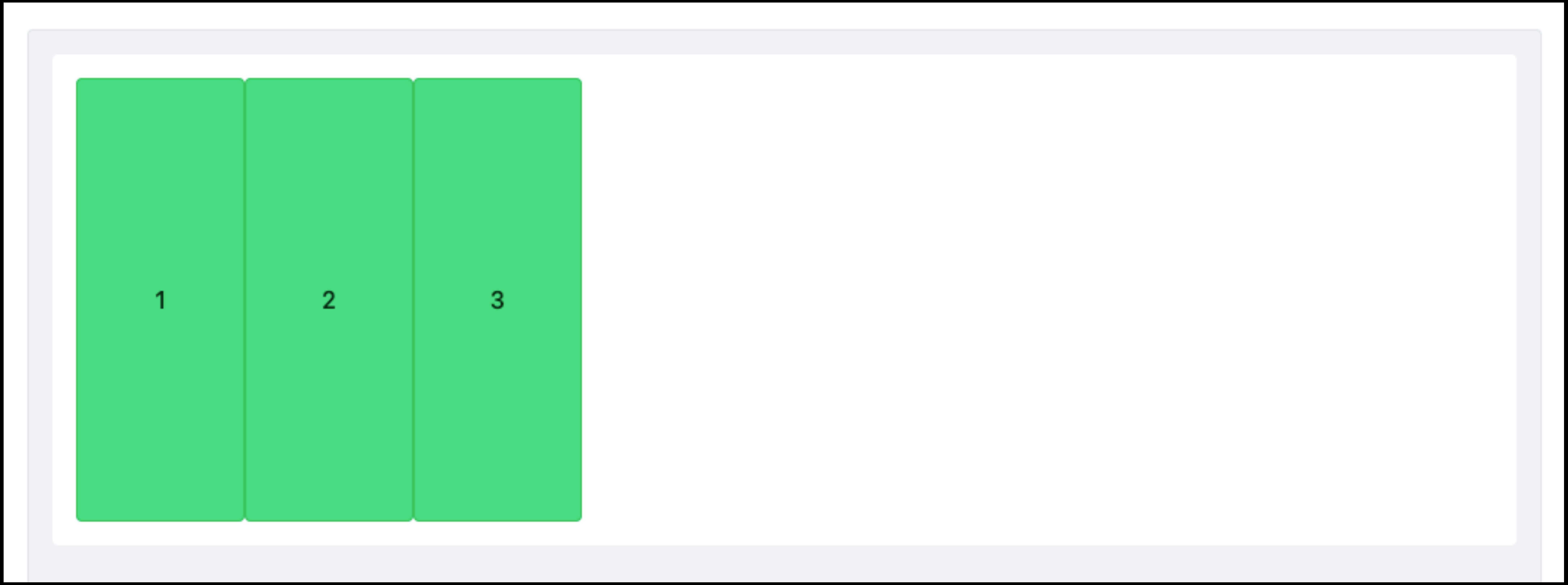
Align Items: Stretch

Grid Gap: Small (8px)

Number of Items: 6 Items

CSS Layout Tools

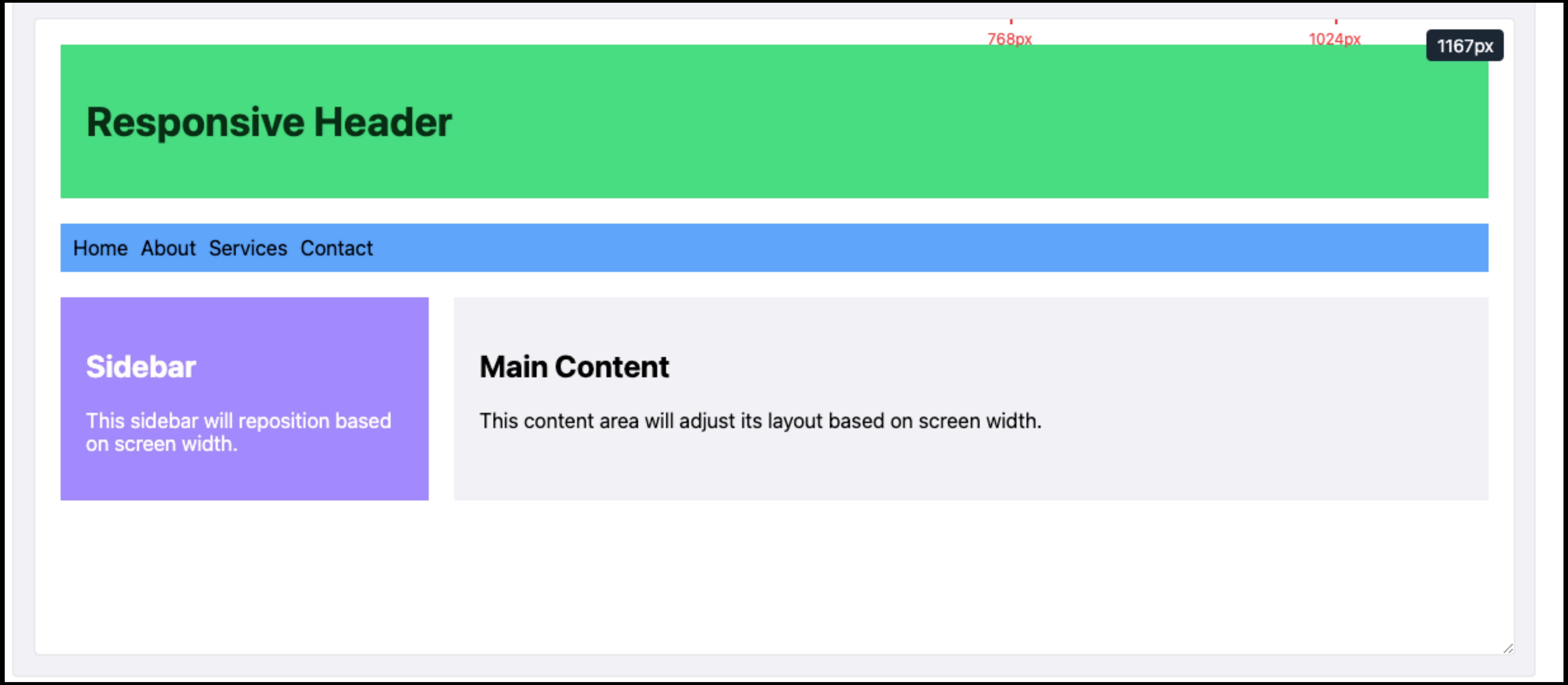
Flex boxes &flex.html



CSS Media Queries

But wait ….. the Web got modern…… What about the iPhone….. &media-query.html

- media queries help making layouts dependent and flexible with different Browser or device sizes



Responsive Header

Home About Services Contact

Sidebar

This sidebar will reposition based on screen width.

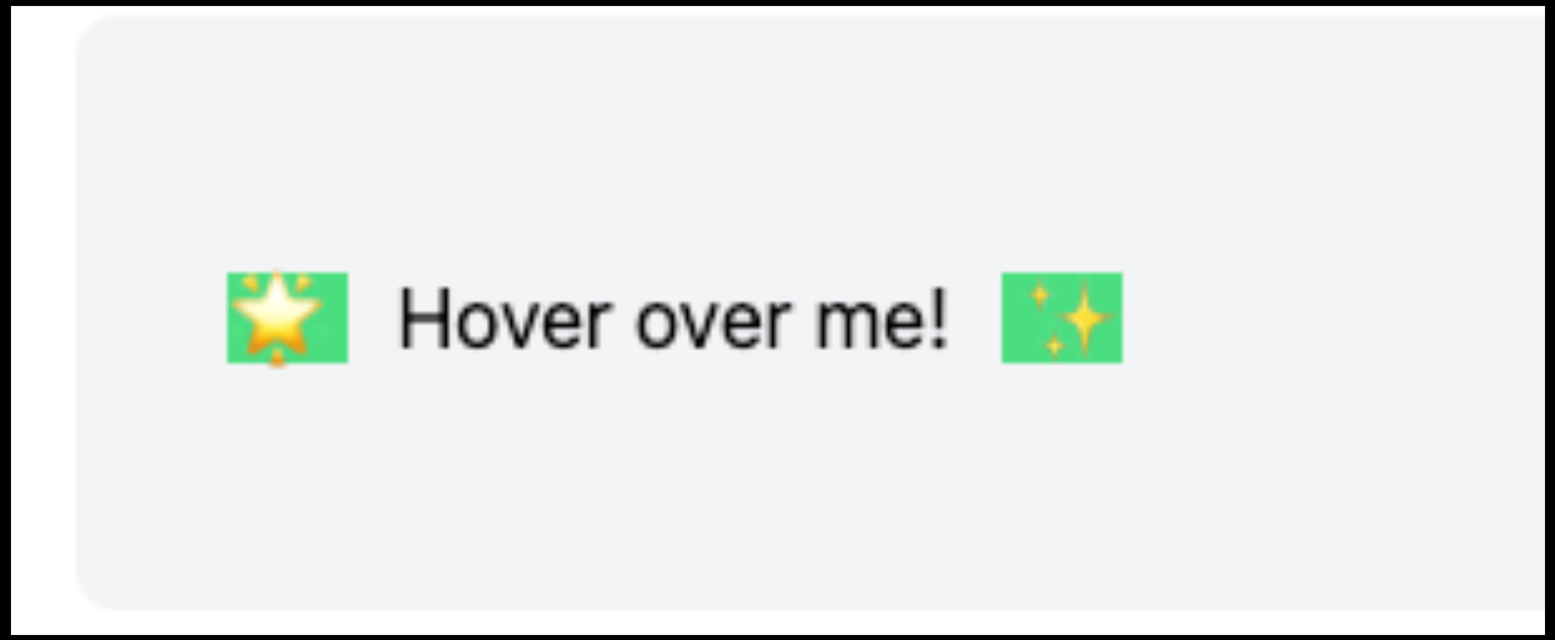
Main Content

This content area will adjust its layout based on screen width.

CSS Pseudo Elements &pseudo-elements.html

We are out of elements….. damn prebuild sh*t!!!

- Pseudo Elements
- Customize Elements with more flexibility
- e.g. custom tooltips https://www.w3schools.com/howto/howto_css_tooltip.asp



Now we are looking better but what about custom logic and interaction…

We don’t compile, we run a text based script. javascript…

JS

Make it interactive…

- Installed an run in Browser environment (not NodeJS)
- loosely typed

```
let age = 25;             // number
let name = 'Foo Bar';     // string
```

HTML Formatting

include Assets

```
<head>
    <!—— Everything else needed in your head ->
    <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="https://ultimate-cdn/external.css">
</head>
<body>
    <!—— Your page content ->
    <script src=„./index.js"></script>
    <script src="https://ultimate-cdn/external.js"></script>
</body>
</html>
```

```
A CDN (Content Delivery Network) might provide text/plain resources
https://getbootstrap.com/docs/3.3/getting-started/
```

`https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap-theme.min.css`

JS

Make it interactive…

- Include in .js files into html by script tag
- or do it inline (in the html file itself

```
<!DOCTYPE html>
<html>
<head>
    <title>Inline JavaScript Example</title>
</head>
<body>
    <h1>Welcome to JavaScript</h1>
    <button onclick="showMessage()">Click me!</button>

    <script>
        function showMessage() {
            alert('Hello from inline JavaScript!');
        }
    </script>
</body>
</html>
```

Selectors &selectors.txt

Pick it!

```
<div class="container"></div>       div.container    or      .container

<p id="intro"></p>                  p#intro          or      #intro

<ul>                                ul > li.item  or  ul li.item   li.item        or         .item
    <li class="item">Item 1</li>
    <li class="item">Item 2</li>
</ul>

<button type="button">Click me</button>   button[type=button]        or      button

<input type="text" name="firstname">   input[type=text]        or     input[name=firstname] or

                                          input[type=text][name=firstname]
```

JS

Make it interactive…

- use selectors like in CSS

```
document.querySelector('button').addEventListener('click', function() {
    alert('Hello, JavaScript!');
});
```

```
document.querySelector('p.interactive').addEventListener('click', function() {
    alert('Hello, JavaScript!');
});
```

JS &js.txt &js-dom.txt

The DOM - or a bunch of nested objects…. (oh noooo lists again) &the-dom.html

- Each HTML element is a Node in a Node Tree (written in XML Format)
- Some Nodes can habe a limited, no, or a fixed number of children
- JS can move, delete, edit, create them according to the rules of the HTML definition
- The DOM contains additional Information swell (document varible) and there are additional Variables like window or available objects like the JSON object
- They provide interaction interfaces e.g. to the Browser itself or helping in working with data

JS

Select elements in the DOM &js-selectors.html

- document.querySelector(“p“) -> selects first p it will find
- document.querySelectorAll(“p“) -> selects all p’s in the whole document
- document.querySelectorAll(“p.active“) -> selects all p’s in the whole document that have the class „active“

JS

Do something with elements in the DOM

```
var   myparagraph = document.querySelector(„p")

myparagraph.innerText = "My new JS updated Text!"
myparagraph.innerHTML = "<strong>fat text here!</strong>“

myparagraph.classList.add("active")
myparagraph.classList.remove(„active")

myparagraph.getAttribute("data-format")
myparagraph.setAttribute(„data-format")

myparagraph.style.color = "blue"
myparagraph.style.fontSize = "14px"
```

JS

Create or move elements in the DOM

```
var newElement = document.createElement("span");
newElement.innerText = "New Text!";
myparagraph.appendChild(newElement);
```

JS

Create or move elements in the DOM

```
// Two existing elements in the DOM
var firstElement = document.querySelector("#element1");
var secondElement = document.querySelector("#element2");

// Function to swap the two elements
function swapElements(el1, el2) {
    // Create a temporary placeholder
    var temp = document.createElement("div"); // Temporary placeholder

    el1.parentNode.insertBefore(temp, el1);  // Insert temp where el1 is
    el2.parentNode.insertBefore(el1, el2);   // Move el1 to where el2 is
    temp.parentNode.insertBefore(el2, temp); // Move el2 to where temp is

    temp.parentNode.removeChild(temp);       // Remove the temporary placeholder
}

// Call the function to swap
swapElements(firstElement, secondElement);
```

JS

Move it smart and add a prototype function

```
// Adding a custom swap function to the Element prototype
Element.prototype.swapWith = function(otherElement) {

    var temp = document.createElement("div");

    this.parentNode.insertBefore(temp, this);

    otherElement.parentNode.insertBefore(this, otherElement);

    temp.parentNode.insertBefore(otherElement, temp);

    temp.parentNode.removeChild(temp);
};

// Example usage
var element1 = document.querySelector("#element1");
var element2 = document.querySelector("#element2");

// Now you can call swapWith() directly on any element
element1.swapWith(element2);
```

JS

Becoming an event manager

```
// INITIAL DOCUMENT EVENT

// Document is completely accessible for js
document.addEventListener("DOMContentLoaded", function() {
    console.log(„document is ready");
});
```

JS

Becoming an event manager

```
// MOUSE EVENTS

// Click event - Fires when an element is clicked
element.addEventListener("click", function() {
    console.log("Element clicked!");
});

// Double click event - Fires when an element is double-clicked
element.addEventListener("dblclick", function() {
    console.log("Element double-clicked!");
});

// Mouse over event - Fires when the mouse moves over an element
element.addEventListener("mouseover", function() {
    console.log("Mouse over element!");
});

// Mouse out event - Fires when the mouse leaves an element
element.addEventListener("mouseout", function() {
    console.log("Mouse out of element!");
});

// Mouse move event - Fires when the mouse moves inside an element
element.addEventListener("mousemove", function(event) {
    console.log("Mouse moved at: ", event.clientX, event.clientY);
});
```

JS

Becoming an event manager

```
// KEYBOARD EVENTS

// Key down event - Fires when a key is pressed down
document.addEventListener("keydown", function(event) {
    console.log("Key pressed: ", event.key);
});

// Key up event - Fires when a key is released
document.addEventListener("keyup", function(event) {
    console.log("Key released: ", event.key);
});

// Key press event (deprecated in some cases) - Fires when a key is pressed and released
document.addEventListener("keypress", function(event) {
    console.log("Key pressed and released: ", event.key);
});
```

JS

Becoming an event manager

```
// FORM EVENTS

// Submit event - Fires when a form is submitted
form.addEventListener("submit", function(event) {
    event.preventDefault(); // Prevents default form submission
    console.log("Form submitted!");
});

// Change event - Fires when the value of an input/select changes
inputElement.addEventListener("change", function() {
    console.log("Input value changed!");
});

// Input event - Fires when typing into a text input
inputElement.addEventListener("input", function() {
    console.log("Input is being typed: ", inputElement.value);
});
```

JS

Becoming an event manager

```
// WINDOW EVENTS

// Load event – Fires when the entire page has fully loaded
window.addEventListener("load", function() {
    console.log("Page fully loaded!");
});

// Resize event – Fires when the window is resized
window.addEventListener("resize", function() {
    console.log("Window resized to: ", window.innerWidth, window.innerHeight);
});

// Scroll event – Fires when the user scrolls the page or an element
window.addEventListener("scroll", function() {
    console.log("Page scrolled!");
});
```

JS

Becoming an event manager

```
// CLIPBOARD EVENTS

// Copy event - Fires when content is copied
document.addEventListener("copy", function() {
    console.log("Content copied!");
});

// Paste event - Fires when content is pasted
document.addEventListener("paste", function(event) {
    console.log("Content pasted: ", event.clipboardData.getData('text'));
});
```

JS

Becoming an event manager

```
// TOUCH EVENTS (for mobile)

// Touch start event - Fires when a touch point is placed on the screen
element.addEventListener("touchstart", function() {
    console.log("Touch started!");
});

// Touch move event - Fires when a touch point is moved across the screen
element.addEventListener("touchmove", function() {
    console.log("Touch is moving!");
});

// Touch end event - Fires when a touch point is removed from the screen
element.addEventListener("touchend", function() {
    console.log("Touch ended!");
});
```

JS

Modules, Libraries and fancy stuff: e.g. Observers

```javascript
// Select the element you want to observe
var targetElement = document.querySelector("#target");

// Create an IntersectionObserver instance
var observer = new IntersectionObserver(function(entries, observer) {
    entries.forEach(function(entry) {
        if (entry.isIntersecting) {
            console.log("Element is in view!");

            // Perform an action when the element is in view (e.g., lazy loading an image)
            entry.target.style.backgroundColor = "lightgreen";  // Change background as an example

            // Optionally, stop observing after the action is triggered
            observer.unobserve(entry.target);
        }
    });
}, {
    root: null,    // Defaults to the viewport
    threshold: 0.5  // Trigger when 50% of the element is in view
});

// Start observing the target element
observer.observe(targetElement);
```

JS

Virtual DOM - or how modern JS Framworks work

```javascript
// The "virtual DOM" representation (as a plain object)
const virtualDOM = {
    tagName: 'div',
    attributes: { id: 'app' },
    children: [
        {
            tagName: 'h1',
            attributes: { style: 'color: blue;' },
            children: ['Hello, Virtual DOM!']
        },
        {
            tagName: 'p',
            attributes: {},
            children: ['This is a simple virtual DOM example.']
        }
    ]
};
```

JS

Virtual DOM - or how modern JS Framworks work

```javascript
// Function to render the Virtual DOM to the real DOM
function renderElement(virtualNode) {
    const element = document.createElement(virtualNode.tagName);

    // Add attributes
    for (let key in virtualNode.attributes) {
        element.setAttribute(key, virtualNode.attributes[key]);
    }

    // Render children
    virtualNode.children.forEach(child => {
        if (typeof child === 'string') {
            element.appendChild(document.createTextNode(child));
        } else {
            element.appendChild(renderElement(child)); // Recursive call for nested elements
        }
    });

    return element;
}

// Initial render of Virtual DOM to real DOM
const realDOM = renderElement(virtualDOM);
document.body.appendChild(realDOM);
```

JS

Virtual DOM - or how modern JS Framworks work

```
// Initial render of Virtual DOM to real DOM
const realDOM = renderElement(virtualDOM);
document.body.appendChild(realDOM);

// New virtual DOM after a state change
const updatedVirtualDOM = {
    tagName: 'div',
    attributes: { id: 'app' },
    children: [
        {
            tagName: 'h1',
            attributes: { style: 'color: red;' },  // Changed color to red
            children: ['Hello, Updated Virtual DOM!']  // Updated text
        },
        {
            tagName: 'p',
            attributes: {},
            children: ['This content has been updated.']
        }
    ]
};

// Diffing algorithm to update the real DOM (simple example)
function updateElement(parent, oldNode, newNode, index = 0) {
    if (!oldNode) {
        // New node
        parent.appendChild(renderElement(newNode));
    } else if (!newNode) {
        // Remove node
        parent.removeChild(parent.childNodes[index]);
    } else if (typeof newNode === 'string' && typeof oldNode === 'string') {
        if (newNode !== oldNode) {
            parent.childNodes[index].nodeValue = newNode;
        }
    } else if (newNode.tagName !== oldNode.tagName) {
        parent.replaceChild(renderElement(newNode), parent.childNodes[index]);
    } else {
        // Update attributes
        for (let key in newNode.attributes) {
            parent.childNodes[index].setAttribute(key, newNode.attributes[key]);
        }

        // Recursively update children
        const newLength = newNode.children.length;
        const oldLength = oldNode.children.length;

        for (let i = 0; i < newLength || i < oldLength; i++) {
            updateElement(parent.childNodes[index], oldNode.children[i], newNode.children[i], i);
        }
    }
}

// Apply updates (diffing the two virtual DOM trees)
updateElement(document.body, virtualDOM, updatedVirtualDOM);
```

```
// Initial render of Virtual DOM to real DOM
const realDOM = renderElement(virtualDOM);
document.body.appendChild(realDOM);

// New virtual DOM after a state change
const updatedVirtualDOM = {
    tagName: 'div',
    attributes: { id: 'app' },
    children: [
        {
            tagName: 'h1',
            attributes: { style: 'color: red;' },  // Changed color to red
            children: ['Hello, Updated Virtual DOM!']  // Updated text
        },
        {
            tagName: 'p',
            attributes: {},
            children: ['This content has been updated.']
        }
    ]
};

// Diffing algorithm to update the real DOM (simple example)
function updateElement(parent, oldNode, newNode, index = 0) {
    if (!oldNode) {
        // New node
        parent.appendChild(renderElement(newNode));
    } else if (!newNode) {
        // Remove node
        parent.removeChild(parent.childNodes[index]);
    } else if (typeof newNode === 'string' && typeof oldNode === 'string') {
        if (newNode !== oldNode) {
            parent.childNodes[index].nodeValue = newNode;
        }
    } else if (newNode.tagName !== oldNode.tagName) {
        parent.replaceChild(renderElement(newNode), parent.childNodes[index]);
    } else {
        // Update attributes
        for (let key in newNode.attributes) {
            parent.childNodes[index].setAttribute(key, newNode.attributes[key]);
        }

        // Recursively update children
        const newLength = newNode.children.length;
        const oldLength = oldNode.children.length;

        for (let i = 0; i < newLength || i < oldLength; i++) {
            updateElement(parent.childNodes[index], oldNode.children[i], newNode.children[i], i);
        }
    }
}

// Apply updates (diffing the two virtual DOM trees)
updateElement(document.body, virtualDOM, updatedVirtualDOM);
```

JS

JSON

```
{

   "id": 1,
   "name": "Test event",
   "description": "A test event",
   "location": "A test location",
   „date_time": "2024-01-01T15:30:00Z",
   „user_id": 1
}
```

```
"{\"id\":1,\"name\":\"Test event\",\"description\":\"A test event\",\"location\":\"A test location\",
\"date_time\":\"2024-01-01T15:30:00Z\",\"user_id\": 1}"
```

JS

JSON

```javascript
// Step 1: Create a JavaScript object
const user = {
    name: "Alice",
    age: 30,
    email: "alice@example.com",
    isActive: true,
    hobbies: ["reading", "traveling", "cooking"]
};

// Step 2: Convert the JavaScript object to a JSON string using JSON.stringify()
const jsonString = JSON.stringify(user);
console.log("JSON String:", jsonString);
// Output: JSON String: {"name":"Alice","age":30,"email":"alice@example.com","isActive":true,"hobbies":["reading","traveling","cooking"]}

// Step 3: Convert the JSON string back to a JavaScript object using JSON.parse()
const parsedUser = JSON.parse(jsonString);
console.log("Parsed User Object:", parsedUser);
// Output: Parsed User Object: { name: 'Alice', age: 30, email: 'alice@example.com', isActive: true, hobbies: [ 'reading', 'traveling', 
'cooking' ] }

// Step 4: Access properties of the parsed object
console.log("User Name:", parsedUser.name); // Output: User Name: Alice
console.log("User Age:", parsedUser.age);   // Output: User Age: 30
```

JS Promises

Got to catch them all!

- JS is single Threaded but can handle asynchronous tasks
- I/O Behavoir like with Requests and Responses

JS Promises

Got to catch them all!

```
// Function that returns a Promise
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const success = true; // Simulating success or failure

            if (success) {
                resolve("Data fetched successfully!"); // Resolve with data
            } else {
                reject("Error fetching data."); // Reject with error message
            }
        }, 2000); // Simulate a 2-second network request
    });
}

// Using the Promise
fetchData()
    .then((result) => {
        console.log(result); // Handle success
    })
    .catch((error) => {
        console.error(error); // Handle error
    });

console.log("Fetching data..."); // This line runs immediately
```

JS Promises

Got to catch them all!

```
// Function that returns a Promise
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const success = true; // Simulate success or failure

            if (success) {
                resolve("Data fetched successfully!"); // Success
            } else {
                reject("Error fetching data."); // Error
            }
        }, 2000); // Simulate a 2-second network request
    });
}

// Async function
async function getData() {
    try {
        console.log("Fetching data...");
        const result = await fetchData(); // Wait for the Promise to be fulfilled
        console.log(result); // Successful response
    } catch (error) {
        console.error(error); // Error handling
    }
}

// Calling the async function
getData();
```

JS

AJAX

```
// Async function
async function getData() {
    try {
        console.log("Fetching data...");
        const result = await fetchData(); // Wait for the Promise to be fulfilled
        console.log(result); // Successful response
    } catch (error) {
        console.error(error); // Error handling
    }
}

// Calling the async function
getData();
```

JS

AJAX

```
// Creating an XMLHttpRequest object
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://jsonplaceholder.typicode.com/todos/1", true);

// Sending the asynchronous request
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && xhr.status == 200) {
        // Successful response received
        console.log(JSON.parse(xhr.responseText)); // Process the response here
    }
};
xhr.send();
```

JS

AJAX

```javascript
// Asynchronous HTTP request with fetch (uses Promises)
fetch("https://jsonplaceholder.typicode.com/todos/1")
    .then(response => response.json())  // Parse the response
    .then(data => console.log(data))    // Process the data
    .catch(error => console.error("Error:", error));  // Error handling
```

```javascript
    async function fetchData() {
        try {
            const response = await fetch("https://jsonplaceholder.typicode.com/todos/1");
            const data = await response.json();
            console.log(data);  // Process the data
        } catch (error) {
            console.error("Error:", error);  // Error handling
        }
    }

    fetchData();
```

JS

Modular notation

```javascript
const MyModule = (function factory() {
    'use strict';

    let isInitialized = false;
    let config = null;

    function memberFunction() {
        if (!isInitialized) throw new Error('Not initialized');
        return 'member function called';
    }

    function helperFunction() {
        return 'helper function called';
    }

    // Public API
    return {
        init: function(userConfig) {
            if (isInitialized) return;
            config = userConfig;
            isInitialized = true;

           memberFunction();
            helperFunction();
        }
    };
}());
```

Objectoriented JS

```javascript
// Define the class
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    greet() {
        console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
    }

    haveBirthday() {
        this.age += 1;
        console.log(`Happy Birthday! ${this.name} is now ${this.age} years old.`);
    }
}

// Using the class
const person = new Person("Alice", 30);
person.greet();          // Output: Hello, my name is Alice and I am 30 years old.
person.haveBirthday();   // Output: Happy Birthday! Alice is now 31 years old.
```

Objectoriented JS

Encapsulation

```javascript
// 1. Using # for private fields (Modern JS)
class ModernClass {
    #privateField = 'private';
    #privateMethod() {
        return this.#privateField;
    }

    publicMethod() {
        return this.#privateMethod();
    }
}

// 2. Using closures for privacy (Traditional approach)
const TraditionalClass = (function() {
    'use strict';

    const privateField = new WeakMap();

    class MyClass {
        constructor() {
            privateField.set(this, 'private');
        }

        publicMethod() {
            return privateField.get(this);
        }
    }

    return MyClass;
})();
```

Objectoriented JS

Encapsulation

```javascript
// 3. Using underscore convention (not true privacy)
class ConventionClass {
    constructor() {
        this._notReallyPrivate = 'pseudo-private';
    }

    _pseudoPrivateMethod() {
        return this._notReallyPrivate;
    }
}

// 4. Using Symbol for semi-private fields
class SymbolClass {
    constructor() {
        this[SymbolClass.privateField] = 'private';
    }

    static privateField = Symbol('privateField');

    publicMethod() {
        return this[SymbolClass.privateField];
    }
}
```

Typescript

There really is a strict JS

```typescript
class UserService {
    // Private properties
    private readonly apiKey: string;
    private users: User[] = [];

    // Private interface
    private interface User {
        id: number;
        name: string;
        email: string;
    }

    // Constructor
    constructor(apiKey: string) {
        this.apiKey = apiKey;
    }

    // Private method
    private validateUser(user: User): boolean {
        return user.name.length > 0 && user.email.includes('@');
    }

    // Public methods
    public addUser(user: User): boolean {
        if (!this.validateUser(user)) {
            return false;
        }
        this.users.push(user);
        return true;
    }

    public getUser(id: number): User | undefined {
        return this.users.find(user => user.id === id);
    }

    public getAllUsers(): User[] {
        return [...this.users];
    }
}
```

Frameworks: JS

Angular, Vue, React, …

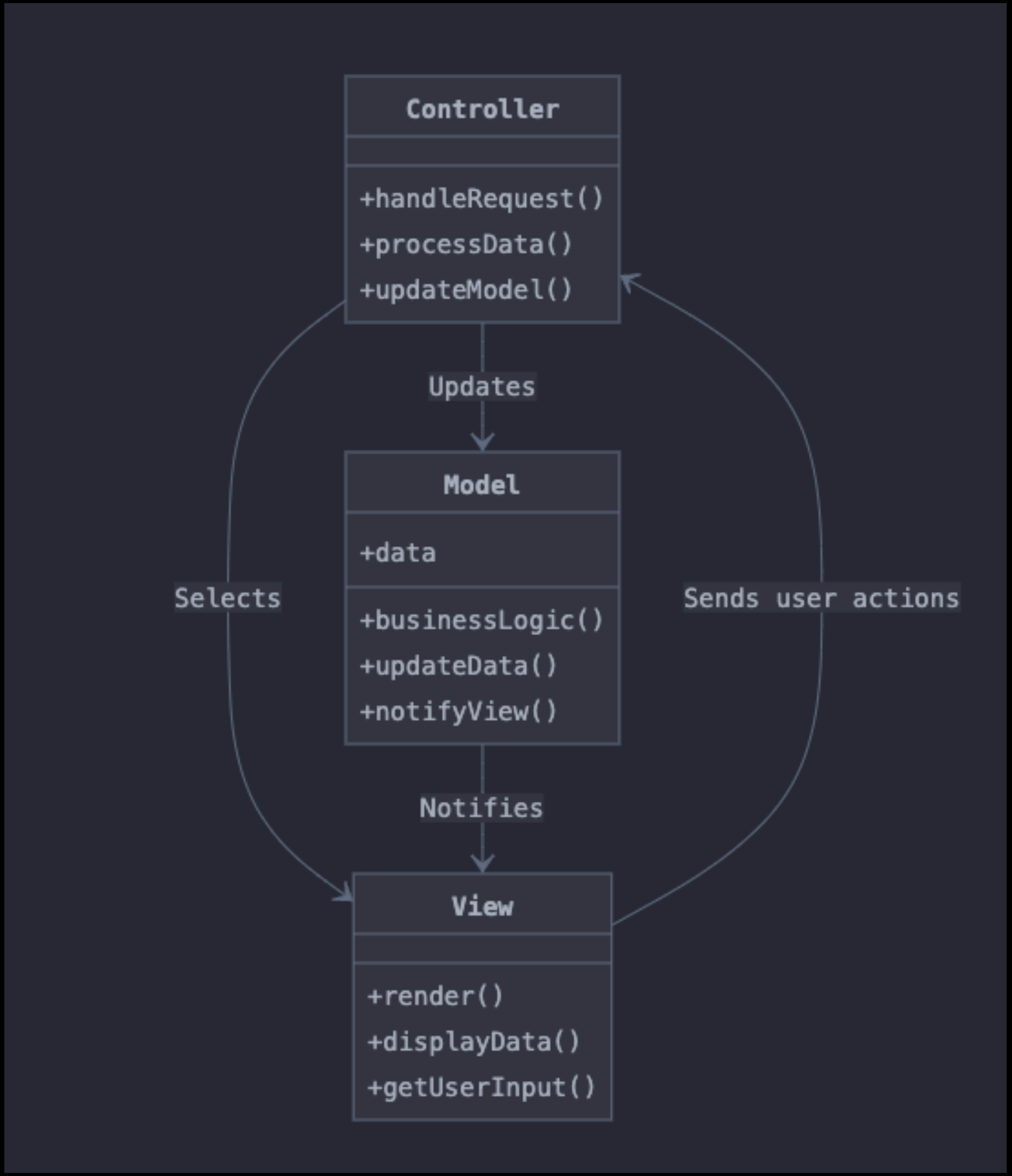
- Are using virtual DOMs and require understanding of their own concepts and structures
- General environment like REST Principles, HTTP(S) etc. stay the same though and are handled in a specific way
- Most of them setup logical groups of elements into Components
- Components have encapsulated logic patterns and interfaces that they exchange public informations
- Lets say they follow a object oriented approach

Backend

Go, PHP, Java, Python or similar…

- Provide Logic to handle Requests and answer in the required format (text, html, json)
- They can either host the application itself with static html pages or provide an api, that interacts with the virtual DOM

MVC Structure



Thanks and good luck…

I recommend a Backend Job :)

Jonathan Eberle - September 2024