0 Contents

1 NodeJS and other WebServers

This is the interface between the operating system and the actual website itself. -> Apache

- Runs everywhere
- async with events
- easy to deploy
- Lots of APIs for different usecases
  HTTP/HTTPS, URL, FileSystem, Console, UDP, Cryptography, DNS
- modular
- not that much "magic"
  you need to write more yourself

negative: IT IS JS, FFS...

1.1 Async via events

The idea of events is *essentially a queue that doesn't block actual functionality*, this means that your UI still does what it should, as it will be prioritized compared to async functionaity.
Just like bevy, the events are handled in the next frame, or more precise *whenever there is nothing else to do*.
This means we will get the illusion of async/multi-threading, without js actually being able to do so.

1.2 Callback hell with async events

If you constantly need to check for finished events, then you will end up with unreadable code.
The solution is simple: Promises
The default in NodeJS is still callbacks, but there is work on this to change.
Example for callback

```
button.addEventListener('click', function (event) {
  console.log("1. subscription");
});
```

1.3 Example for NodeJS

```
const http = require('node:http'); // lua?

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200; // HTTP response
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

2 Modules

2.1 NPM

- npm init
  - -g for global installation
  - –save-dev for a module that will only be used during development
-
- item 3
- item 4

2.2 Exporting Modules

```
function add() { return ++counter; }
function get() { return counter; }
// default export
export default {count: add, get: get};
// named
export {add, get};
```

2.3 Importing Modules

```
// default
import counterA from './counter.mjs';
// Named
import {add, get} from './counter.mjs';
```

2.4 package.json

```
{
  "name": "my_package",
  "description": "",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
```

```
    "url": "https://github.com/mgfeller/my_package.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/mgfeller/my_package/issues"
  },
  "homepage": "https://github.com/mgfeller/my_package"
}
```

2.5 package-lock.json

This handles dependency problems, where one dependency will be updated but the first package still wants an older one.

```
{
  "name": "my_package",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "fancy-calc-demo": {
      "version": "5.0.2",
      "resolved": "https://registry.npmjs.org/fancy-calc-demo/-/fancy-
      calc-demo-5.0.2.tgz",
      "integrity": "sha512-
      93xBMjZMU6HfGLXlwi1uYtQKL6eiNqOsWqkuFmlWMGJqKVBMF3+jb/dSrsRHrvpplIIDxU
      klwLNUzeZ5BTMjwQ=="
    }
  }
}
```

This *must also be in the git repository!*

2.6 Native Modules

These are modules that are performance critical and are therefore written in other languages such as c++.
The problem with this is that you will now need OS specific versions again!

2.7 nvm/nodenv

This is a tool to handle different node versions, you prob don't need this crap on arch, btw...

2.8 API versions

typically you will have 2 versions of a specific API, one that is synchronous, and one that is async.
The async will not throw an exception, as it is either handled with callbacks or with promises, the synchronous variant however will throw an exception if it fails.

```
// async
let fs = require('fs');
let path = "test.txt";
fs.readFile(path, function(err, content) {
  if (err) return console.error(err);
  console.log('content of file: %s\n', path);
  console.log(content.toString());
});
// the synchronous variant
// readFileSync

// note this specific problem can be done better with streams, as you might have continous input/ouput
let server = http.createServer(function (req, res) {
  let stream = fs.createReadStream(__dirname + '/data.txt');
  stream.pipe(res);
});
```

2.9 EventEmitter

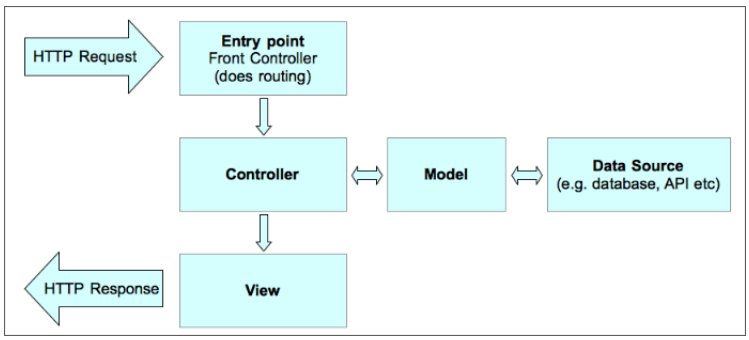
Similar to bevy, nodejs has an event emitter, which can then also be used in classes:

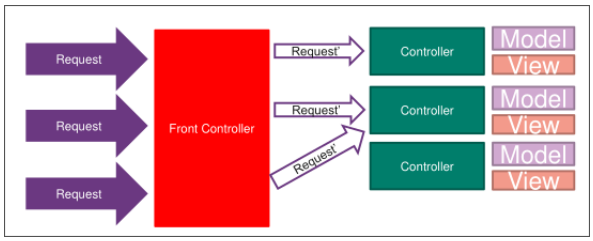
```
import {EventEmitter} from 'events'
export class Door extends EventEmitter {
  constructor() {
    super();
  }
  ring() {
    setTimeout(() => {
      this.emit('open');
    }, 1000);
  };
}
```

3 Web Framework Patterns

3.1 MVC

- Model
  Only responsible for the data
- View
  Only responsible for the presentation
- Controller
  The connection between the two





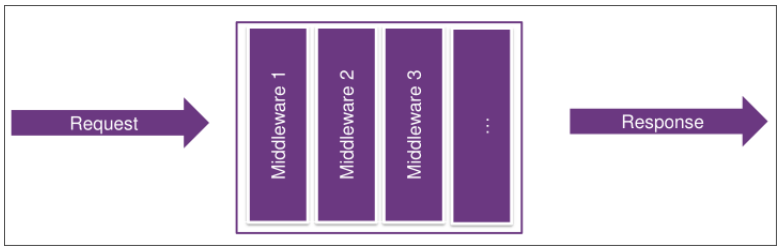
3.2 ExpressJS

- Most used framework
- Javascript
- rather old
- Integrated Middleware

Base Usage:

```
import http from 'http';
import express from 'express';

const app = express();
http.createServer(app).listen(3000);

// OR

import express from 'express';
const app = express();

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

3.3 Middleware



Modular components, that will handle individual tasks -> one for authentication, one for authorization.

- Authorization: What am I allowed to do?
- Authentication: Who am I?

3.3.1 Usage of Middlewares in ExpressJS

In order to register a middleware in expressJS, use the app.use(middleware) function.

```
import express from 'express';
import bodyParser from 'body-parser';

app.use(express.static(__dirname + '/public'));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(router);

const app = express();
const router = express.Router();
```

3.3.2 Middleware archive "Connect"

- body-parser
- cookie-parser
- cookie-session
- errorhandler
- method-override
- serve-static

3.3.3 ExpressJS defaults

Middlewares:

- Router
- Static

Server-Side-Rendering

Request extensions:

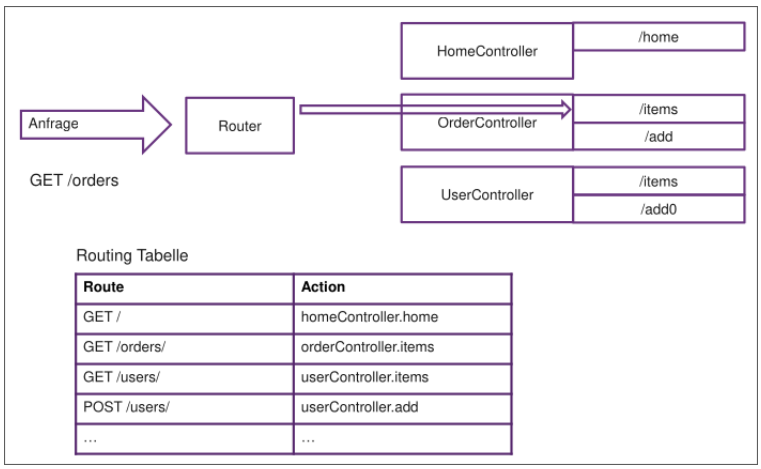
- params
- is() -> checks if content-type is correct
- get() -> check header

Response extensions:

- sendFile
- format
- json/jsonop

3.3.4 Routing

Routers handle requests with endings. This enables you to easily create single page application swith REST.



Setup:

```
import express from 'express';
const router = express.Router();
```

Example for *all methods*:

```
router.all('/', function(req, res){
res.send('hello world');
}); // will be called without checking the method
```

Example with a specific method

```
router.get('/', function(req, res){
res.send('hello world');
}); // will only be called if it is a get method
```

Example with multiple defined methods

```
app.route('/book')
.get(function(req, res) {res.send('Get a random book');})
.post(function(req, res) {res.send('Add a book');})
// used to map multiple methods to one route
```

3.3.5 Route Specifics

Router uses pattern matching, this means that you can check for things such as:

- /ab*de -> c or something else implied
- /* map to anything

You can use variables to store requests from users:

```
router.get('/something/:id', function(req, res) ... // this set the variable id as req.params.id
// note that the req is passed to the function in question, meaning you can handle that variable in that function!
```

3.3.6 Satic-Middleware

In order to always deliver something like a png, you can use static.

```
app.use(express.static(__dirname + '/public'))
```

3.3.7 Custom Middleware

In order to create a middleware, you need to use 3 functions/parameters.
The next() has to be called, in order to end the request, otherwise the page will load indefinitely!
Reponse and Request are represented as the res and req in the function.

```
function myDummyLogger( options ) {
  options = options ? options : {};
  return function myInnerDummyLogger(req, res, next) {
    console.log(req.method +":"+ req.url);
    next();
  }
}
```

3.3.8 Error Middleware

This should always be registered as the last middleware -> makes sure it catches all errors.
You can however define multiple error middlewares, the last one just needs to end the request.
Will be called if an "error-object" is passed to the next() function inside another middleware.

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Note for forms, you can only use the GET and POST methods in forms, this means that something like delete won't work!
It will be converted to GET requests!
As a solution, *use the POST method as a workaround.*

3.4 Databases and Frameworks

You can use any kind of databases like SQL -> Postgres, NoSQL, JSON, in memory databases etc.

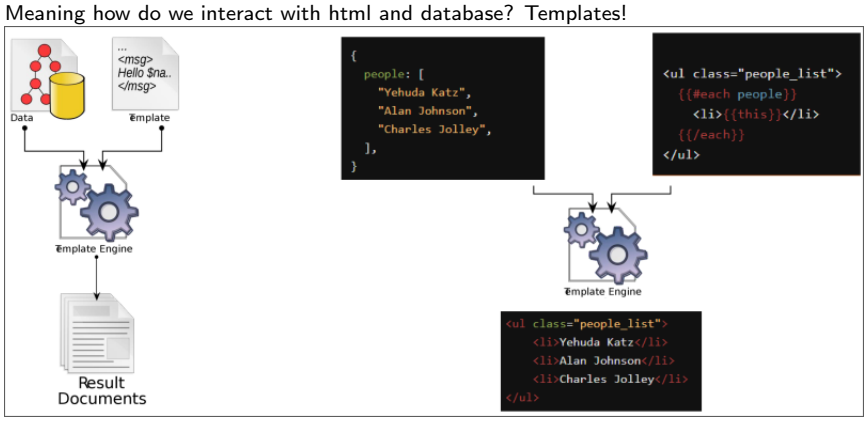
3.4.1 Example with nedb (NoSQL)

```
import Datastore from '@seald-io/nedb';
const db = new Datastore({filename: './data/order.db', autoload: true});

// insert
db.insert(order, function(err, newDoc){
  if(callback){
    callback(err, newDoc);
  }
```

```
});

// find or findOne
db.findOne({ _id: id }, function (err, doc) {
  callback( err, doc);
});

// update
db.update({_id: id}, {$set: {"state": "DELETED"}}, {}, function (err, doc) {
  publicGet(id,callback);
});
```

3.4.2 Templating engine with DatabaseTemplating engine with Databasess

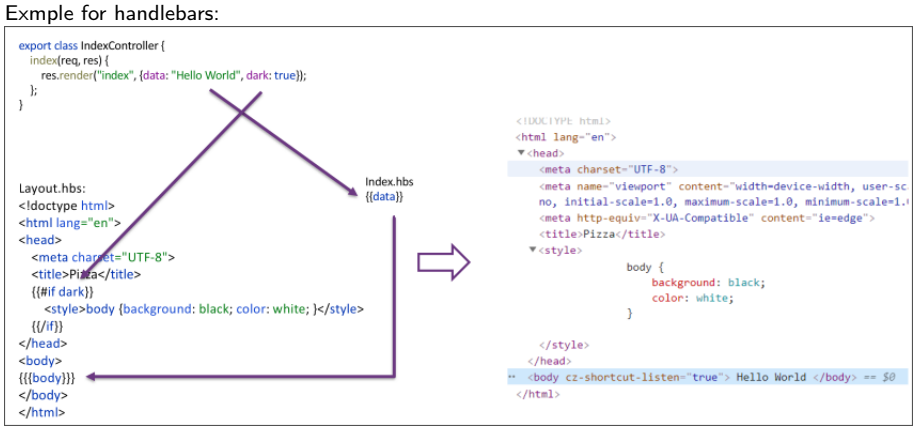
The question is, how do we generate html when we want to do it dynamically?
Meaning how do we interact with html and database? Templates!



- definition of look -> handlebar framework language
- definition of tables -> json
- caching
- Partials
- converts html and json to the result document.

3.4.3 Handlebars template engine

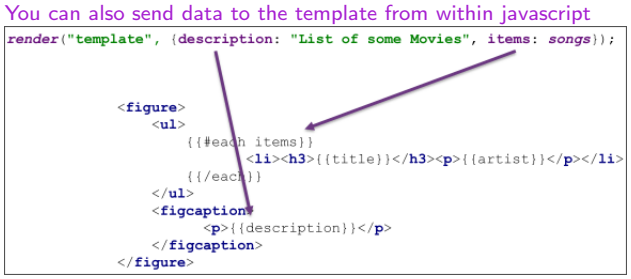
Exmple for handlebars:




```
// 1. import express-handlebars
import exphbs from 'express-handlebars';

const app = express();

// 2. configure
const hbs = exphbs.create({
  extname: '.hbs',
  defaultLayout: "default",
  helpers: {
    ...helpers
  }
});

// 3. set engine and global values
app.engine('hbs', hbs.engine);
app.set('view engine', 'hbs');

// 4. path to views
app.set('views', path.resolve('views'));

// usage afterwards
// app.render(view, [locals], callback)
createPizza = async (req, res) => {
  res.render("succeeded", await orderStore.add(req.body.name, "unkown"));
};

<p>Order-Informationen</p>
{{#if pizzaName}}
  <p>Bestellte Pizza: {{pizzaName}}</p>
  {{#if_eq state "OK"}}
    <form action='/orders/{{_id}}' method='post'><input type='hidden' name='_method' value='delete'><input
      type='submit' value='Delete order'></form>
  {{/if_eq}}
{{/if}}

// render
async showOrder(req, res) {
```

```
  res.render("showorder", await orderStore.get(req.params.id));
};
```

You can also send data to the template from within javascript



In loops, the current object is the current context:

```
<ul>
  {{#each items}}
  <li>
    <h3>{{title}}</h3>
    <p>{{this.artist}}</p>
  </li>
  {{/each}}
</ul>
// you can access the root with {{@root}}
```

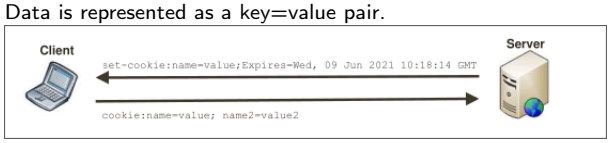
Handlebars only defines a few helpers (functions), but you can define some yourself!

```
//Usage: {{? hasError 'error' 'ok'}}
Handlebars.registerHelper('?', function(exp, value1, value2, options) {
  if(exp) {
    return value1;
  }
    return value2;
  });
{{? hasError 'FEHLER' 'OK' }}
```

3.5 Security with ExpressJS

3.5.1 Authentication with Cookies

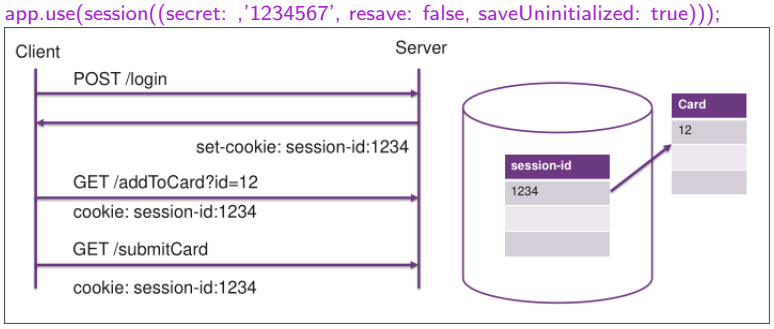
Cookies represent a small piece of data. This can be a state in which the website was in before the client left the last time.
Data is represented as a key=value pair.



Usage:
npm install cookie-parser -save
app.use(require("cookie-parser"));

3.5.2 Session with Cookies

Usage:
npm install session-parser -save
app.use(session((secret: ,'1234567', resave: false, saveUninitialized: true)));



3.5.3 Storing cookies

Cookies (more likely its data) are often stored in json, this can be done as follows

```
// valid but dangerous
JSON.stringify([1,2,4,5]);
// better:
json = JSON.stringify({elements : [1,2,4,5]});

// get the elements again with
elements = JSON.parse(json);
```

3.6 REST with Cookies

The problem is that with REST we want a stateless connection, this means that we need to change the way data is transferred, by changing the format.

```
res.format({
  'text/html': function () {
    res.redirect("/");
  },
  'application/json': function () {
    res.send(true);
  },
});
```

The problem is that now, because of the stateless connection, we need to send all data each time.
For example you would need to send something like a *token* to login. -> see discord and why you can create bridges.
Token requirements:

- creation date

- expiry date
- signature
- Other data that might help protect user and server

- Connection can be sent to any server!

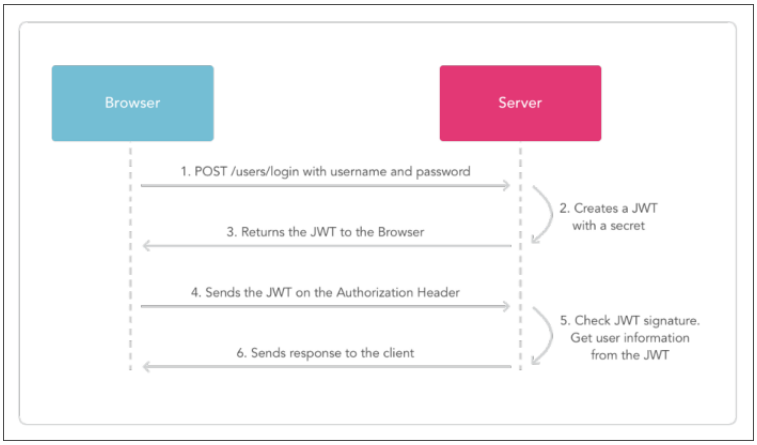
- What happens if the token gets stolen?
- What happens when a device gets stolen?
  How would we invalidate the token on it?

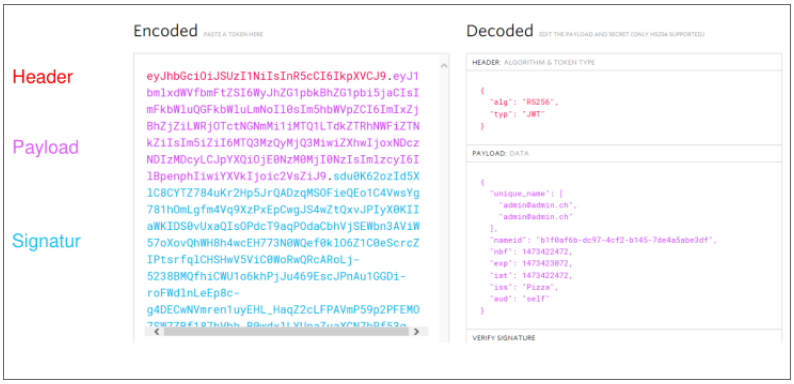
Tokens can be used either as a pseudo key, which can be used to connect to services, or as a temporary login -> remember that you are logged in for a while. RIGHT MOODLE??

Libraries:

- oauth
- express-jwt
- node-jsonwebtoken

3.6.1 JWT Token





3.7 AJAX

When using ajax on webpages, make sure to change the state of the application so that a redo or undo button doesn't end redirecting you back to google. As an alternative use frameworks like react which do this automatically.

```
history.pushState({}, "Demo", "bar.html");
```

3.8 Fetch

```
fetch(url, {
  method: method,
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
}).then(x => {
  return x.json();
});
```

Possible other data:

- arrayBuffer()
- blob()
- formData()
- text()

3.8.1 Cookies with Fetch

Cookies aren't sent automatically, this means you have to send them alongside!

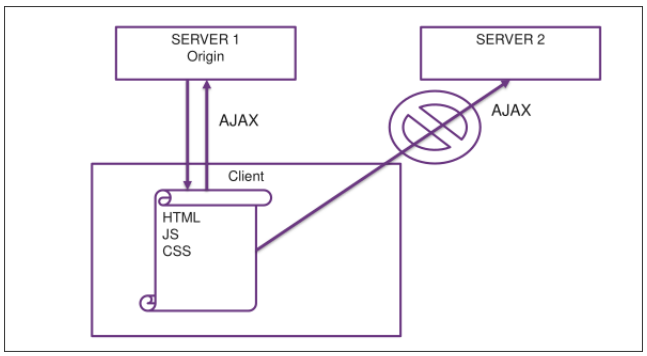
```
fetch('https://example.com', {
  credentials: 'include'
})
```

3.8.2 Fetch helper classes for requests

```
const myHeaders = new Headers();
myHeaders.append('Content-Type', 'text/plain');
const myInit = {
  method: 'GET',
  headers: myHeaders,
  cache: 'default'
};
const myRequest = new Request('/example', myInit);
fetch(myRequest) /*   */
```

3.9 CORS

AJAX security feature. It basically says that a client may only request ajax promises from the one main server.



You can define yourself, who exactly is allowed use what type of functions.

```
const allowCrossDomain = function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
  res.header('Access-Control-Allow-Headers', 'Content-Type');
  next();
};
app.use(allowCrossDomain);
```

4 Typescript

4.1 Motivation

- types...
- actual help in IDEs -> does the function or variable actually exist?
- Warns when types do not match, as said actual types!
- object Object

4.2 Configuration

Each typescript project needs a config file that defines how tsc will compile the files back to js.

```
// tsconfig.json
{
  "compilerOptions": {
    "target": "ES6",
    "module": "ES6",
    "moduleResolution": "Node",
    "allowJs": true,
    "checkJs": true,
    "strict": true,
    "esModuleInterop": true
  },
  "exclude": [
    "node_modules"
  ]
}
```

4.2.1 Import of JS module types

One problem that we have is that the js modules have their own classes etc, these are obviously not included in ts and therefore need a special module import. For example if you use express, then you need to install express types as well: *npm i –save-dev @types/express*

4.2.2 Prettier

Theoretically it is necessary to use these plugins, but nvim fucking rocks so no need.

```
{
  "parser": "@TypeScript-eslint/parser",
  "plugins": [
    "@TypeScript-eslint"
  ],
  "extends": [
    "plugin:@TypeScript-eslint/recommended"
  ]
}
```

4.3 Types

4.3.1 Primitives

- boolean
- number: yes number... -> i32,i64,u32,u64..
- string
- null: WHY
- undefined
- void: special type
- never: special type
- any: like js but only in non strict mode!

4.3.2 Strict Mode

"strict"=true in the config file makes sure that the any keyword can't be used for variables!
Strict also has the following results:

- NoImplicitThis: this needs to be used explicitly
- strictBindCallApply
- strictFunctionTypes
- strictPropertyInitialization: properties of classes need to be initialized -> like rust.
- NoImplicitAny: you need to explicitly write any
- alwaysStrict: set in the first line of a file
- strictNullChecks: null and undefined are no longer base types -> can't be assigned!
- item 4

4.3.3 Collection Types

```
let myInferredNumArray = [1, 2, 3];
let myNumArray: number[] = [1, 2, 3];
// let myNumArray: Array<number> = [1, 2, 3]; // same as above but explicitly
let myTupel = [1, "abcd"]
// problem with this -> myTupel[1] = 2; would work since it can't see type, use the one below to avoid this
let myTupel: [number, string] = [1, "abcd"]   // same as above but explicitly
```

4.3.4 Union Types

Similat to C, unions are a space that can be interpreted as multiple types.

```
let something = string | number; // can be either a number or a string.
something = "pinpang"; // ok
something = 50; // ok
```

4.3.5 Type Narrowing

Due to the fact that a variable might have multiple types, you need to make sure that the type has the correct type for a certain operation at the time:

```
// Type Narrowing - due to assignment
let myVar3: string | number;

// not allowed
console.log(myVar3 / 3)
// allowed
myVar3 = 9;
console.log(myVar3 / 3)

// Type Narrowing - due to condition interpretation
function typeNarrowingFn (myParam: string | number) {

  // not allowed
  console.log(myParam / 3)
  // allowed
  if (typeof myParam === "number") {
    console.log(myParam / 3)
  }
}
```

4.3.6 Function declarations and definitions

```
function add(s1: string, s2: string): string; // function with 2 parameters and return type of string
function add(n1: number, n2: number): number; // function with 2 parameters and return type of nubmer
function add(n1, n2) {
  return n1 + n2;
} // inferred return type

// this function has an optional parameter -> this is annotated with the ? symbol
function combineFunction(sn: number | string = "", ns?: number): string {
  return String(sn) + String(ns || ""); // the double pipe represents the OR as it needs a fallback
  // aka if ns is undefined use the other one
}
// allowed
let myNum: number = add(1, 2);
combineFunction(1);
```

Note, when using the optional parameter, you always need a fallback for this parameter should it not be passed.
Alternatively, and probably *preferably*, use optional parameters with *a default fallback value!*

```
  // here the function takes 2 parameters with the second one being 10 if it is not provided!
  function something(a: number, b = 10 ): number {
    return a + b;
  }
```

Obviously, you can also use functions as parameters:

```
function numberApplicator(numArray: number[],
                          numFun: (prevRes: number, current: number) => number): number {
  return numArray.reduce(numFun);
}
```

4.3.7 Classes

private variables can be defined either with the private keyword, or with a #.
You can of course declare and instantly assign variables in typescript, apparently that was not a thing in js for a while...

```
class Counter {
  #doors: number; // ES2022 private field
  private wood: string = "oak"; // TS private fields
  something?: number; // optional field
  public static readonly WOOD_FACTORS = {'oak': 80, 'pine': 20};

  // readonly -> const
  // # -> private
  // ? -> optional
  // private -> private

  constructor({doors = 2}: {doors?: number} = {}) {
    this.#doors = doors;
  }
  set doors(newDoorCount: number) { // ES6 setter
    if (newDoorCount >= Counter.MIN_DOOR_COUNT && newDoorCount <= Counter.MAX_DOOR_COUNT) {
    this.#doors = newDoorCount;
    } else {
      throw 'Counter can only have';
    }
  }
  get doors() {
    return this._doors;
  }
```

```
}

// You can also define new properties right in the constructor.
// !!initialization is made automatically!!
class SportsCar {
  constructor(
    public make: string,
    public color: SportsCarColor,
    public gears: number,
  ) {}
}
```

4.3.8 Interfaces

```
interface IPoint {
  readonly y: number;
}

interface IPoint {
  readonly x: number;
} // adds value to previous interface

interface ILikableItem {
  likes?: number;
} // optional entry

class DescribableItem {
  constructor(
    public description: string
  ){}
}
class PointOfInterest extends DescribableItem implements IPoint, ILikableItem {
  constructor(public x: number, public y: number, description: string, public likes?:number) {
    super(description);
  }
}
```