**Error Handling In JavaScript | Try - Catch In JavaScript**

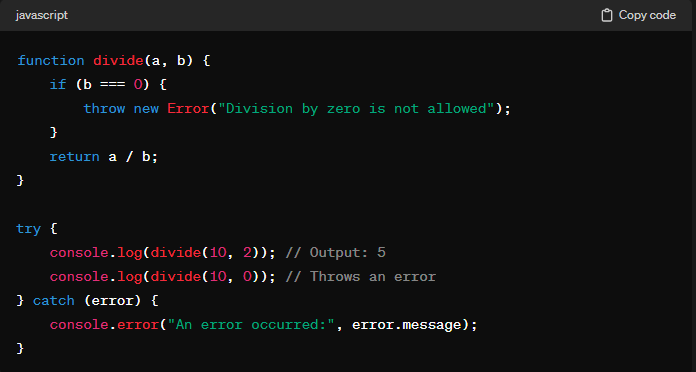
تأتي قيمة throw و tray and catch أن يكون فيه عندك أكثر من جهة بتستدعي الدالة ولما يصير خطأ في هذه الدالة أنت ما محتاج تعالج الخطأ من الدالة نفسها وانما بتسوي throw رمي للمسؤلية ويجي اللي بيستدعي الدالة هو اللي يتكفل بمسؤلية معالجة الخطأ وعرضه للمستخدم عن طريق try and catch

هذا الشئ بيساعدك انك حتى لو أستدعيت الدالة من أكثر من مكان بتقدر تتعامل مع صيغة وح الحطأ بالطريقة اللي تحبها وقتها

النقطة التانية انك تخيل انك ما بتشتغل لوحدك في مشروع وانما معاك مجموعة من المبرمجين وانت بتشتغل على المنطق البرمجي فقط فأنت سويت هذه الجزئية بالمنطق البرمجي بدك تخلي الدالة تبعك عامةيتم أستخدامها لأغراض عامة بدون ما تحط قيود على المبرمجين اللي معك عشان تساعدهم يستخدموها وهم يتعاملوا مع الأشياء اللي خارج عن مسؤليتها

لذلك try and catch بتساعدك انك تبني منطق برمجي لكن الأشياء اللي ما لك علاقة فيها ما راح تطلع alert مثلا انت بتقله انت على حسب ال style اللي فث الصفحة تبع عالج الخطأ بالطريقة اللي انت شايفها وتمشي معاك

الفايدة التالتة أنك تخيل انك بتسوي مكتبة للمبرمجين



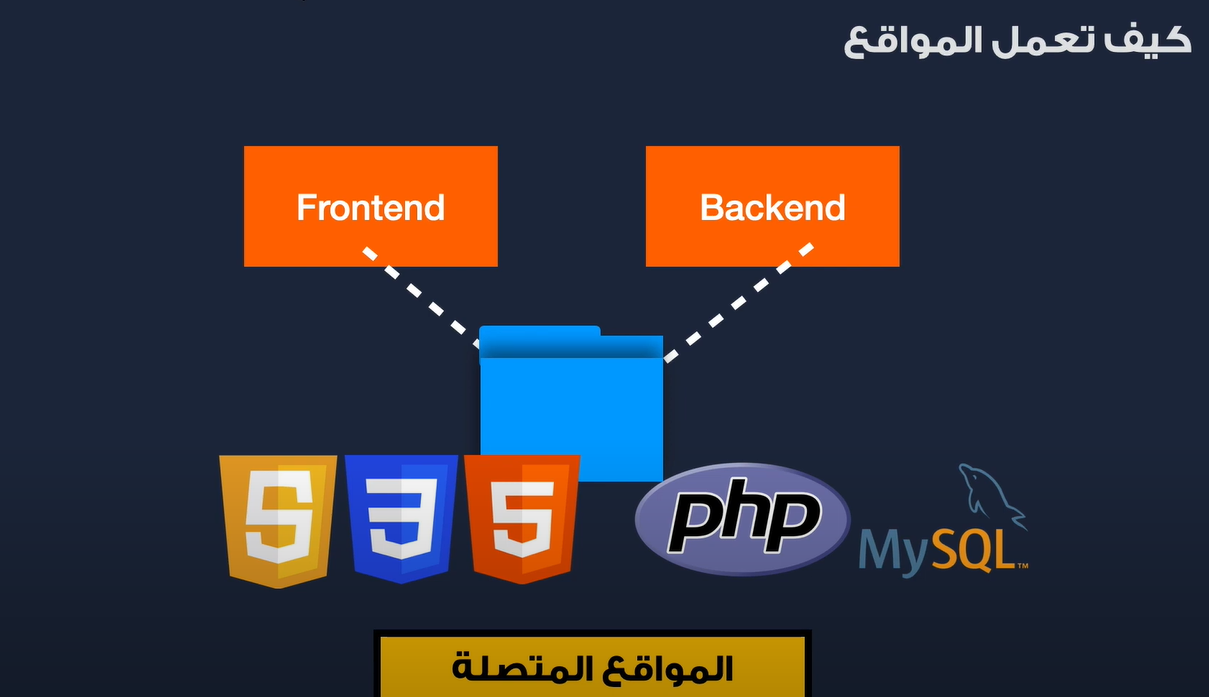
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**أنواع المشاريع**

**مشاريع مواقع متصلة**

وفيه تكون الفرونت ايند مرتبط ارتباط وثيق بالباك ايند عن طريق قيام السيرفر بجلب الداتا من قواعد البيانات

وقيامه بتوليد صفحة html كاملة بهذه البيانات ومن ثم ارسالها لل فرونت ايند او browser





**مشاريع مواقع منفصله**

وفيه يقتصر دور server بجلب الداتا من قواعد البيانات وارسالها للفرونت ايند بواسطة

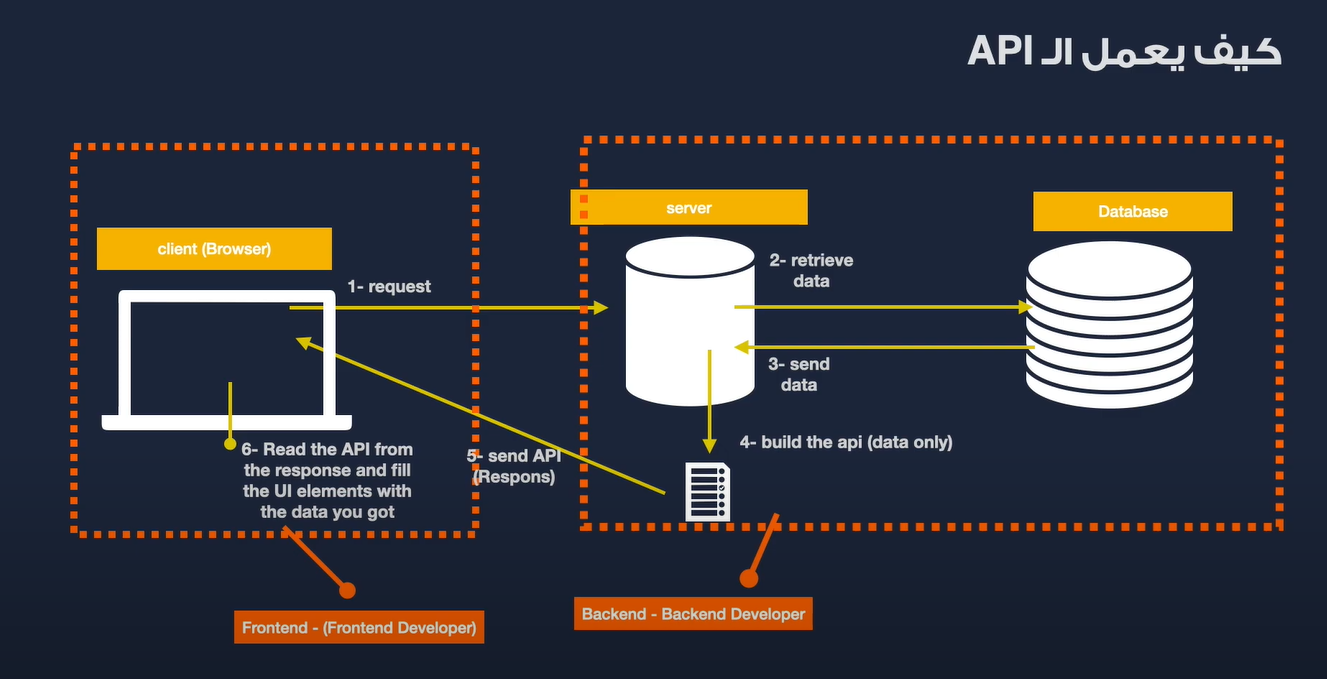
ما يسمى ب API حيث هذه عمليه يقوم فيها السيرفر بتحويل البيانات التي لديه الى لغة موحدة تفهمها كل التقنيات ويفهما أي فرونت ايند حابب انه يتواصل مع هذا API

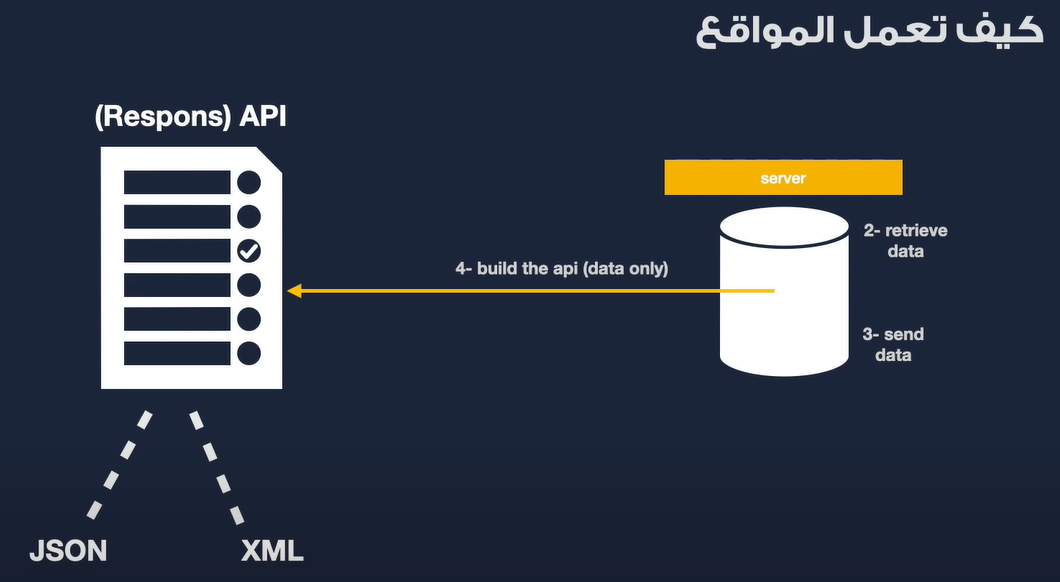
وبذلك نجد ان المشروع منفصل او مقسم ل لشطرين كل شطر له تقنياته وملفاته الخاصة

وخط الربط بين الشطرين أو القسمين هو API الذي يحتوي على صيغة بيانات تفهمها معظم التقنيات مفهومة للباك ايند وهو اللي بينشئها

وللفرونت ايند ايضا وبذلك نستطيع الربط بين القسمين







\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**API == Application programing interface**

واجهة التطبيق البرمجية وهي واجهة أو مفتاح للوصول للباك ايند للتعامل معها

ويكون ميكانيزم التواصل هو أن الفرونت ايند بيرسل طلب للسيرفر بواسطة API الذي يستلم هذا

الطلب ويفهمه ويحوله للغة JSON حيث يستطيع السيرفر أن يفهم محتوى الطلب

يستلم السيرفر الطلب ومن ثم يذهب لقواعد البيانات ويقوم بعملية جلب للداتا وينظمها ويرتبها في شكل معين ويسلمها ل API حيث تترجم هذه البيانات لصيغة JSON ليستطيع الفرونت ايند أو browser فهمها والتعامل معها حيث يستلم هذا ال response والبيانات المتواجدة فيها ويعبيها في ملف html جاهز وموجود في المتصفح من الأساس وما بيحتاج تعبئته أو تجميعه من خلال السيرفر

--انتاج API هو عبارة عن عملية من عدة خطوات منها

استدعاء البيانات

ترتيبها بشكل معين

وبعدين عرضها بصيغة معينة

ويوجد صيغتين لعرض API === الصيغة الأولى هي صيغة JSON

والصيغة التانية هي صيغة XML ودي اللي كانت منتشرة في التعامل مع API

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**أهمية برنامج Postman**

Postman من عائلة البرامج اللي بنقدر نسميها API client

بمعنى ان هذا البرنامج بيتم استخدامه عشان تجرب api وتفهمه وتستوعبه وتفهم طريقة التواصل معه قبل ما انك تسوي هذه requests من الجافا سكربت

وهنا سؤال ليه انا متطر أستخدم برنامج بس عشان أجرب وبعدين أروح أكتب الأكواد ف الجافا سكربت واجربها

لأن عملية التواصل مع API لازم من خلالها تستوعب ايش هي المسارات اللي بتتيحلك ياها API وايش هي انواع requests وايش هي وايه هي parameters اللي مفروض ارسلها مع request والأشياء المختلفة اللي لازم تكون مستوعبها

عملية فهم هذا API وفهمه وتجربته واستيعابه ما هي عملية سهلة من انك تسويها من خلال الجافا سكربت بشكل مباشر

اذا كان فيه عندك API CLIENT بينظملك العملية وبيوضحلك ياها بشكل أفضل فهذا الشئ راح يسهل عليك مشاكل جدا كبيرة

ولا تفكر ان خطوة postman مثل خطوة اضافية بالعكس معظم الشركات في سوق العمل

بيستخدموها سواء postman او برنامج اخر

فيه نقطة مهمة عاوزك كفرونت ايند تعتبر الباكايند ك صندوق اسود انت ما متطر تعرف ايه اللي جوة الصندوق ده بس الصندوق ده بيكون ليه api معين تقدر من خلاله انك تتواصل معه

لذلك راح نروح عند api معين موجود ومجهز ومخصص للممارسة اذا حابب يكون عندك api بدك تتعلم منه اللي هو اسمه

JSON Placeholder وهو احد المنصات اللي بتوفرلك api مجاني للتدريب

* اتفقنا ان backend عبارة عن صندوق اسود بقدر اوصله وادخل لمحتواه من خلال API وأن هذه APIS تحتوي على مجموعة من الروابط هذه الروابط تسمى END POINT كل واحدة منها تحتوي على معلومات ومهام
* انا عن طريقها برسل لها REQUEST وبتعطيني respose في هذه الداتا وتوجد مع هذه الداتا رقم 200 وهو يدل على ان عملية request و response تمت بنجاح بدون حصول اي مشاكل وعادة الارقام من 200-299 هي تعبر عن نجاح العملية واللي أعلى من 400 تعبر عن وجود مشكلة في العملية
* مكونات ال request
* Postman ما هو الا برنامج للتجربة
* لما نشتغل بشكل عام دائما بنجهز request في postman ونتاكد انه شغال وان انا فاهمه كمبرمج وبعدين بحاول انقله الى الجافاسكريبت

--- فيه خطوة كتيير مهمة وهي أن أننا لازم نشيك على status code قبل ما نتعامل مع response اللي جاي من api مباشرة عشان لو فيه خطأ نحاول نتعامل معاه قبل ما يوقف عمل كل JS كله ويوقف البرنامج ودي خطوة لابد منها وضرورية جدا جدا جدا

* لابد status code = 200-300 لا اقل ولا أكثر
*  وهذا هو الشرط اللي بشيك بيه على سلامة response

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**مكونات request**

**A screenshot of a computer

Description automatically generated**

Postman راح يضل معنا على طول الخط وكل end point جديد بدنا نتعامل معه راح نجربه اول شئ في postman ونتأكد ان عملية التواصل تمت بنجاح وان فاهم بالضبط ايش متطلبات ال end point من خلال ال postman وبعدين راح ابدأ اخدها ع الجافاسكربت

**A screenshot of a computer

Description automatically generated**

**types of requests**

In the context of web development and APIs, there are several types of HTTP requests that clients can make to interact with servers. These requests correspond to different actions that the client wants the server to perform. The most common types of HTTP requests are:

1. **GET**: This request method is used to retrieve data from the server. When a client sends a GET request, it asks the server to send back a specific resource, such as a webpage, a file, or an API endpoint's response. GET requests should only retrieve data and should not have any other effect on the server's state.
2. **POST**: POST requests are used to submit data to the server to create or update a resource. Clients typically use POST requests when submitting form data, uploading files, or making requests that change the server's state in some way. The data sent with a POST request is included in the request body.
3. **PUT**: PUT requests are similar to POST requests in that they are used to submit data to the server. However, PUT requests are idempotent, meaning that making the same PUT request multiple times should have the same effect as making it once. PUT requests are often used to update existing resources on the server.
4. **DELETE**: As the name suggests, DELETE requests are used to delete a resource from the server. When a client sends a DELETE request to a specific URL, it asks the server to remove the resource at that URL.
5. **PATCH**: PATCH requests are used to make partial updates to a resource on the server. Unlike PUT requests, which typically replace the entire resource, PATCH requests only modify the specified fields of the resource.

These are the most commonly used HTTP request methods, but there are also less common methods like OPTIONS, HEAD, and TRACE, each serving specific purposes within the HTTP protocol.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**API with javascript**

* **عمل GET request بالطريقة القديمة طريقة new XMLHttpRequest()**

**GET <<<==**



**POST <<<==**



**PUT <<<==**

هنا هشوف path موجود ضمن url عشان يحددلي المكان اللي أنا هعدل فيه بالضبط زي العنوان كدة هتروح ل end point الفلانية هتلاقي جواها path لعنوان التعديل المطلوب



الرقم 1 هذا يمثل path

-------

الفرق بين patch و put

وهو ان في patch بحدد جزء معين في ال resource اللي انا بدي أحدثه

انما في put بنحدث كل ال resource

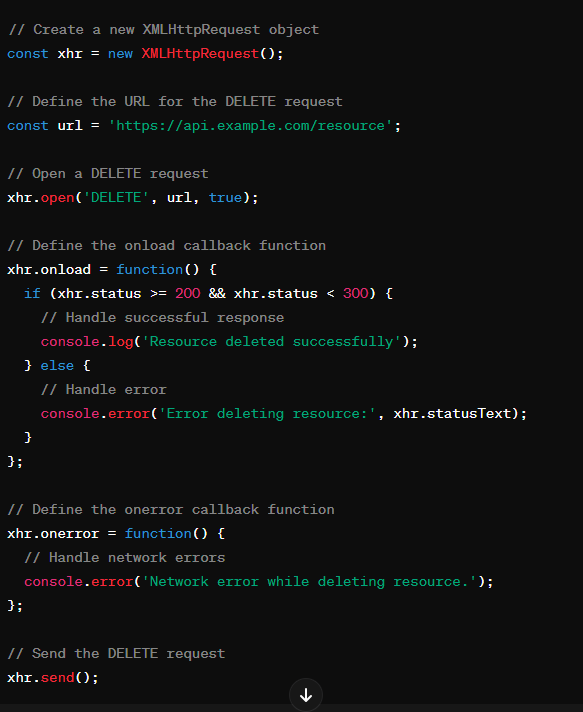
****

**PATCH <<<==**

****

**DELETE <<<==**

هنا الوضع أسهل لأننا مش محتاجين نكتب حاجة ف body parameter بس بيطفيه ال path ضمن url

****

**Filtering resources**

بنقدر نعمل filtrate ل response القادم من api بواسطة ما يسمى ب query parameters

وهو عبارة عن شئ بيتحط ضمن الرابط URL ويكون بهذه مثل هذه الصيغة



حيث نكتب علامة "؟" ثم key and value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**OOP=> Object oriented programming**

**البرمجة كائنية التوجه**

في java script في عندنا شغلتين بيتم تسميتهم على انهم object

الشئ الأول هو أي شئ بيتم بناءه بأستخدام ال JSON

الشئ التاني هو object اللي بيتم بناءه بواسطة class

و class اصلا هو شئ بيندرج تحت مفهوم OOP اللي هي البرمجة كائنية التوجه

البرمجة كائنية التوجه هي بتعامل أجزاء الكود عندك على انها كائنات واذا بدنا نعرف الكائن

فاحنا راح نقول الكائن هو : عبارة عن شئ له سلوكيات وخصائص وممكن ننقل هذا التعريف بحيث يتوافق مع البرمجة

كيف بنسويه بدنا نحاول نلاقي شئ يمثل الخصائص وشئ يمثل السلوكيات ونجمعهم مع بعض ضمن اطار او قالب واحد ومن هذا القالب بنقدر نطلع مجموعة من المنتجات

السلوكيات ====< بيتم ترجمتها على انها function

الخصائص ===< بيتم ترجمته على انها متغيرات

القالب اللي على هذه السلوكيات وهذه الخصائص فعليا هو عبارة عن class

وبيصير هذا ال class عبارة عن مصنع انت بتقدر تستخدمه كقالب لتنتج مجموعة من الأشياء

ف class في نهاية الأمر ترى هو عبارة عن قالب لا يعني شئ لوحده ف لوحده ما له قيمة

أول شئ انت بتسويه في OOP هو أنك بتبني القالب بعدين بتبني منتجات من هذا القالب

اللي يمثل هذا القالب في OOP هو class

فمثلا أنا بدي اسوي قالب وهذا القالب ينشئلي كائنات هي عبارة وليكن عن حيوانات

وهسمي هذا القالب animals

اذا انا بستطيع من خلال هذا المصنع أو ال class اللي اسمه animals انتاج مجموعة من المنتجات أو الأشياء أو objects

في هذا المصنع توجد السلوكيات وهي عامة

وتوجد الخصائص وهذه الخصائص هي اللي بتميز كل حيوان عن غيره

لذلك بتوضع بداخل دالة لها اسم constructor هذه الدالة يوجد بها مجموعة parameters بيتم التعويض عنهم عند صناعة ال object بقيم خاصة (arguments) كما أن هذه الدالة بيتم استدعائها تلقائيا مع كل مرة بننشئ فيها object ومحتويات هذه الدالة من خصائص بتنتقل الى ال object الجديد اللي تم انشائه بال بال parameters الجديد اللي اتمررت معاه و كلمة this اللي موجودة جوة الدالة بتعبر عن object اللي بيتن انشائه لذلك هذه الخصائص اللي داخل ال constructor تنتقل ل object الجديد

لذلك هنلاقي ال objects مختلفين في بعض الخصائص اعتمادا على ما مرر له وقت صناعته وبيتفقوا في functions او المسؤليات

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Promisis and callback hell**

* سبب أستدعاء promisis في JS

هو ان عندي مجموعة من الأكواد الغير المتزامنة اللي هي اسمها Asynchronous code وأنا عاوز أدخلهم مسار زمني واحد يعني أخلي طريقة عملهم متزامنة أو انها تمشي في مسار زمني واحد مثل اكواد Synchronous code وذلك لأي سبب من الأسباب أو للقيام ب وظيفة معينة اتمام هذه الوظيفة معتمد على عمل هذه الأكواد بطريقة التزامن الزمني والمسار الواحد ولتنفيذ هذه العملية وعمل هذه الترابط

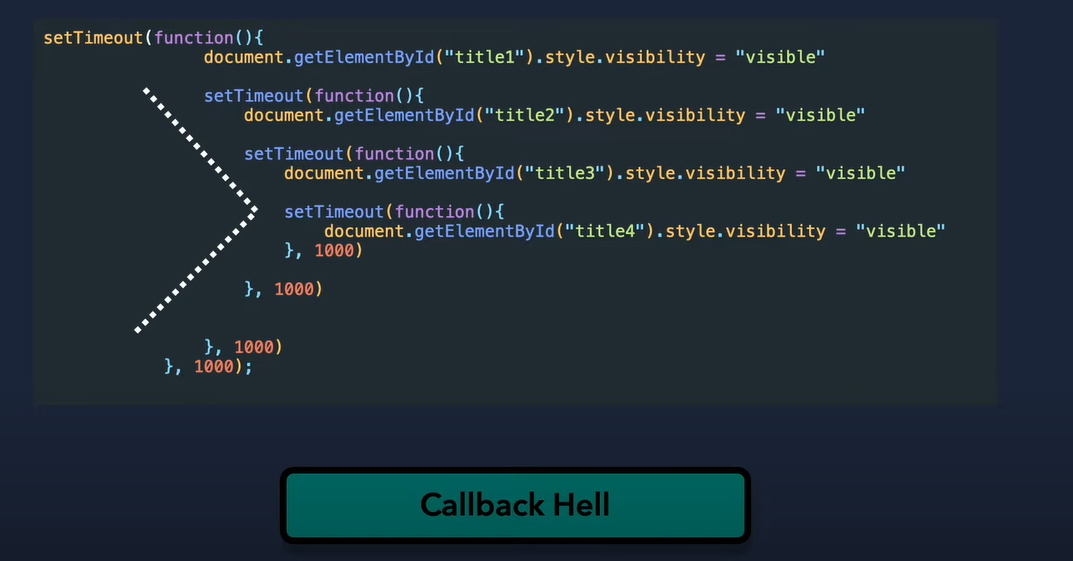
محتاجين نحط هذه الأكواد في نفس المكان اللي بنشغل فيه async code بحيث يكون ضمن الزمن الفرعي لهذا async code وهكذا لو فيه كود تاني محتاجة يشتغل بعد هذا الكود الأخير محتاج احطه في نطاقة بحيث ياخد الزمن الفرعي ليه

ولهذا

نلجأ الى ما يسمي ال callback function كتلك التي نستخدما ف sittimeout وغيرها

ولكن وجدنا ان العمل بهذه الطريقة صعب جدا لصعوبة قراءة الأكواد والتعديل عليها وذلك بسبب ما يسمي

ب **callback hell**

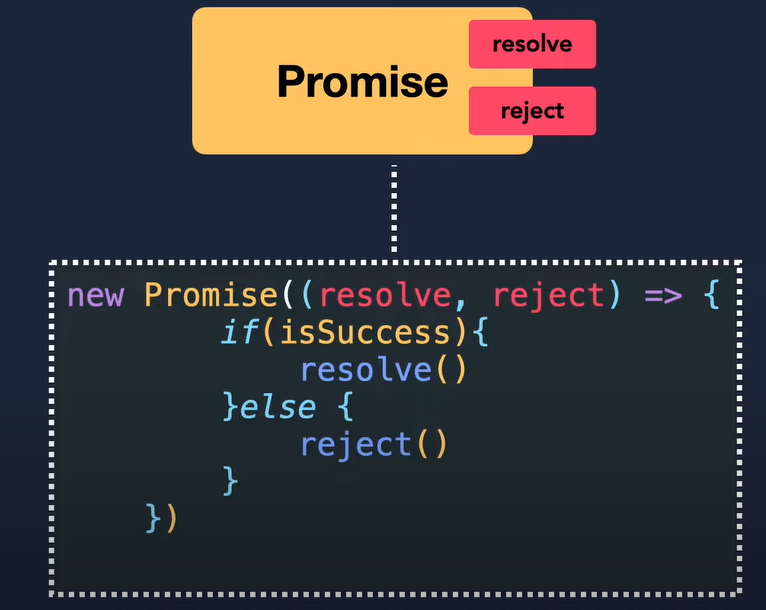


لذلك لجأنا لحل أسهل وأفضل يتيح لنا قراءة الأكواد بسهولة وكذلك التعديل عليها لو لزم ذلك

لذلك لجأنا لتقنية جديدة تسمي ب promises

**Promises**

عبارة عن وعد أنت بتديه لشئ معين عشان ترجع لهذا الشئ وبربط تنفيذا هذا الوعد من خلال تنفيذ async code كنا مررناه لداخل ال promise والشئ اللي من خلاله بربط بيه تنفيذ الكود بالوعد هي دالتين واحدة تسمى resolve و الأخرى تسمى reject بيتم أستدعاء الأولى مع نجاح الكود والثانية مع فشل الكود ويكون الأستدعاء بداخل async code اللي شغال غليه وغلط يكون هذا الأستدعاء خارجه لأنه لو كان خارجه هيتم استدعائها قبل اكتمال هذا async code



Promise عبارة عن class بنشئ منه object وبقدر اسمي هذا object بأي أسم عادي

هذا class بيقبل مني parameter هذا parameter يكون عبارة عن دالة هذه الدالة بدورها تقبل مني اتنين parameter

الاول هو ===< هو resolve وهو عبارة عن دالة

الثاني هو ===< هو reject وهو عبارة عن دالة أيضا

وهذه الدالة أيضا بكتب بداخلها Async code هذا الكود يكون بمثابة وعد أو promise للدالتين then and catch

اذا تم تحقيق هذا Async code أو هذا الوعد عندها يتم أستدعاء دال <==== Resolve()

اذا لم يتم تحقيق هذا Async code أو هذا الوعد عندها يتم أستدعاء دالة <==== Reject()

وهذه الدالة then و catch عبارة عن دالتين يتوقف عملهما على نجاح او فشل promise

وهما يعملان بشكل تلقائي بنائا على نتيجة promise

وكلا منهما يقبلان مني parameter هذا parameter عبارة عن دالة هذه الدالة تقبل مني parameter هذا parameter يمثل القيمة الموجودة بداخل دالتين الأستدعاء resolve and reject

حيث القيمة بداخل resolve تذهب الى =====< then

حيث القيمة بداخل reject تذهب الى =====< catch

فالموضوع بأختصار promise مكون من جزئين

الجزء الأول وهو بناء promise واللي بيتضمن تنفيذ async code

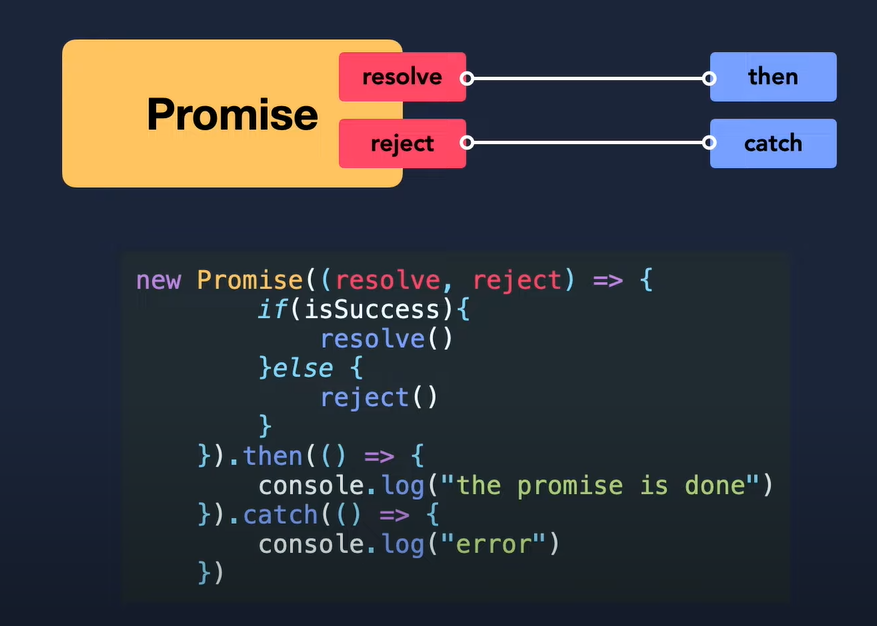
ونتيجة ل response اللي راح يجيني من تنفيذ Async code

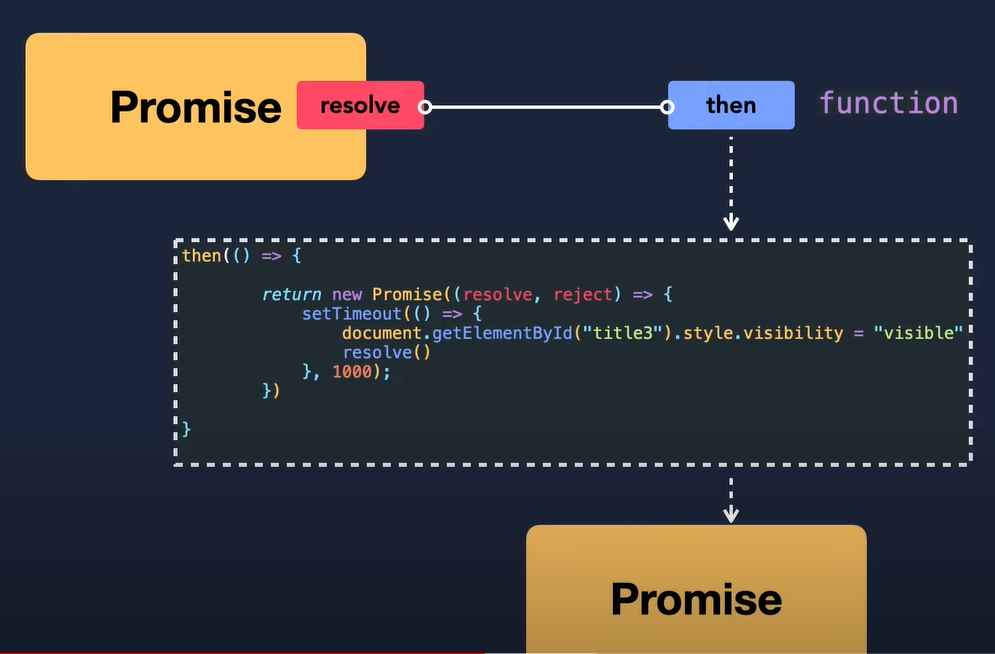
اذا كان ناجح بروح أستدعي الدالة resolve() اللي راح تقوم بدورها وتشغل الكود اللي موجود في

Then بشكل تلقائي

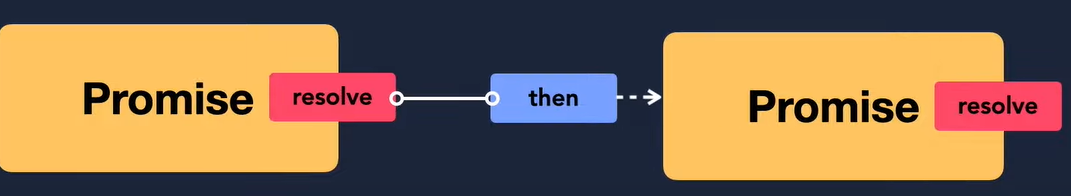
وذا كان فيه مشكلة فهتستدعي reject واللي راح تستدعي الدالة التانية اللي موجودة في then

أو لو مش موجودة هتروح تستدعي دالة catch بشكل تلقائي



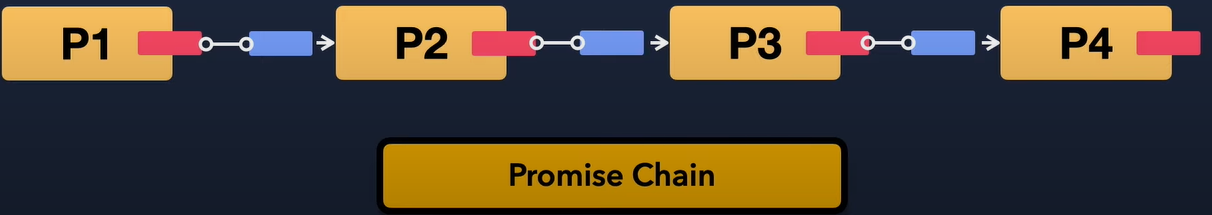


يمكن ل لدالة ال then ان ترجع هي الأخرى promise آخر أقدر من خلاله أربطها ب then أخرى ويمكن تكرار العملية تالت ورابع وخامس الى ملا لانهاية

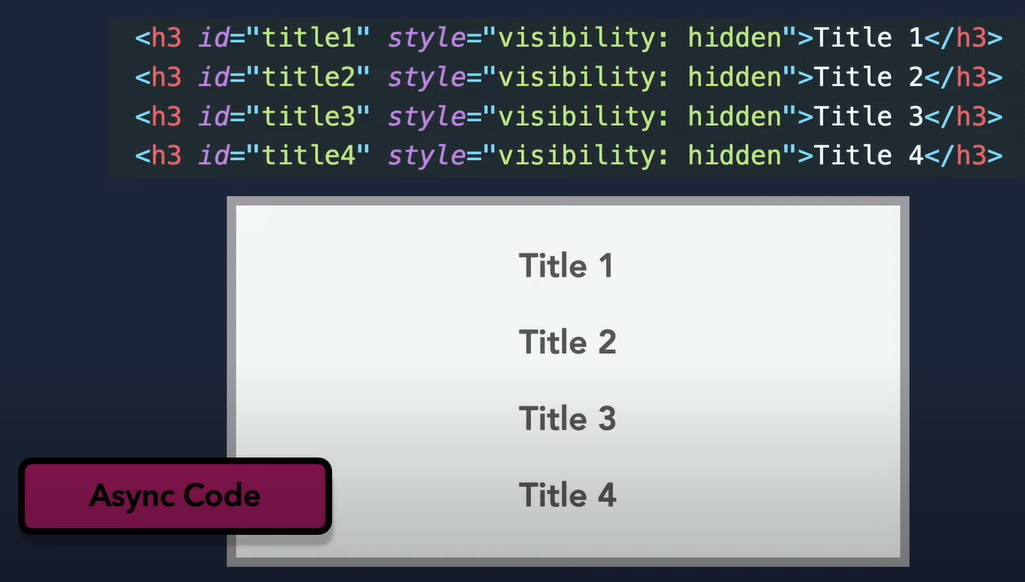
وهو ما يعرف ب promise chain



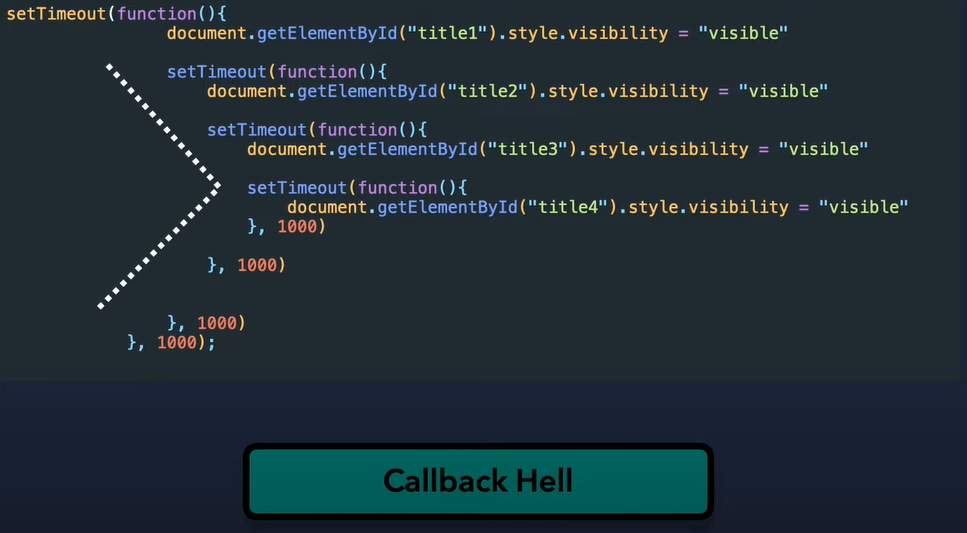


وهذا مثال على على ذلك

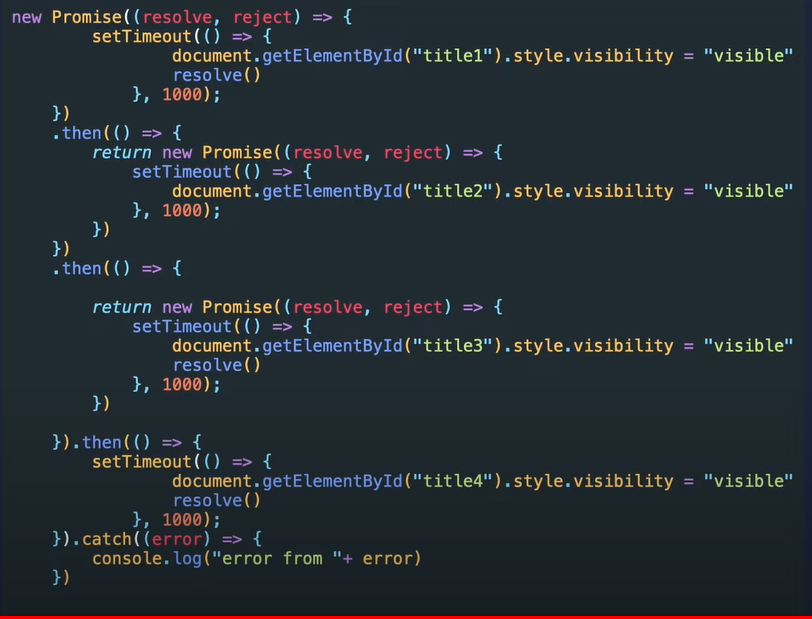
المطلوب اظهار هذه العناصر في الصفحة بشكل متوالي لا يتم الثاني الا بعد تنفيذ الأول بشكل متزامن



تم تنفيذه بالطريقة القديمة هكذا callback



وبطريقة promises هكذا



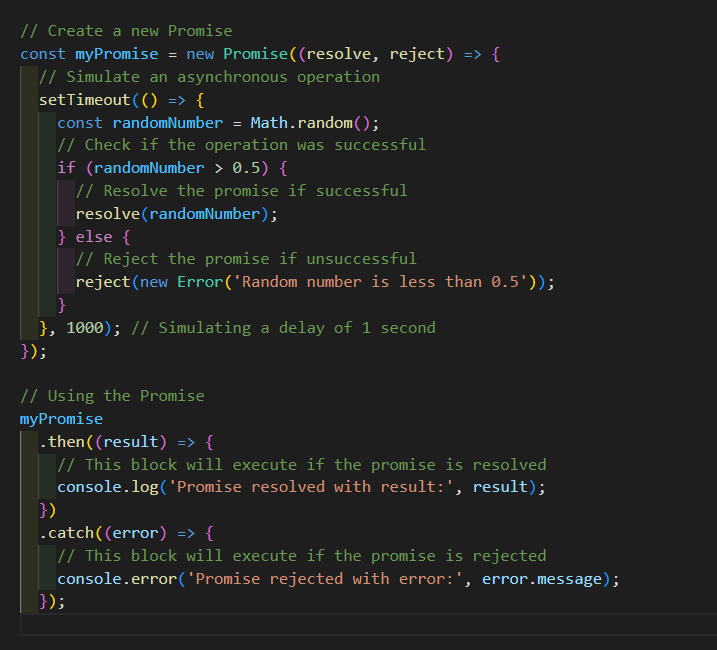
قواعد مهمة

* أي شئ يرجع لك promise سيجعلك قادرا على أستخدام then اللي بعده
* يتم استدعاء الكود اللي موجود في then عندما يتم استدعاء resolve في promise
* يتم أستدعاء الكود الموجود في catch بمجرد أستدعاء reject

\*\* القيمة أوالكود الموجود بداخل دالة resolve(------) or reject(-------) تكون قيمته هو قيمة parameter

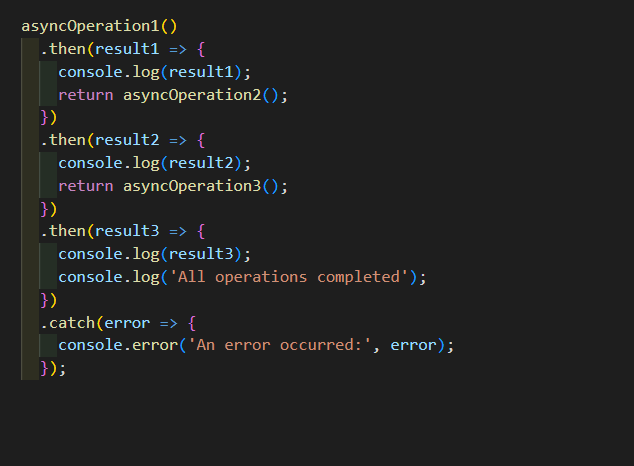
الذي يكتب بدخل function الموجودة ك pramater بداخل دالة then or catch

Example==🡺



**Example to convert callback hell code to promise code**





**لماذا promises مهمة في كورس API**

لأن API call كلها عبارة عن Async code كود غير متزامن بستنى فيه كود معين يخلص عشان أنفذ كود آخر

فهنا بيجي دور promise كركيزة أساسية تسهلي شغلي وتعطيني مساحة أكبر أعرف أشتغل فيها بدون التعرض لما يسمى callback hell

**Fetch**

وظيفة fetch تتلخص في أنها تقوم بوظيفتين

## الوظيفة الأولى وهي عمل GET request ل API وجلب response منه

## الوظيفة الثانية وهي عمل promise أو أعطاء promise يمكن من خلاله أستخدام دالة then

ويكون لها اتنين then

في then الأولى نقوم بجلب محتوى response في شكل array of objects or json format وعمل return لهذه الداتا

ل then الثانية وفيها يمكننا استخدام الداتا فيما نريد

1. **Asynchronous Data Fetching**: **fetch()** allows you to asynchronously fetch resources from the network. This is crucial in web development as it allows you to fetch data from servers without blocking the main thread, ensuring that your application remains responsive.
2. **Promise-based Interface**: **fetch()** returns a Promise, which simplifies asynchronous programming in JavaScript. Promises provide a clean and readable way to handle asynchronous operations and manage their results or errors.
3. **Modern Replacement for XMLHttpRequest (XHR)**: While XMLHttpRequest has been the traditional method for making HTTP requests in JavaScript, **fetch()** offers a more modern and flexible alternative. It provides a simpler and more intuitive interface for making network requests.

يجب ملاحظة أنه يجب رمي error خارج then ليتم التقاطها بواسطة catch والا فان الكود كله لن يعمل ولن نرى رسالة الخطأ المدونة في catch

وذلك بعكس **axios** حيث أن الخطأ يتم رميه تلقائيا

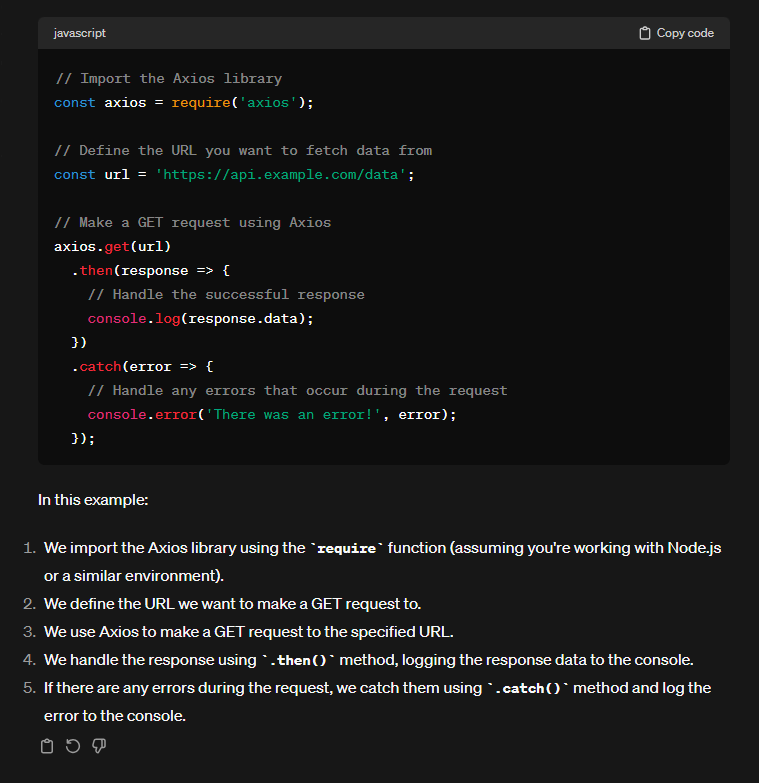
****

**axios in javascript**

عبارة عن مكتبة بتم استدعائها لملف html ويمكننا من خلالها عمل

GET request وأعطاء promise الذي من خلاله نستطيع أستخدام دالة then وكذلك عمل handle to error بشكل تلقائي

بشكل أفضل وأيسر



الفرق بين fetch and axios

هو ان axios أكثر سهوله في استخدامه ففيه نستخدم then مرة واحدة بعكس fetch

الأمر الثاني وهو فحص response ورمي الحطأ نجد في axios ان مثل هذه الأمور تتم بشكل تلقائي

ويوجد طريقتين لأستخدام هذه المكتبة

--- الأولى أما بطريقة CDN وفيها لابد أن نكون متصلين بالأنترنت لكي تعمل

--- الثانية وهي عن طريق أستيراد كامل المكتبة عن طريق ما يعرف package manager (npm) node وهنا لا تستدعي اتصال الأنترنت

للتأكد من ان node JS تم تحميله بطريقة صحيحة بنروح ع terminal وبنكتب node -v و npm -v وبيعطيني رقم الأصدار وده بيدل على انه اتحمل بطريقة صحيحة

Node JS هو وسيط لتنزيل مكتبات خارجية خاصة ب JS

طريقة أستخدام npm داخل المشروع هو

---- أول حاجة أروح لمكان المشروع بتاعي من خلال terminal وأقله خلي npm يديرلي المكتبات على هذا المشروع ,اعمل initialize ليه بواسطة كتابة هذا الأمر npm init

بعدها بيتولد عندي ف المشروع ملف بأسم package.json وبيكون ويفته أنه بيسجل كل المكتبات اللي أنا مشغلها عندي المشروع اسم كل مكتبة وتاريخ الأصدار بتاعها بشكل مختصر بحيث لو أرسلت هذا المشروع لشخص آخر فمن خلال هذا الملف بيعرف ايه المكتبات اللي شغالة عالمشروع ده واللي لازم تنزلها

---- بعد كدة بنعمل تنزيل لمكتبة axios بواسطة هذا الامر npm install axios - -save كلمة save هي لحفظ هذه المكتبة في ملف package.json وهذه الكلمة ممكن نكتبها وممكن لا بس الأفضل أن نكتبها

وبمجرد كتابة هذا الأمر بنجد أن فيه ملفين تم أضافتهم للمشروع

الأول ==+ هو node-modules وهذا الملف هو اللي بيحتوي على كل ملفات الأكواد لكل المكتبات اللي بنستخدمها

الثاني ==+ هو ملف package-lock.json ووظيفة هذا الملف هو كتابة رقم الأصدار الخاص بكل مكتبة عشان لو المشروع اشتغل عليه حد تاني ميحصلش تضارب لو نزل اصدار غير الأصدار

والغرض الأساسي من وجود ملفات package هي أن يتم مشاركتها ومشاركة المعلومات اللي فيه مع باقي التيم اللي شغال معايا في نفس المشروع عشان عملية رفع وتنزيل المشروع بملف node-modules هي بطيئة والبديل انهنا نشارك معلومات المكتبات اللي موجودة باخل هذا الملف عن طريقة مشاركة ملفات package الخفيفة فقط

وكل اللي علينا ساعتها لو حبينا ننزل نفس المكتبة اللي في المشروع الرئيسي اننا نكتب كو التنزيل وليكن هكذا في حالة npm

Npm install

--- المكتبة الآن أصبحت موجودة عندي ضمن مشروع العمل وعشان أشغلها لازم أعمل استيراد للأكواد بتاعتها

بواسطة script tag ودي في حالة لو مش شغال بأي js framework

=====+ في حالة لو بنستخدم احدى اطر عمل JS بنستخدم ذلك الكود const axios = require('axios'); وبنستغنى عن خطوة الأستيراد السابقة

الخلاصة اننا بنجيب اكواد المكتبات لداخل المشروع بتاعنا بدل الأتصال بالأنترنت عن طريق npm اللي بتنزلي المكتبات على ملف node-module واللي بيساعدني اني اعرف المكتبات اللي شغال عليها واصداراتها هي ملفات package

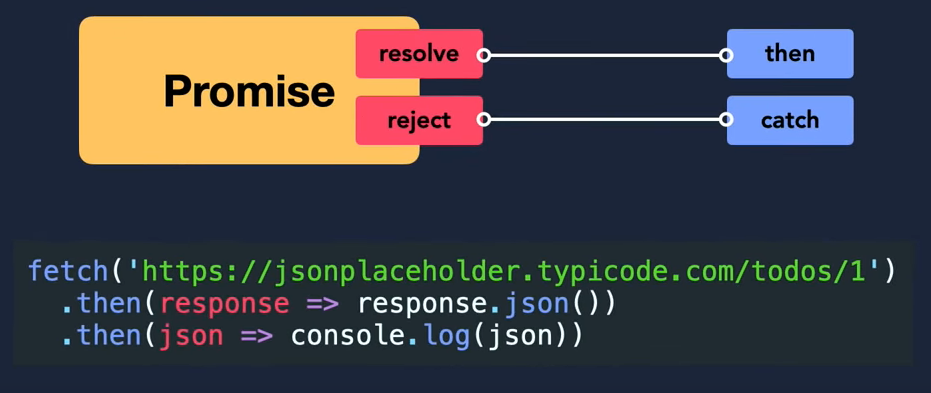
عشان يوفر علي عملية نقل هذا المجلد اللي عادة بيبقى ضخم وكبير

**async / await in javascript**

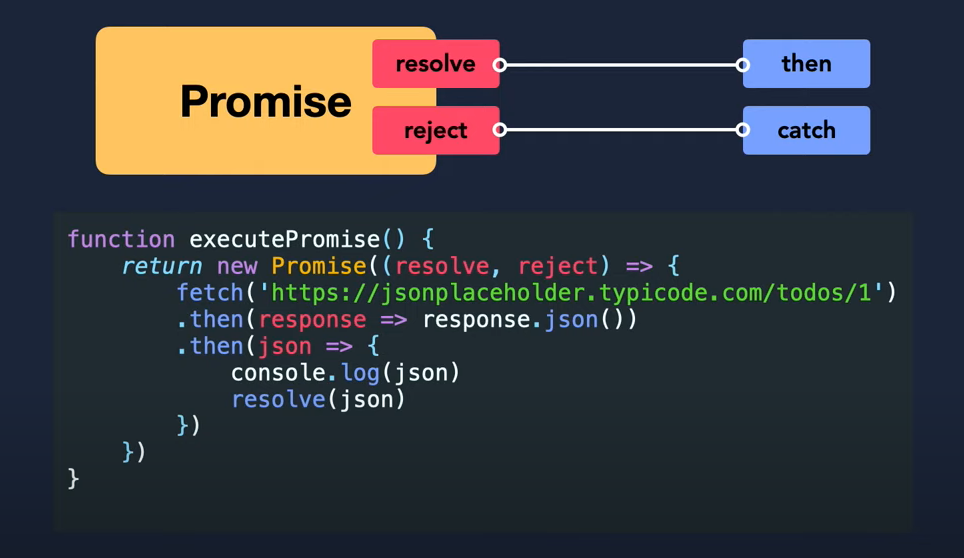
ال promise كان عبارة عن شئ بيستخدم عشان ينفذللك ال async code وبمجرد ما يكتمل هذا async code بيستدعيلك ال دالة resolve اللي من خلالها ممكن تربطها ب then واذا فشل بيستدعي reject اللي بيربطها ب catch

وكمثال عليها دالة fetch اللي بتعمل request و promise في نفس الوقت وبناء عليها هنقدر نستخدم then

من بعدها



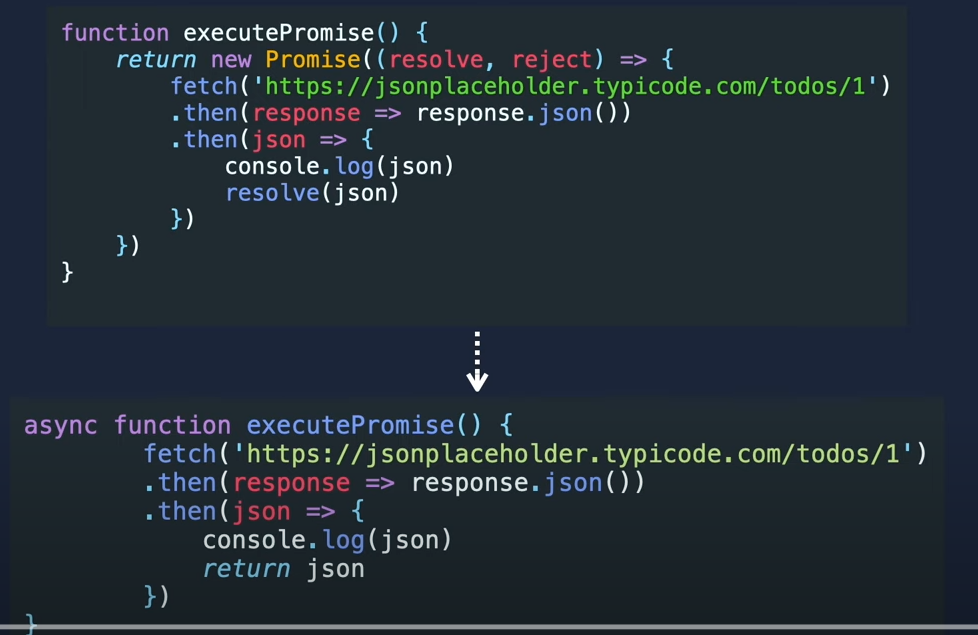
وهذا api call اذا حابب تنفذه وبمجرد ما يكتمل حابب تنفذ شئ تاني مثل api call تاني فممكن تحطها داخل دالة في حد ذاته بحيث هذه الدالة ترجعلك promise وداخل هذا promise تستدعي الدالة fetch اللي بتروح تجبلك البيانات ولما بتتمل البيانات بتستدعي ال resolve



ومع تطور لغة JS وجدوا ان هذا البناء نوعا ما فيه شوية تعقيد وممكن يتم تحسينه بطريقة أفضل وخصوصا لما تستخدم promises chain اللي هي دالة بترجعلك promise بعدها بتستخدم then وبداخلها ترجع promise بعدها بتستخدم then الى آخره

وكانت طريقة التحسين من خلال استخدام async and await

حيث يمكنها تحويل هذا الشكل اللي في الأعلى الى الشكل التالي



حيث تكتب قبل كلمة function كلمة async

وبمجرد كتابة هذه الكلمة فهذا كافي انه يحولك الشئ اللي عم بترجعه الدالة اللي هو ب هذه الحالة json كانه يصير ك parameter موجود في resolve الموجودة في promise بترجعه هذه الدالة

فالكود الموجود في الأعلى هو مساوي تماما للكود الموجود في الأسفل لكن احنا اختصرناه بأن حطينا كلمة async فصارت الدالة لحالها كل شئ بترجعه ك promise وهذه جزئية الأختصار الأولى بأستخدام async

بعد ما رتبنا الدالة بهذا الشكل احنا لسة ممكن نرتها بشكل أكبروهذه السلسة من then ممكن نحولها للشكل التالي

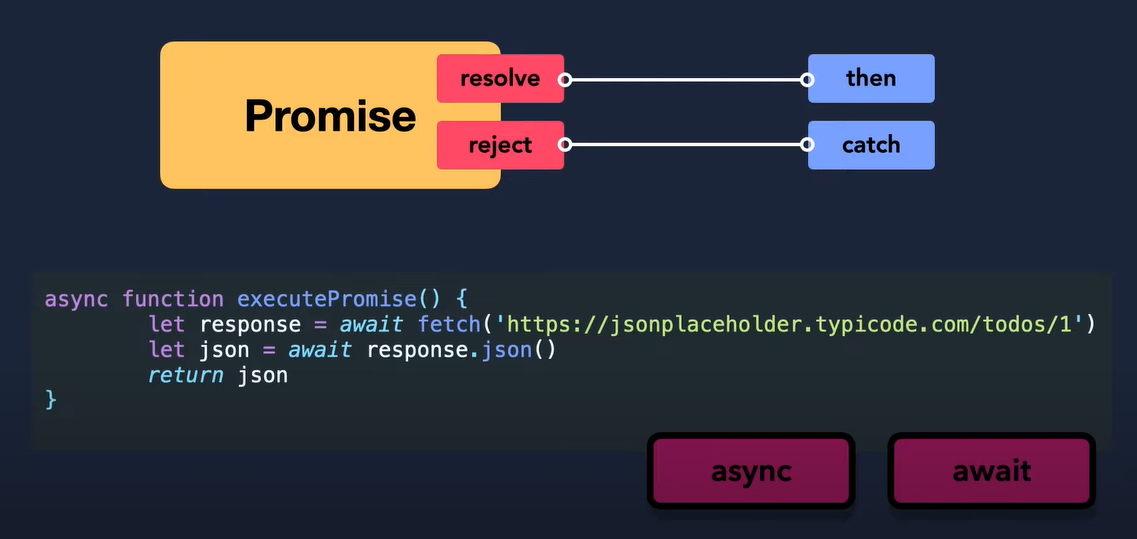
بدل ما تصير تقول then هتعرف متغير يساوي أستدعاء الدالة اللي بترجعلك promise لكن قبلها تقله أو تكتب الكلمة await بمعني ان لا تنتقل للسطر التالي حتي تنتهي من إكمال هذه السطر أو يكتمل هذا API call ويصير هذا المتغير اللي أسمه response عنده قيمة وقتها بتقدر تنتقل للسطر اللذي يليه اللي عم بيعرفلك json وبالتالي كأن ال await أستبدلت then ورتبت الكود بشكل أكبر

علما بأنه لا يمكن أستخدام ال await الا بداخل async function

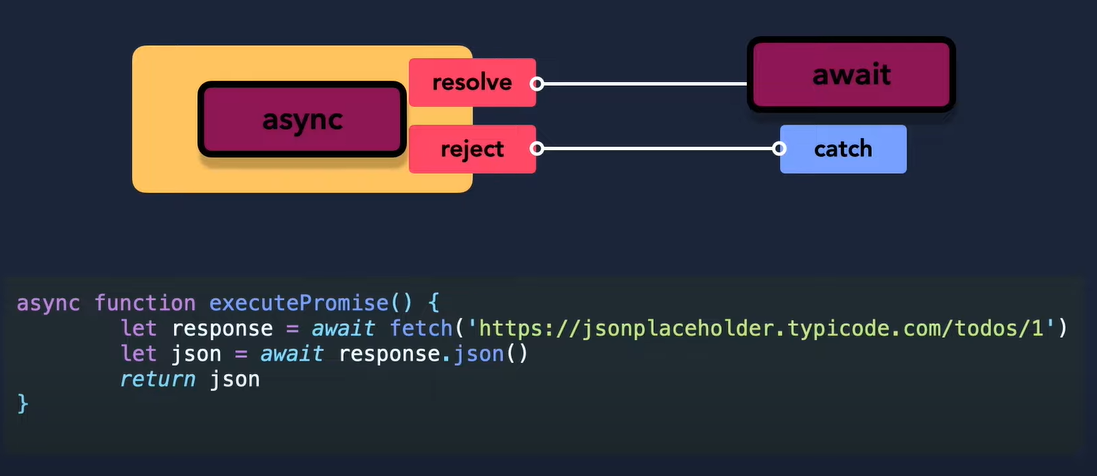
فاذا كان عندك async بتقدر وقتها تستخدم await بديلا عشان يترتب معك الكود بشكل أكبر



وبذلك نتيجة لأستخام ال async and await احنا حولنا promise الى هذا الشكل المرتب من الدالة



وبذلك نقدر نقول ان async بتستبدل ال promise وال await بتستبدل ال then بحيث أنك تستخدم async للدالة اللي بدك منها تطلع promise وال await بتستخدمها للأسطر اللي بدك تنتظر فيها أنه تكتمل العملية قبل ما تنتقل للسطر التاني وكأنك عم تستبدلها بدل ال then بحيث يطلع معاك الكود بهذه الطريقة المرتبة



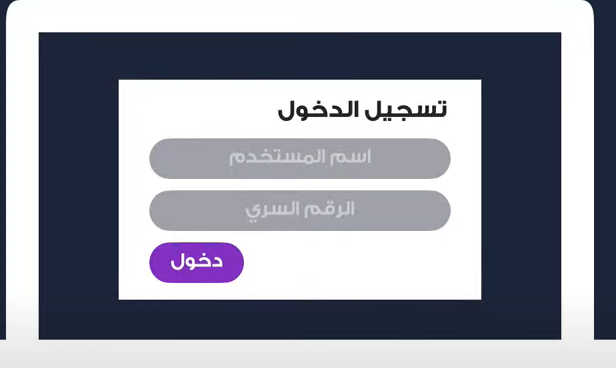
**أهم قاعدتين في async and await**



**كيف يعمل ال auth و ال token في المواقع**

هناخد مثال عند فتح أي تطبيق من تطبيقات التواصل الأجتماعي مثلا وليكن Facebook

غالبا هيطلب منك أسم المستخدم + كلمة المرور عشان يتيح لك التعامل مع البرنامج



وعندما تدخل هتكون حينها قادر على أدارة المنشورات اللي موجودة عندك واللي هيتم نشرها بأسمك

==== < لكن السؤال الآن كيف بيتم تمييز أن هذا الشخص فعلا مخول أنه يكتب هذه المنشورات بأسمك وأنك فعلا هذا الشخص بناء على شاشة تسجيل الدخول وبمعنى آخر ايه اللي بيتم خلف الكواليس عشان السيرفر يتأكد من هويتك وأنه فعلا هو أنت الشخص المخول ليه بأدارة الحساب مش حد آخر

== أول حاجة عندنا شاشة تسجيل الدخول اللي موجودة ف client

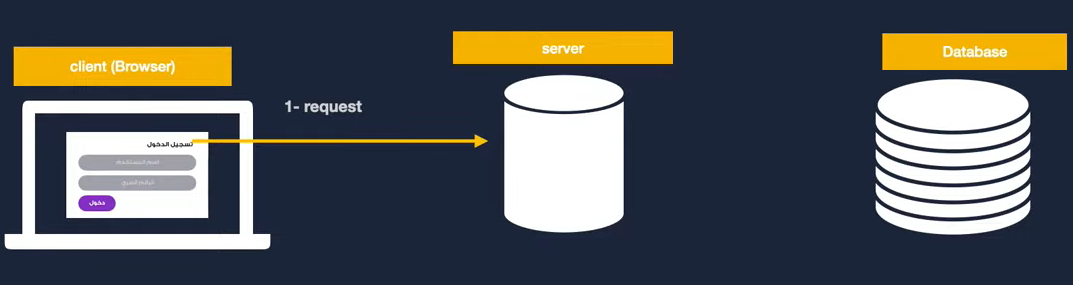
== server واللي بيتلقى ال requests اللي أنت بتسرلها من جهازك

== قاعدة البيانات أو data base

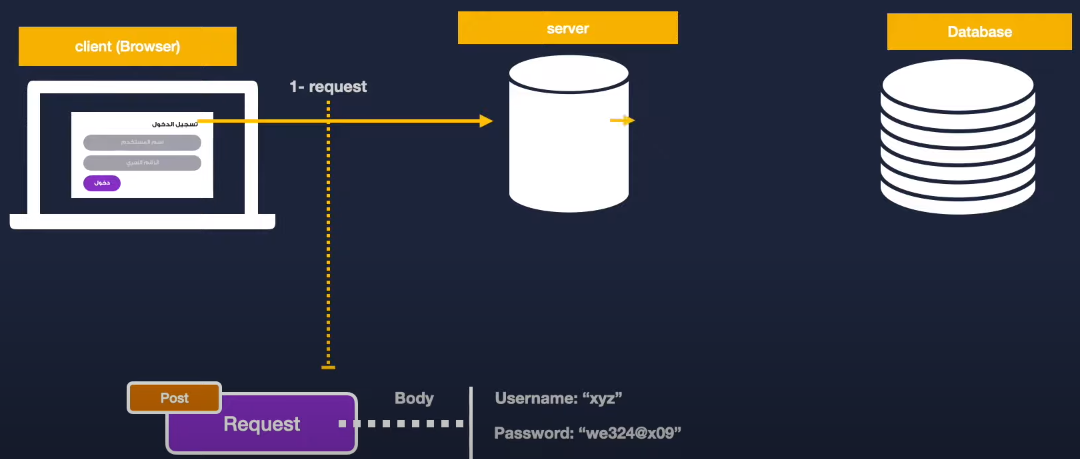


ولما بتكتب اسم المستخدم وكلمة السر في صفحة تسجيل الدخول وبتضغط ارسال اللي بيحصل ان client بيرسل request

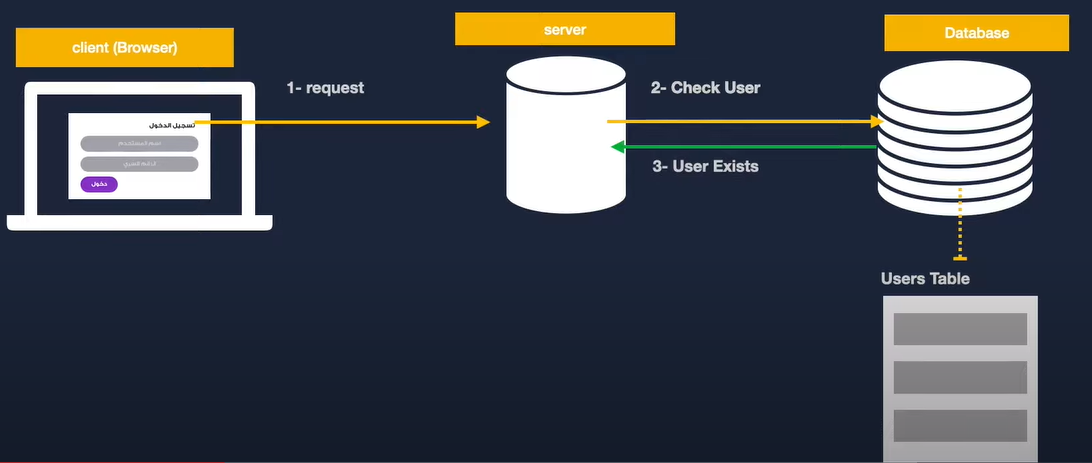
وهذا ال request يتم ارساله الى ال server



وهذا ال request عبارة عن post request وهو عبارة عن طلب محمل بمجموعة من المعلومات اللي بنسميها body parameters وهذه المعلومات هي اسم المستخدم وكلمة المرور

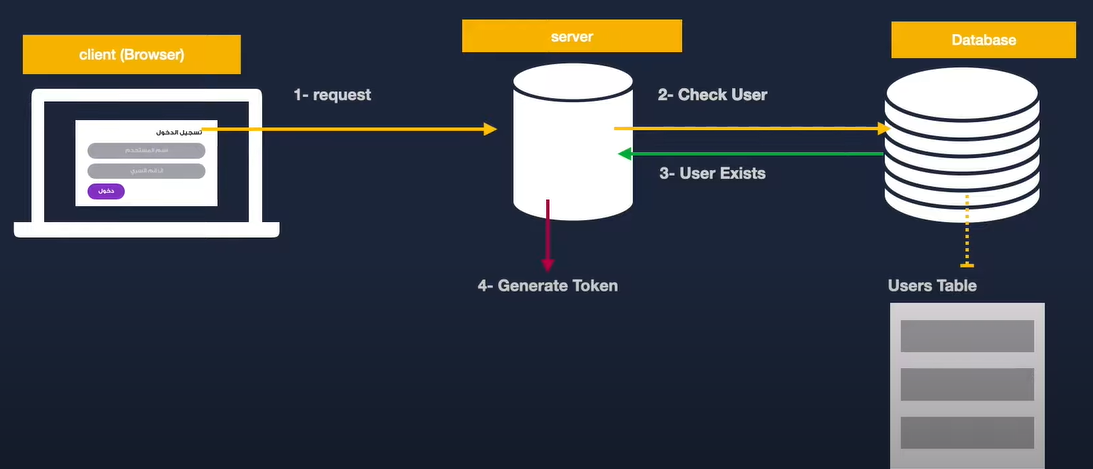


اللي بيسوية السيرفر بعدين هو أنه بيروح وبيسأل قاعدة البيانات وبيطلب منها أنه تشيك على هذا المستخدم هل فيه فعلا مستخدم بياناته تطابق مع البيانات المرسلة حيث أن قاعدة البيانات لديها جدول المسمى ب users table اللي بيحتوي على كل بيانات المستخدمين واللي هيصير ان قاعدة البيانات هتشيك على هذا المستخدم وتشوف فعلا ان فيه مستخدم يتطابق بياناته مع هذه البيانات ام لا وبناءا عليه لما تلاقيه راح ترجع ل server أن هذا المستخدم فعليا موجود



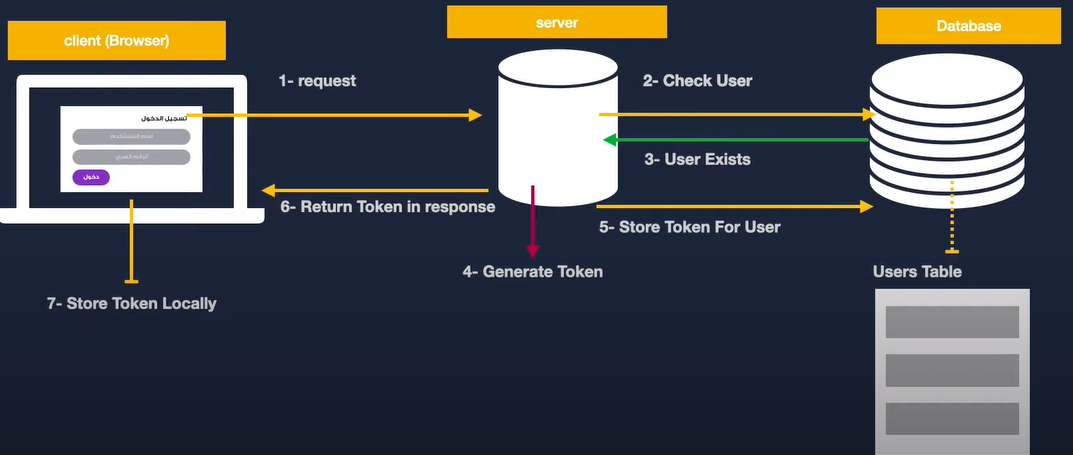
نتيجة لذلك server بيقوم بأنشاء لشئ يسمى ب token

و token عبارة عن رمز مركب بطريقة معينة يستخدم لتمييز هذا المستخدم في أي request لاحق بيسويه مثل عملية أنشاء المنشورات



وبعد أنشاء هذا token بيروح ال server وبيطلب من قاعدة البيانات أن تخزن هذا ال token في أحد الجداول الموجودة عندها وتربطه بهذا المستخدم وكذلك بيروح يرجع response المحمل ب token لل client اللي هو المتصفح

والل بيسويه المتصفح بعدين أنه بيروح يحفظ هذا ال token عنده في الجهاز عشان يستخدمه في أي request تاني بيسويه المستخدم لاحقا



وعند الذهاب لصفحة المنشورات وعند تسوية منشور جديد وعبأنه بياناته وضغتنا على ذر النشر أو الأرسال فاللي بيصير كالتالي هو أنه client بيرسل request ل server وهذا ال request هو عبارة عن post request الذي يحتوي بدوره على body params التي تحتوي بداخلها على المعلومات الخاصة بهذا المنشور أضافة الى أهم معلومة وهي token واللي هو تم أنتاجه من ال server وتم تخزينه في data base والمتصفح أيضا عشان يميز هذا المستخدم عن غيره عند عمل مطابقه له مع قاعدة البيانات

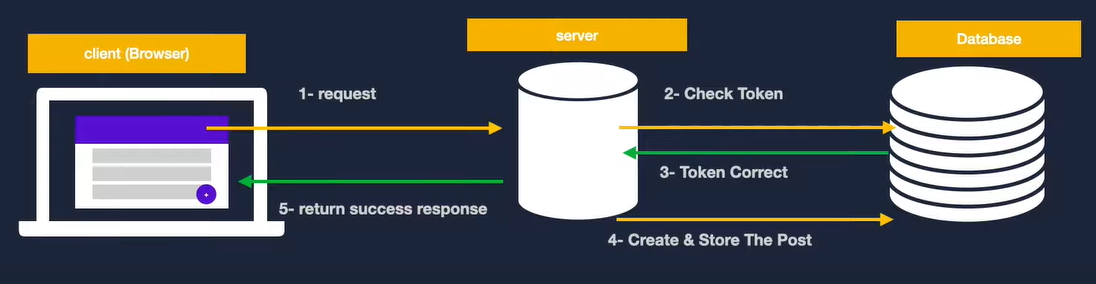


وبالتالي ال server بيستقبل هذا token وبيروح يطلب من قاعدة البيانات انها تفحصه وانه موجود ومرتبط مع مستخدم معين

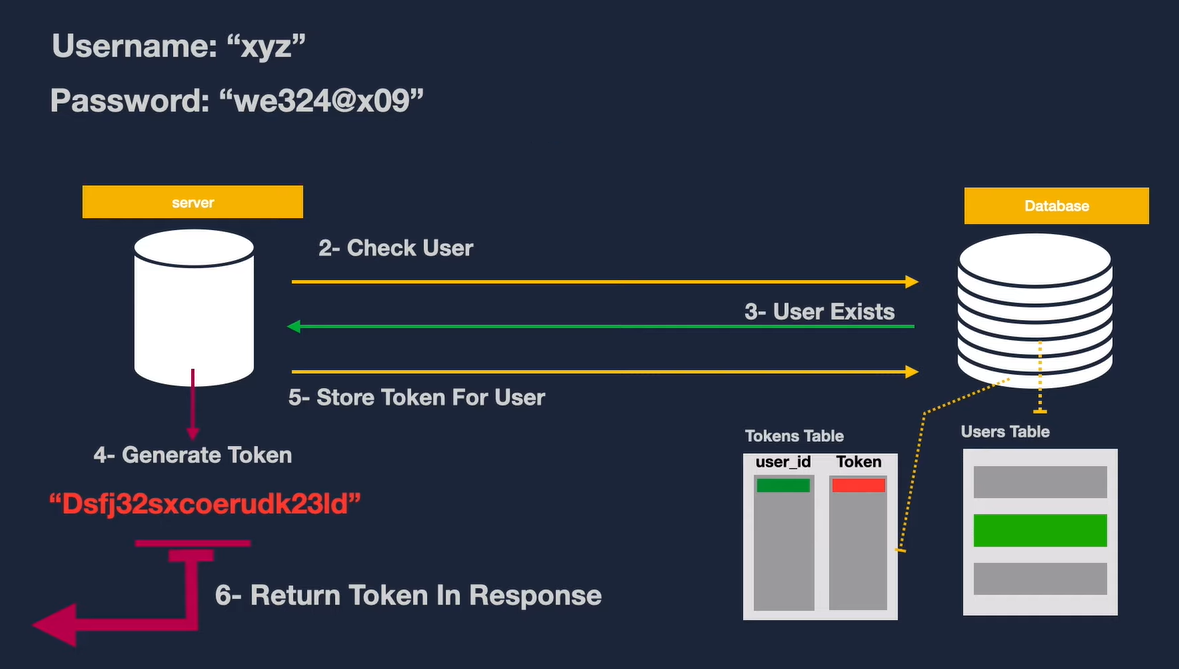
فتقوم قاعدة البيانات في البحث في الجداول الموجودة عندها وعندما تطابقه راح ترسل ل server وتقله ان هذا المستخدم صحيح ومرتبط مع مستخدم معين

وعندها بيقول ال server ان ممكن نسوي هذا المنشور وبيروح بيقول لقاعدة البيانات سوي منشور جديد وانه يتم تخزين بيانات هذا المنشور مثل ما حددها المستخدم ويكون المنشور تحت اسم هذا المستخدم تحديدا

وبعدها راح يرجع server ويرسل response لل client ويقله تم نجاح انشاء هذا المنشور



وده ملخط للميكانيكة اللي بيتم بيها تعامل server مع قاعدة البيانات ملخصة في هذا الشكل



بعد ما المستخدم بيسجل بياناته في صفحة تسجيل الدخول ف client وبيضغط ارسال

ال server بيقوم استقبال هذه البيانات وبيقوم بأدخالها في هذا الميكانيزم وصولا انه يبعت token لل client

واللي بيرسله client بعد كدة ضمن بيانات اي request قادم وده الل بيتم بيه المطابقة عشان اي action يتم عمله بواسطة المستخدم يتم تخزينه تحت البيانات الخاصة بيه في قاعدة البيانات

**ايه اللي ممكن استفيده من token**

Token اللي بيرجعلنا من عملية request عملية الدخول بيصير هو المفتاح لأي request لاحق انت بتحطه كجزء من header الخاص بهذا ال request وهذا header لازم تكون مسميه بتسمية معينة لازم تكون دقيق فيها وما تسوي فيها أي خطأ والا ما راح تستفيد منه

اللي بيحصل في الواقع اني عند أنشاء request جديد سواء لأنشاء post او user جديد أو غيره

أني بنسخ token اللي جالي من عملية login وبروح أحطها كقيمة ل key اسمه Authorization بهذا ال spilling تحديدا بداخل header param

ال token اللي بتحصل عليه هو عدة أنواع تعتمد على API فيه token العادي وفيه شئ اسمه Bearer token وفيه أنواع أخرى

ف ال token ممكن نحطة كقيمة ل auth… بشكل مباشر لو هو token عادي

وفيه بعض ال token اسمها bearer tokens بعد ما أجيب قيمة هذا ال token ف القيمة او value بتاع ال key اللي موجود في header اللي اسمه authorization لازم تبدأ بكلمة Bearer ثم مسافة ثم الرمز الخاص بال token

--- يحب وضع ال token في ال local storage عشان مش كل مرة أعمل فيها refresh للصفحة مش أكون محتاج أعمل login من جديد

وعند عمل أي request جديد هروح أجيب token ده من local storage

وهذا مثال بسيط

