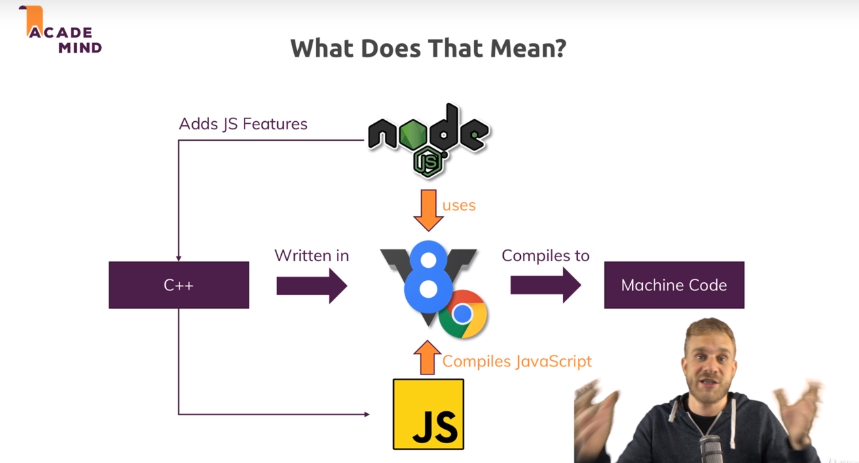
1. What is node?

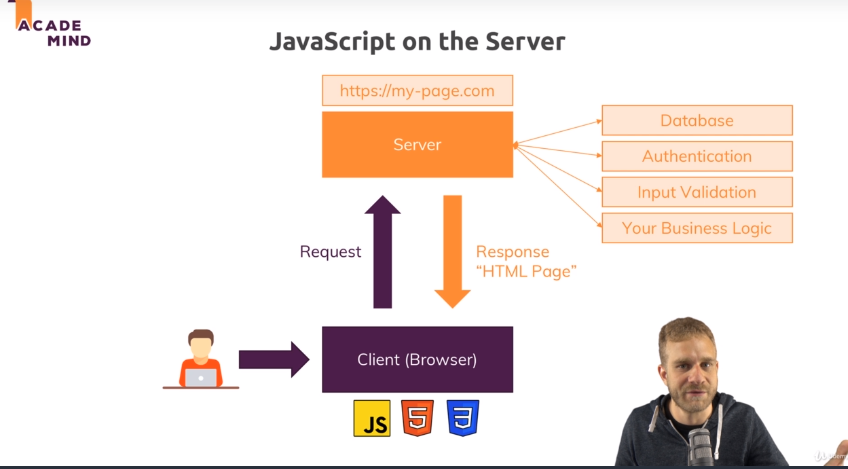
V8 Engine compiles javascript language and compiles to machine code to communicate with machine.

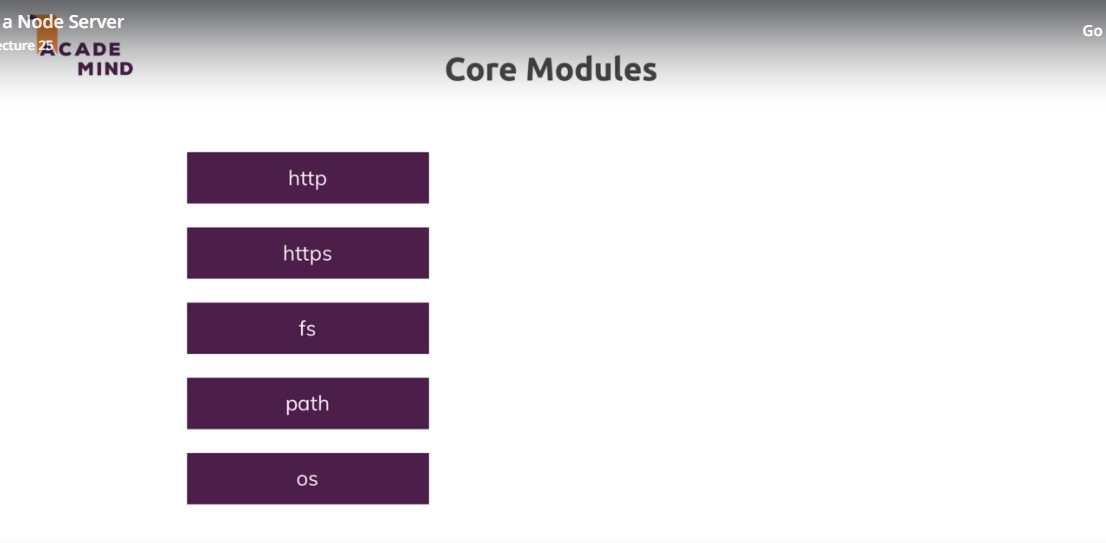
Node is a javascript runtime.



1. Node’s role (node works for server, not for browser)

Node creates server & listen to incoming requests. Also, handle requests from the client and build business logics.

1. 
2. Frequently used node module



1. About basic headers in http (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers)

HTTP headers allow the client and the server to pass additional information with the request or the response. An HTTP header consists of its case-insensitive name followed by a colon ':', then by its value (without line breaks). Leading white space before the value is ignored.

Custom proprietary headers can be added using the 'X-' prefix, but this convention was deprecated in June 2012, because of the inconveniences it caused when non-standard fields became standard in [RFC 6648](https://tools.ietf.org/html/rfc6648); others are listed in an [IANA registry](http://www.iana.org/assignments/message-headers/perm-headers.html), whose original content was defined in [RFC 4229](http://tools.ietf.org/html/rfc4229). IANA also maintains a [registry of proposed new HTTP message headers](http://www.iana.org/assignments/message-headers/prov-headers.html).

Headers can be grouped according to their contexts:

* [General header](https://developer.mozilla.org/en-US/docs/Glossary/General_header): Headers applying to both requests and responses but with no relation to the data eventually transmitted in the body.
* [Request header](https://developer.mozilla.org/en-US/docs/Glossary/Request_header): Headers containing more information about the resource to be fetched or about the client itself.
* [Response header](https://developer.mozilla.org/en-US/docs/Glossary/Response_header): Headers with additional information about the response, like its location or about the server itself (name and version etc.).
* [Entity header](https://developer.mozilla.org/en-US/docs/Glossary/Entity_header): Headers containing more information about the body of the entity, like its content length or its MIME-type.

Headers can also be grouped according to how proxies handle them:

**End-to-end headers**

These headers must be transmitted to the final recipient of the message; that is, the server for a request or the client for a response. Intermediate proxies must retransmit end-to-end headers unmodified and caches must store them.

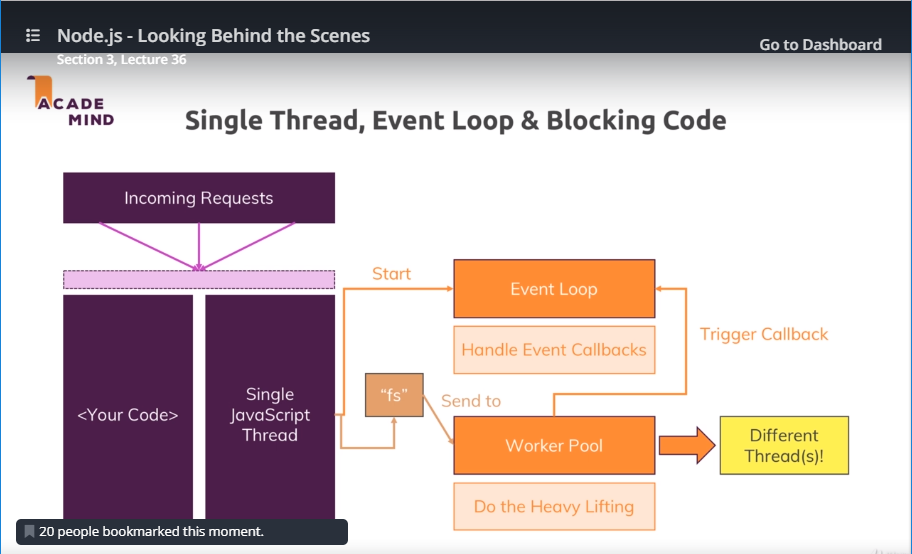
**Hop-by-hop headers**

These headers are meaningful only for a single transport-level connection and must not be retransmitted by proxies or cached. Such headers are: [Connection](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Connection), [Keep-Alive](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Keep-Alive), [Proxy-Authenticate](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Proxy-Authenticate), [Proxy-Authorization](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Proxy-Authorization), [TE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/TE), [Trailer](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Trailer), [Transfer-Encoding](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding) and [Upgrade](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Upgrade). Note that only hop-by-hop headers may be set using the [Connection](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Connection) general header.

The following list summarizes HTTP headers by their usage category. For an alphabetical list, see the navigation on the left side.

1. Node.js
2. Non-blocking event-driven framework.

* Performance is super fast because most of the events are working asynchronously. The event function registers the callback in the registry. Then, The events stack up in the event que. Finally, in the event loop, whenever the clients requests the function, the event is fired through the event loop.

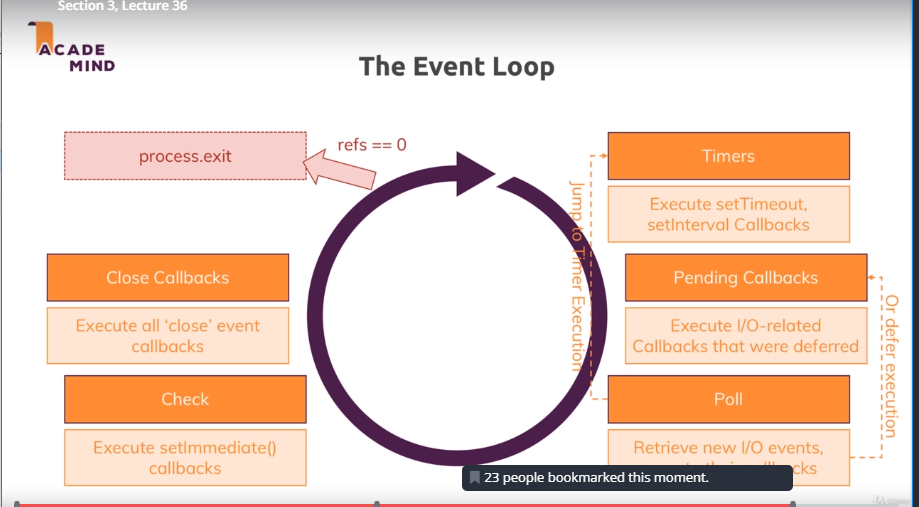
1. Single Thread but it is dealing with multiple thread.
2. 

* As mentioned up and above, node is implementing the event loop. Therefore, even in a single thread, the node is able to run the multiple tasks requested by the client by implementing ‘stack’ of the events in the event que that is fired by the request.

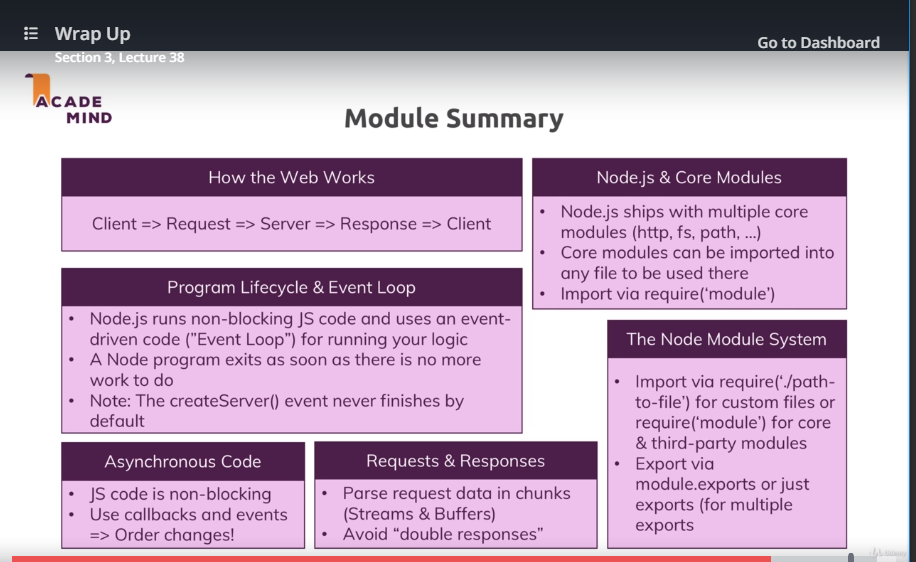
1. Full CPU Peformance

If some parts of CPU are available to the I/O task, node.js is able to use the spare CPU to maximize the best performance.

<How the event loop works>



Wrap up of the basic node.js



NPM

Except for a few commands like “start”

We must put “run” to execute the npm command.

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

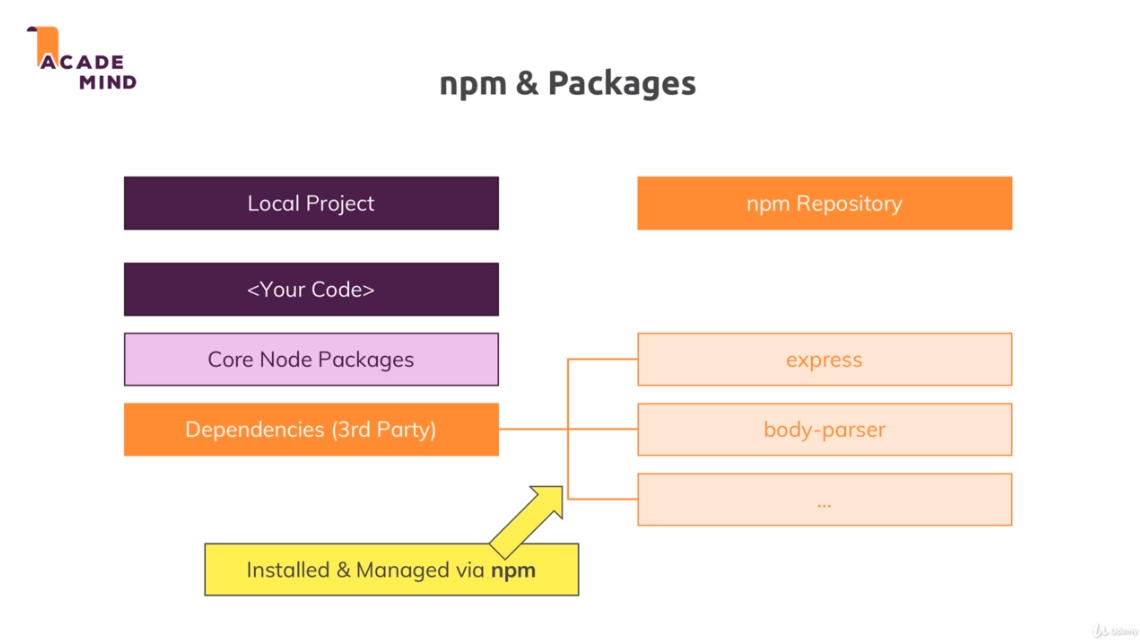
"start": "node app2.js",

"start-server": "node app2.js"

},

* Npm start (OK)
* Npm start-server (error)
* Npm run start-server(OK)

1. NPM and Packages



1. Global Module VS Core Module VS 3rd Party Module

Global Features vs Core Modules vs Third-Party Modules

Section 4, Lecture 43

The last lectures contained important concepts about available Node.js features and how to unlock them.

You can basically differentiate between:

* **Global features**: Keywords like const or function but also some global objects like process
* **Core Node.js Modules**: Examples would be the file-system module ("fs"), the path module ("path") or the Http module ("http")
* **Third-party Modules**: Installed via npm install - you can add any kind of feature to your app via this way

**Global features** are **always available**, you don't need to import them into the files where you want to use them.

**Core Node.js Modules** don't need to be installed (**NO npm install** is required) but you **need to import them** when you want to use features exposed by them.

Example:

const fs = require('fs');

You can now use the fs object exported by the "fs" module.

**Third-party Modules** **need to be installed**(via npm install in the project folder) **AND imported**.

Example (which you don't need to understand yet - we'll cover this later in the course):

1. // In terminal/ command prompt
2. npm install --save express-session
3. // In code file (e.g. app.js)
4. const sessions = require('express-session');

Browse Q&A

Continue

Global & Local npm Packages

Section 4, Lecture 45

In the last lecture, we added nodemon as a local dependency to our project.

The good thing about local dependencies is that you can share projects **without the node\_modules** folder (where they are stored) and you can run npm install in a project to then re-create that node\_modules folder. This allows you to share only your source code, hence reducing the size of the shared project vastly.

The attached course code snippets also are shared in that way, hence you need to run npm install in the extracted packages to be able to run my code!

I showed that nodemon app.js would **not work** in the terminal or command line because we don't use local dependencies there but global packages.

Only Global Package is used with its own name. We need to install again the package in a shared project, though. Otherwise, the local packages is used with npm. We do not need to reinstall it. Just implement npm install.

You could install nodemon globally if you wanted (this is NOT required though - because we can just run it locally): npm install -g nodemon would do the trick. Specifically the -g flag ensures that the package gets added as a global package which you now can use anywhere on your machine, directly from inside the terminal or command prompt.

1. Error: Syntax error, Runtime Error, and Logical Error
   * Syntax error : error message with red line
   * Runtime Error: error message (like can’t find the module, or can’t stop at the spot.)
   * Logical error: no error message but got incorrect value we did not expect
2. Debug for Logical Error

Debug Console -> at the end of the terminal, write a command that executed in the code.

7.‘debugger’ Update

* + Menu -> Debug -> Add Configuration
  + Must use nodemon in global
  + (할 수 있는 일)
  + 옆에 창에서 현재 variable 상태 확인
  + 옆에 창 Variables에서 value 직접 변경 및 결과 값 확인
  + Debug terminal에서 먼저 함수 값 처리 및 결과 확인
  + 이전 line을 breakpoint로 설정 하고PLAY Button눌러서 다시 시작
  + -g nodemon사용해서 값 변경 및 자동으로 변경된 값 확인
  + Local nodemon 은 반드시 스톱하고 debugmode에서 시작 및 브라우저에서 값입력

{

// Use IntelliSense to learn about possible attributes.

// Hover to view descriptions of existing attributes.

// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

"program": "${workspaceFolder}/firstServer\\app2.js",

"restart": true,

"runtimeExecutable": "nodemon",

// need to install nodemon globally

// we can still use debug console, by the way.

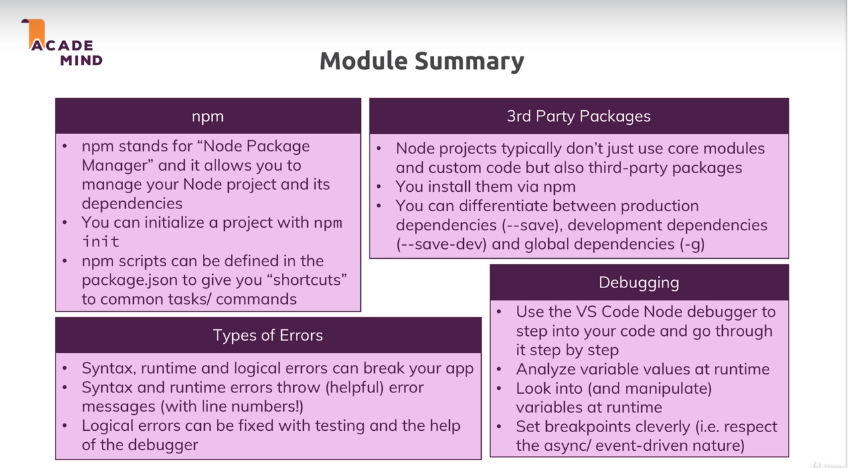
"console": "integratedTerminal"

}

]

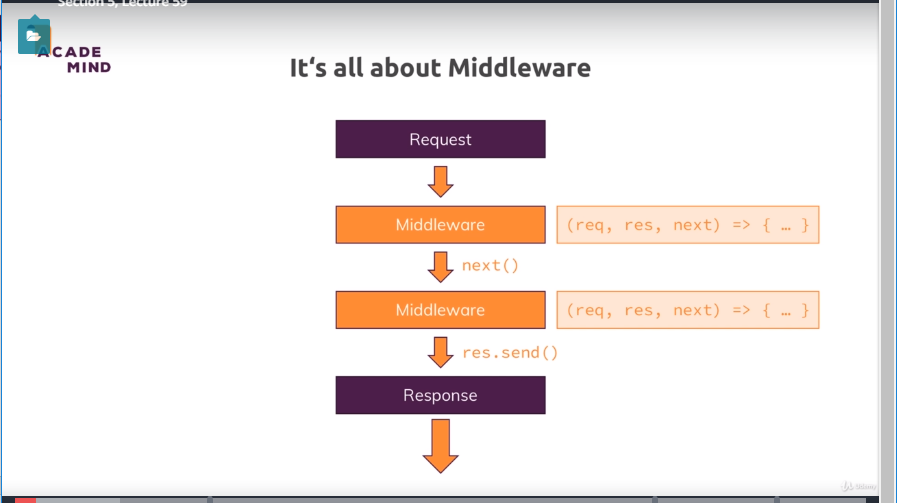
}

1. Wrap up

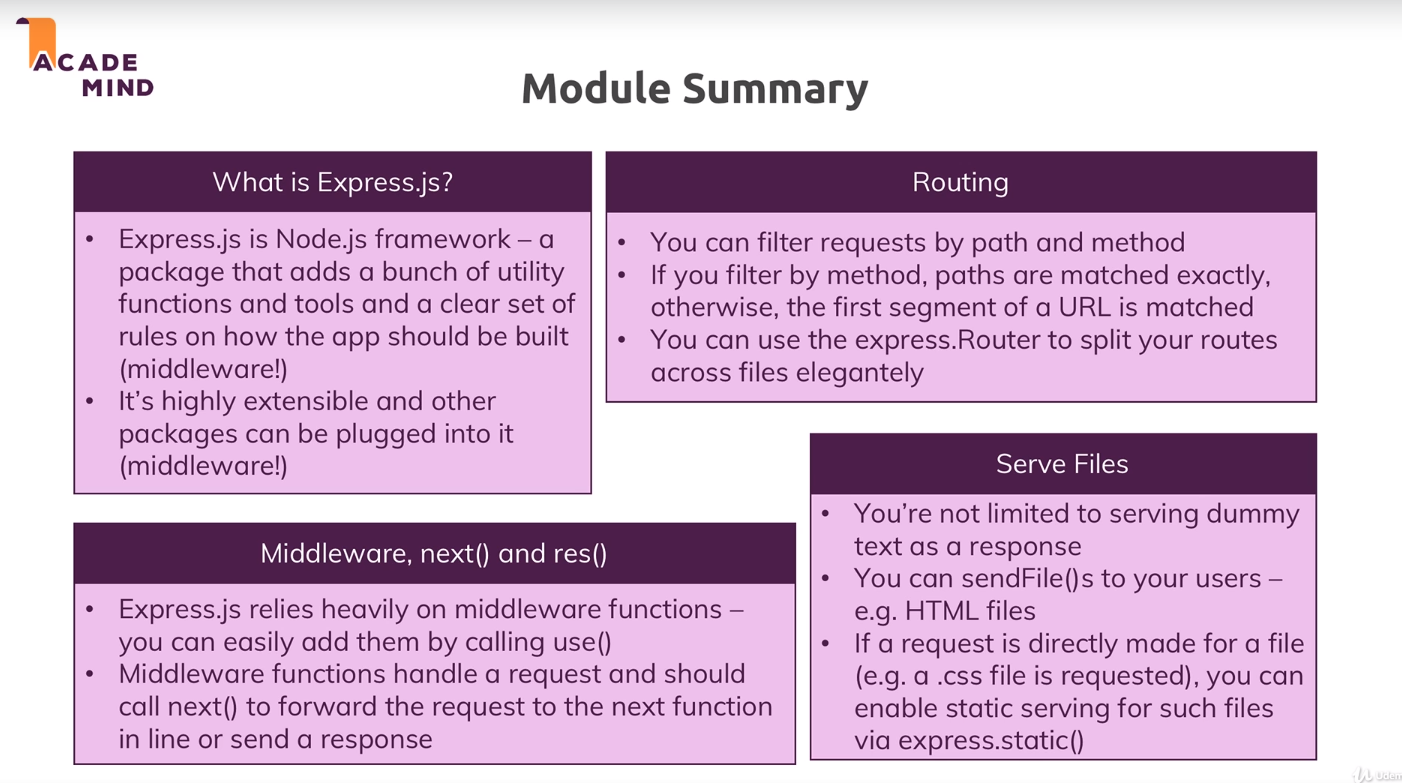


/\* Express.js \*/

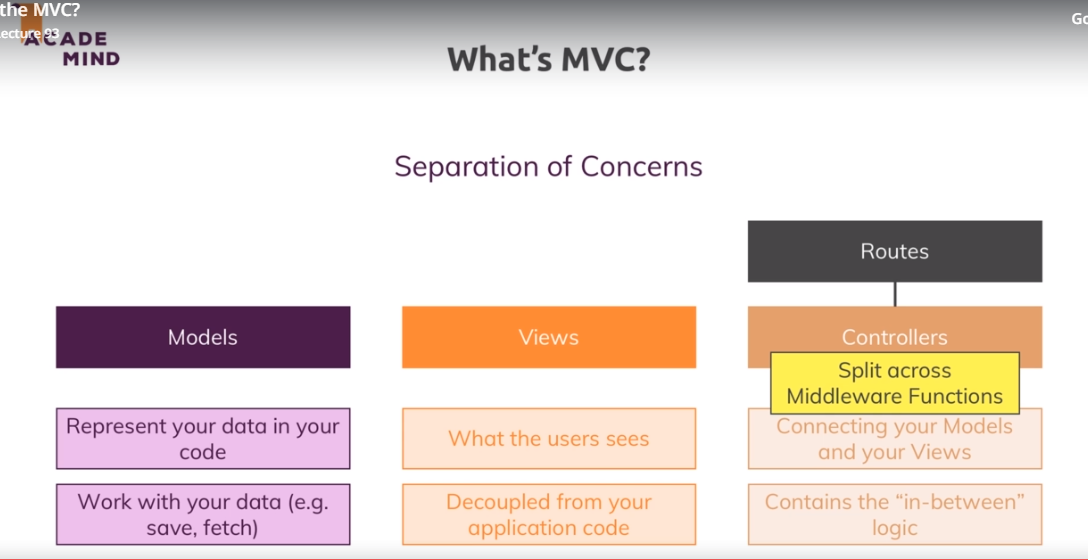
1. Middleware

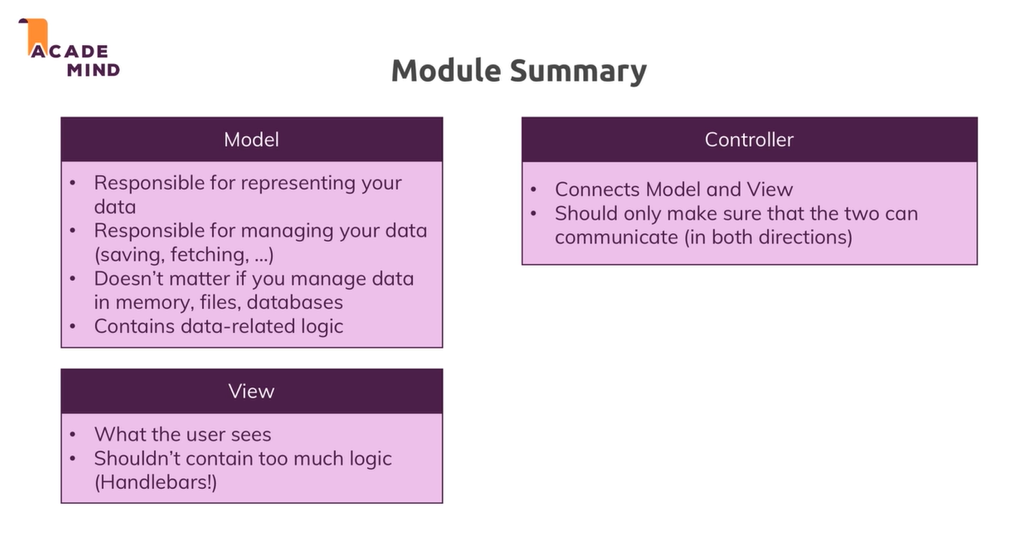


1. Express.js

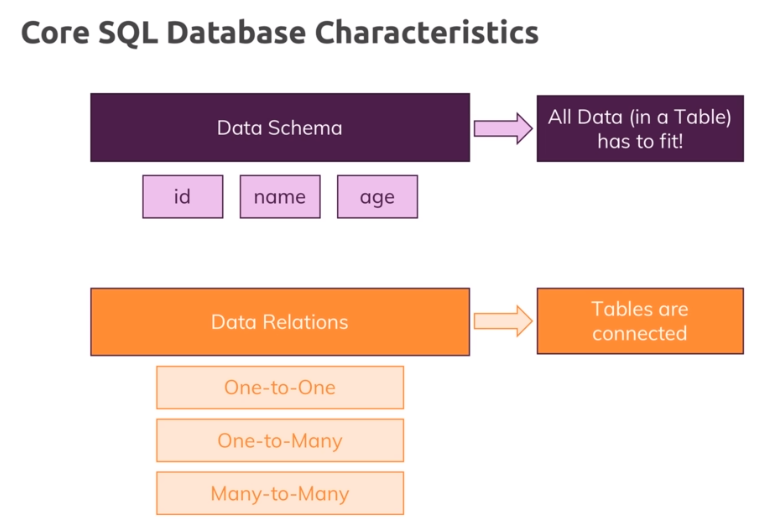


1. MVC (Model, View, and Controller)



ss

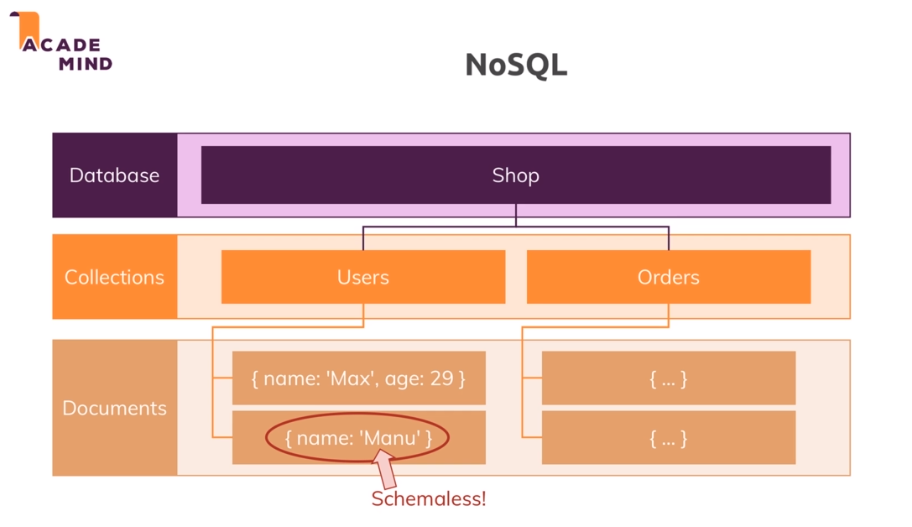
SQL

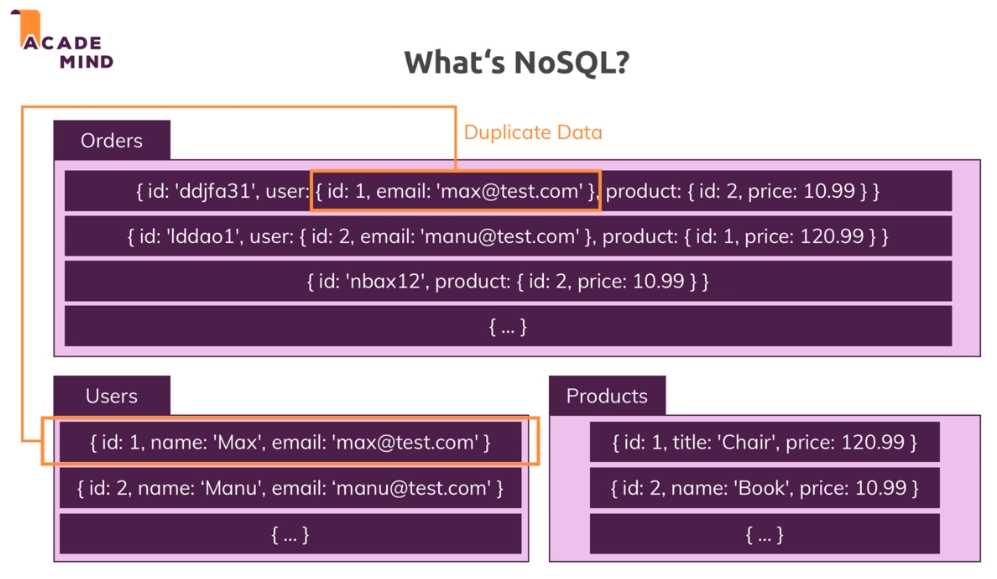


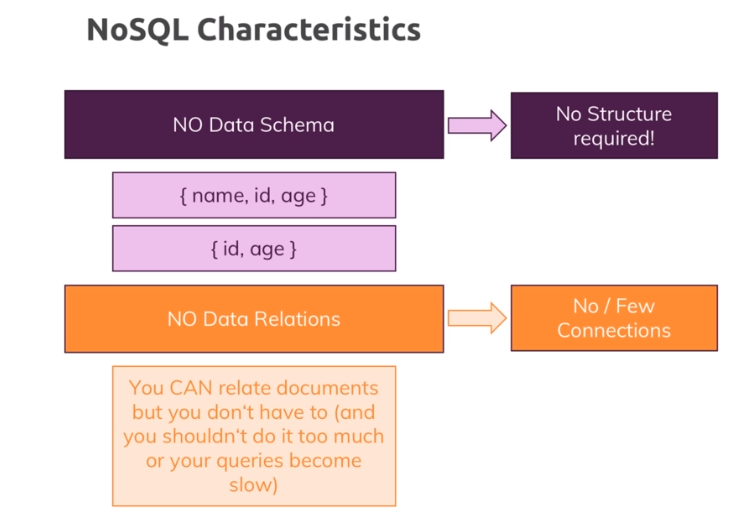
NoSQL.COLLECTION === SQL.TABLE

NoSQL does not have shema in a document.

Therefore, the noSQL can have different structures in a collection.

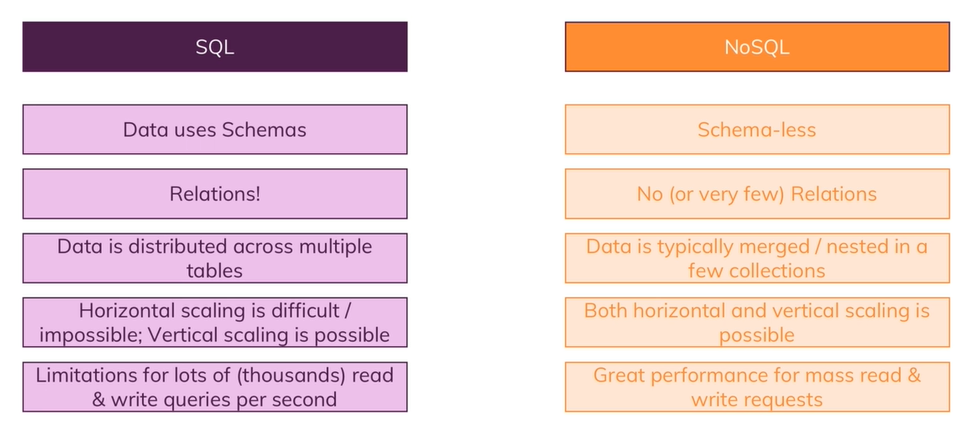






No relations: Each collections does not have any relation to other collection. Instead, It directly stores id/user value in the document to independently exist as a table/collection. However, in SQL, it does not directly store the user value into product database because all data are artificially related through primary and foreign key.

In NoSQL, once a data in a collection is switched, it does not automatically change in the other collection. We should save the switched data. If it is automatically saved, it might consume a lot performance (as in case of SQL). Instead, for the better performance, data can be easily manipulated, merged and nested in a collection.



MySQL Setup

1. Go to MySQL web
2. Select download
3. Select ‘MySQL Community’ at the bottom of the page
4. For window, we can use [**MySQL on Windows (Installer & Tools)**](https://dev.mysql.com/downloads/windows/)  only
5. For window (if you do not like MySQL ON Window), mac and linux, we need to download both MySQL Community SERVER and MySQL WorkBench.
6. Try to download MySQL Community Server. Then select the zip based msi file, which is ok with 32bit. (For now, it should not be web installer)
7. And select ‘[**No thanks, just start my download.**](https://dev.mysql.com/get/Downloads/MySQL-8.0/mysql-8.0.15-winx64.zip)**’ At the bottom.**
8. Execute the app.
9. Select “custom”
10. MOVE SQL SERVER AND SQL Workbrench into THE RIGHT WINDOW.
11. Next
12. Ignore python ~~~~~~
13. Next and Execute
14. Next and Next as it is…
15. Select Legacy Authentication
16. Input password here 1111
17. Next and next and finally finish

If I use mac/ linux MySQL Workbench should be downloaded again and installed. For window, it is ok, now.

Then,

16) Connect MySQL by using MySQL bench (at the left side bar)

17) Create a new schema by right clicking the Shema area, not the button

18) Flip over to bash shell and npm I –save mysql2

Regarding for Table setup, please refer to the Terminology below.

PK - Primary Key

NN - Not Null

BIN - Binary (stores data as binary strings. There is no character set so sorting and comparison is based on the numeric values of the bytes in the values.)

UN - Unsigned (non-negative numbers only. so if the range is -500 to 500, instead its 0 - 1000, the range is the same but it starts at 0)

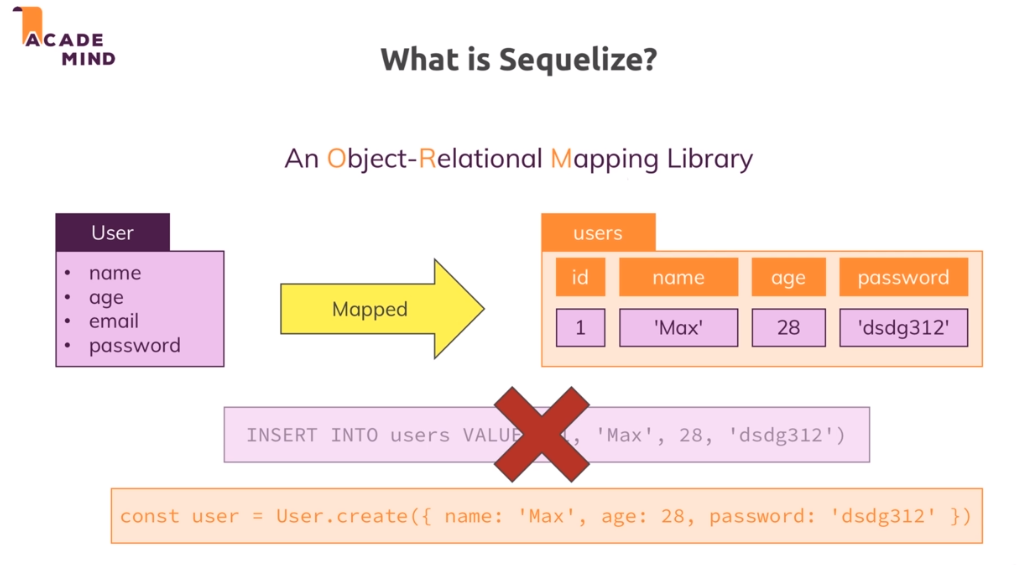
UQ - Create/remove Unique Key

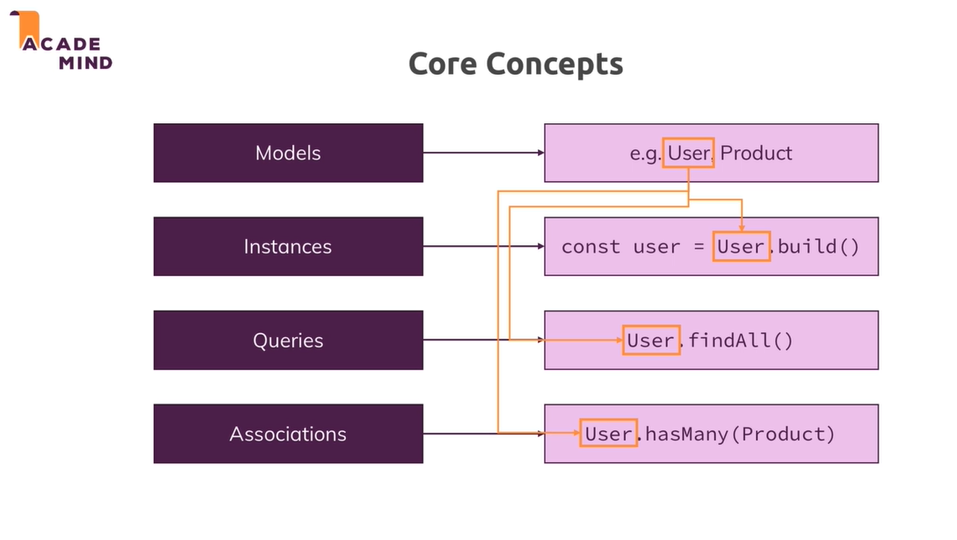
ZF - Zero-Filled (if the length is 5 like INT(5) then every field is filled with 0’s to the 5th digit. 12 = 00012, 400 = 00400, etc. )

AI - Auto Increment

G - Generated column. i.e. value generated by a formula based on the other columns

Sequelize



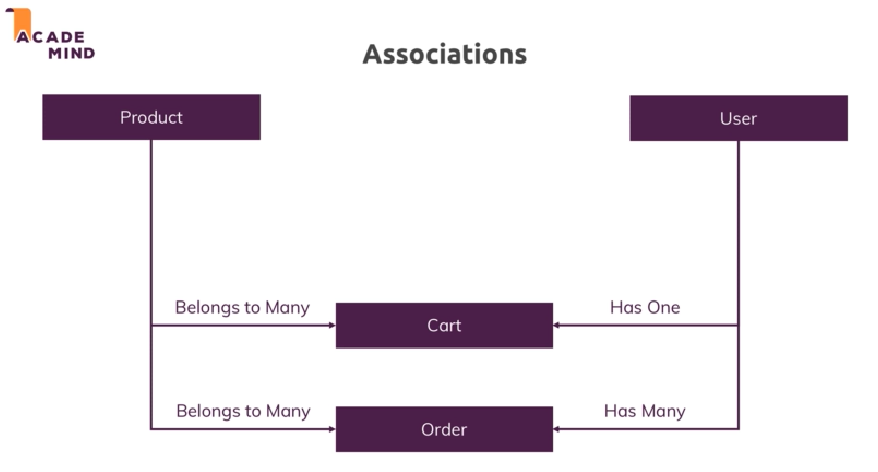


Sequelize Setup : npm I –save sequelize

Prerequisite : mysql2 must be installed ahead.

SEQUELIZE - ASSOCIATION

: Many one to many associations here.



[Setup Associations by using sequelize ]

// [Association]

// Before all the models are up

// we need to define associations

// If the USER deletes, products will be gone, as well.

/\*

ASSOCIATION is set in console.log()

userId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`userId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB; \*/

Product.belongsTo(User, { constraints: true, onDelete: 'CASCADE'});

// optional but it should be explicitly defined.

User.hasMany(Product);

// All the models up \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// and create their tables (if the tables do not exist) and relations;

// { force: true }

// Optional in development: when we need to overrite the data into database.

// Actually, it drops the existing tables, then create the table again, and insert into data again.

/\*

Executing (default): DROP TABLE IF EXISTS `products`;

Executing (default): DROP TABLE IF EXISTS `users`;

Executing (default): DROP TABLE IF EXISTS `users`;

Executing (default): CREATE TABLE IF NOT EXISTS `users` (`id` INTEGER NOT NULL auto\_increment , `name` VARCHAR(255) NOT NULL, `email` VARCHAR(255) NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;

Executing (default): SHOW INDEX FROM `users`

Executing (default): DROP TABLE IF EXISTS `products`;

Executing (default): CREATE TABLE IF NOT EXISTS `products` (`id` INTEGER NOT NULL auto\_increment , `title` VARCHAR(255) NOT NULL, `price` DOUBLE PRECISION NOT NULL, `imageUrl` VARCHAR(255) NOT NULL, `description` VARCHAR(255) NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `userId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`userId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB;

Executing (default): SHOW INDEX FROM `products`

\*/

// A

sequelize.sync({force: true})

.then((result) => {

// finally after all models (tables) are created

// of course, and all the apps (express) correctly are up as weel

// listen to requests form the client.

app.listen(3000, () => {

console.log('Port: 3000')

});

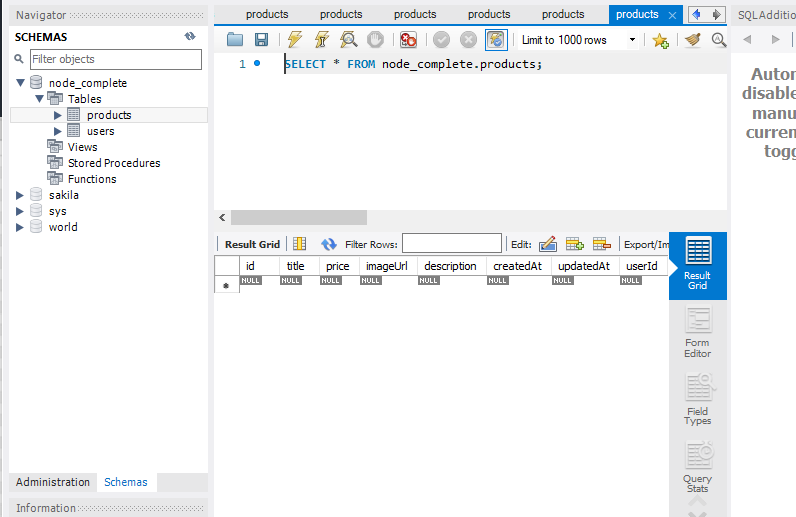
})

.catch(e => {

console.log(e);

});

Foreign key userId is automatically created.



Association definition

The key difference is that in a One To Many relationship, you only need the two database tables that correspond to the related models. This is because the reference to the relation is *stored on the owned model's table itself*. For instance, you might have a Country model and a City model. A Country has many cities. However, each City only exists in one country. Therefore, you would store that country *on the City model itself* (as country\_id or something like that).

However, a Many To Many relationship requires a *third* database table, called a **pivot table**. The pivot table stores references to both the models and you can declare it as a second parameter in the relationship declaration. For example, imagine you have your City model and you also have a Car model. You want a relationship to show the types of cars people drive in each city. Well, in one city people will drive *many* different types of car. However, if you look at one car type you will also know that it can be driven in *many* different cities. Therefore it would be impossible to store a city\_id or a car\_id on either model because each would have more than one. Therefore, you put those references in the pivot table.

As a rule of thumb, if you use a belongsToMany relationship, it can *only* be paired with another belongsToMany relationship and means that you have a third pivot table. If you use a hasManyrelationship, it can *only* be paired with a belongsTo relationship and no extra database tables are required

// [

// (belongs to : One-to-one association)

// option should be explicitly defined.

Product.belongsTo(User, { constraints: true, onDelete: 'CASCADE' });

// (hasMany : one to many association)

/\*

hasMany is used in a One To Many relationship

while belongsToMany refers to a Many To Many relationship.

They are both distinct relationship types and each require

a different database structure - thus they take different parameters.

\*/

User.hasMany(Product);

// ]

// [

// (hasOne: One-to-one association)

User.hasOne(Cart);

// (belongsTo: One-to-one association)

Cart.belongsTo(User);

// ]

// [

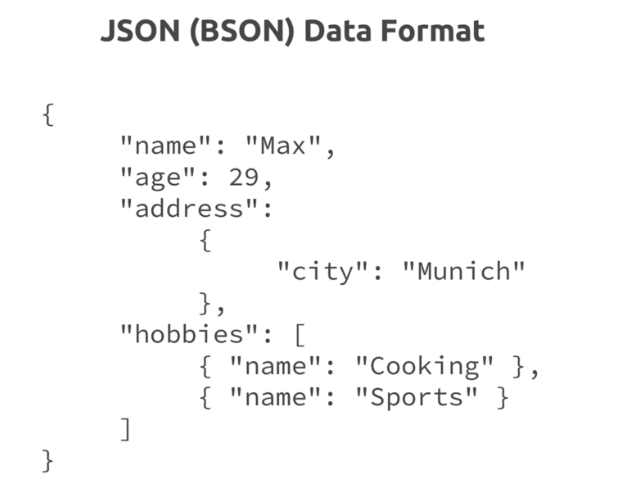
// (belongsToMany: many to many association)

Cart.belongsToMany(Product);

Product.belongsToMany(Cart);

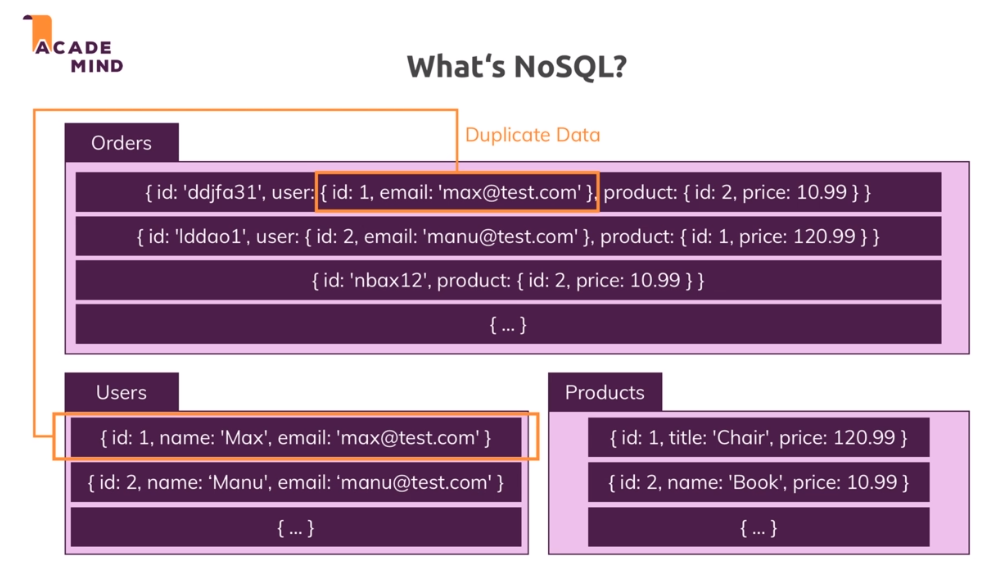
// ]

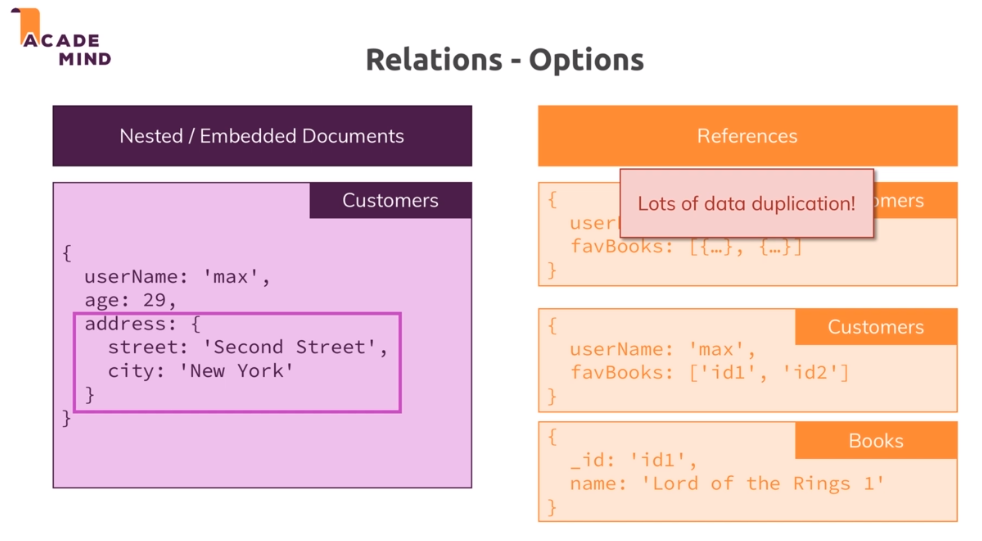
MongoDB

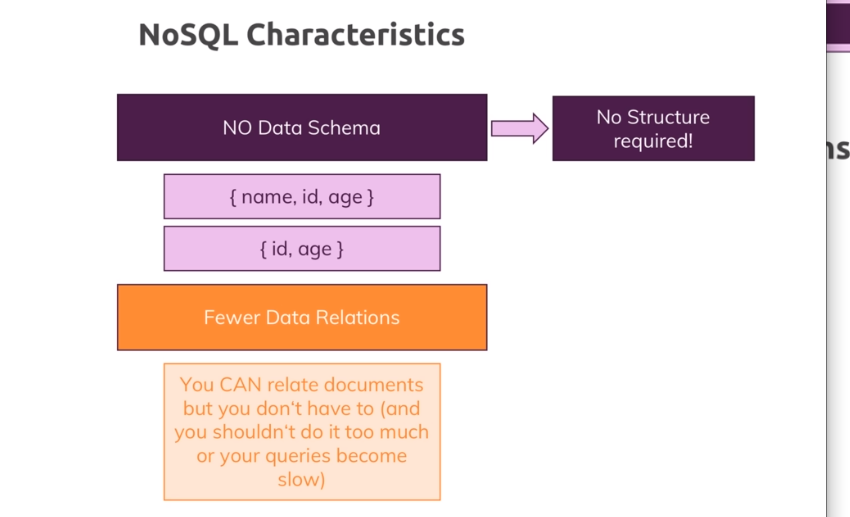
1. MongoDB format based on binary JSON.
2. Address : embedded (nested) document
3. Also, it can have arrays.
4. Important : MongoDB is schemaless database. Therefore, it does not depend on the schema definition.
5. 

What is noSQL?

As shown in the diagram below, a document data is duplicated in the collection. At this point, we do not need to merge tables. We just need to need to fetch with the key word.







# Overview

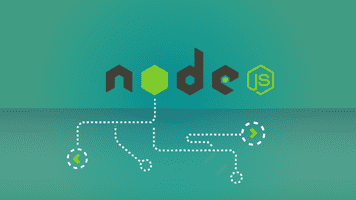
After taking your first steps into web development, you’ll relatively early be confronted with one choice: **Should you work with a SQL database like**[**MySQL**](https://mysql.com/)**or a NoSQL database like**[**MongoDB**](https://mongodb.com/)**?**

Especially if working with [Node.js](http://nodejs.org/) you could quickly get the impression that NoSQL (MongoDB) is strictly better.

This, however, is wrong!

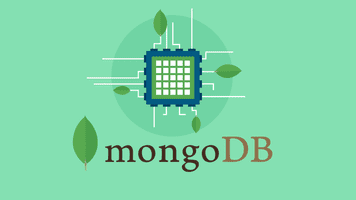
In this article, I’ll quickly introduce you to the core concepts of both database worlds (SQL and NoSQL) and then highlight the differences and advantages or disadvantages of each solution.

Not a reader? Watch the video above this article!

[C:\Users\joona\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1F08DA0C.tmp[](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W)](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W" \t "_blank)

# [Node.js - The Complete Guide](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W" \t "_blank)

[Learn Node.js from A - Z with this bestselling, high-rated complete guide!](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W" \t "_blank)

[C:\Users\joona\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E16923B3.tmp[](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W)](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W" \t "_blank)

# [MongoDB - The Complete Guide](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W" \t "_blank)

[Every web app needs a database, MongoDB is one of the best ones you can dive into. Learn all about it from scratch!](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W" \t "_blank)

# [#](https://academind.com/learn/web-dev/sql-vs-nosql/#sql-databases)SQL Databases

SQL stands for Structured Query Language and it therefore is **not** a database itself but only a query language you can use to interact with a specific type of database.

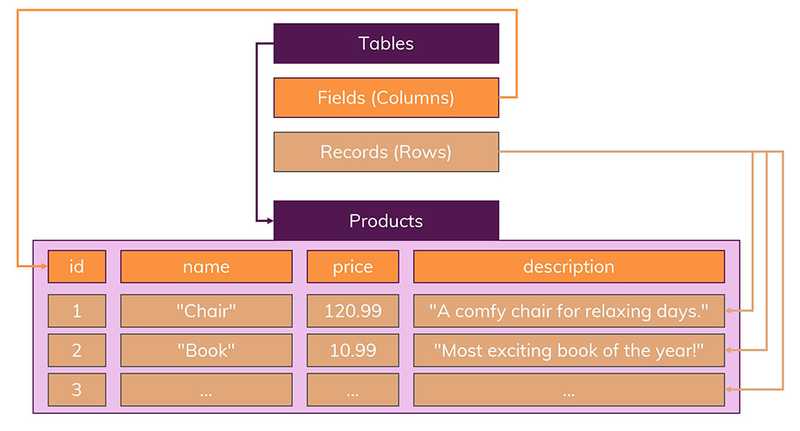
SQL allows you to store, update, delete and of course retrieve data from **relational database management systems** ([RDBMS](https://techterms.com/definition/rdbms)).

Such databases have two key characteristics:

* Data is stored in database tables by following a strict data schema (= structure)
* Data is distributed across multiple tables which are connected via relations

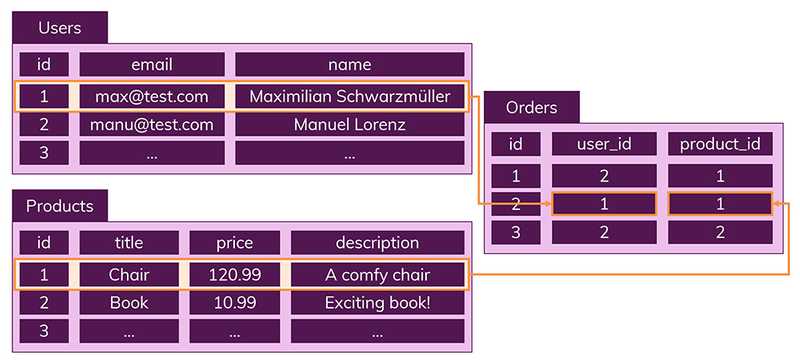
# [#](https://academind.com/learn/web-dev/sql-vs-nosql/#strict-schema)Strict Schema

Data is stored as records in tables and each table has a clearly defined structure - a set of fields which defines which data may go into the table and which data may not.

The structure is defined regarding the names of the fields as well as the datatypes.[](https://academind.com/static/c6c8b088e9d9dd4722a965cde6b76e0d/ece6b/sql-schema.jpg)You can’t add records which don’t adhere to this schema. Got more fields? Sorry, you have to pick another table. Missing some field data? Not your table then!

# [#](https://academind.com/learn/web-dev/sql-vs-nosql/#relations)Relations

The other important part of SQL-based databases are relations.

You split data into multiple tables so that you avoid data duplication. Therefore, you’ll have a Users, Products and Orders table but each table will only hold data that is not stored in one of the other tables.[](https://academind.com/static/5df24f0f34a3d98feb531b5fc7776f72/5a312/sql-relations.jpg)This clear structure can have advantages - you won’t end up with incorrect data in one of your tables whilst having correct one in all others. That’s a case you won’t run into because data is always only managed in one table, it is not duplicated across tables.

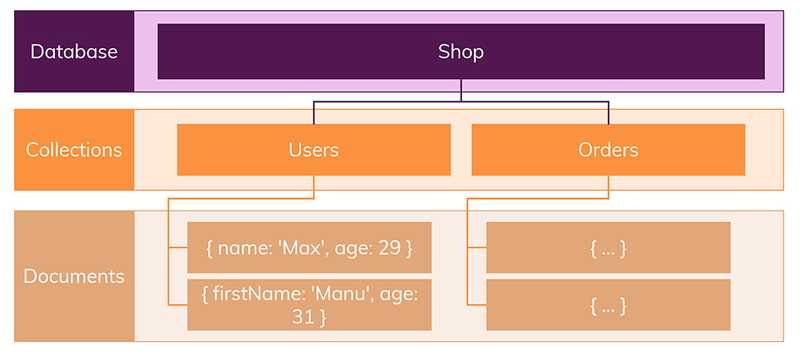
# [#](https://academind.com/learn/web-dev/sql-vs-nosql/#nosql-databases)NoSQL Databases

NoSQL is named like this because it basically follows the opposite approach of SQL databases.

* No schemas
* No relations

That’s how you could summarize it.

Instead, you structure data in collections (= tables in the SQL world). Records are now called documents.

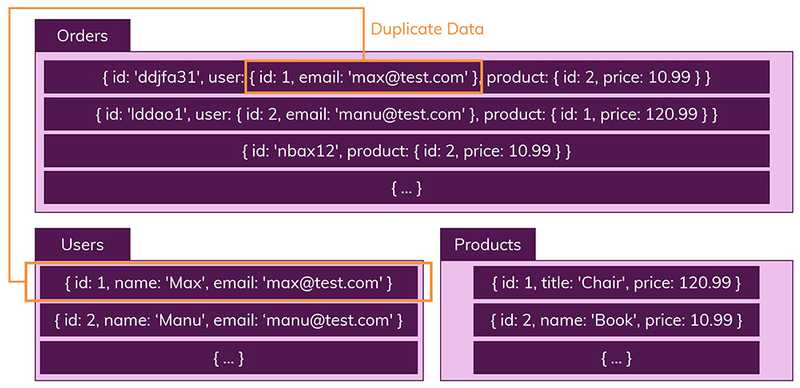
But it’s not just about the naming, there is a core difference: You can put data of different structure into the same collection. You couldn’t do that in the SQL world - each table had a clear schema there.Documents look a bit like [JSON](https://www.json.org/) data - and as mentioned, you don’t need to worry about any schema.

Additionally, you typically put related data into the same collection. So if you got a bunch of orders, you would store your order documents in your Orders collection by including all data you typically query.

So you would end up with documents that already got everything you need - no need to join multiple tables/ collections.

Indeed, NoSQL databases don’t know the concept of joining tables/ collections.

You can manually do that (by retrieving a foreign id in collection A and looking it up in collection B) but this will **not** be your typical flow.

Instead, you duplicate data across collections so that each collection yields exactly the data some part of your app might be looking for.[](https://academind.com/static/bbea2ff32393dedce24d2114b26254fb/ece6b/nosql-no-relations.jpg)The concept of data duplication looks disturbing at first. Doesn’t that introduce the danger of making mistakes and updating data in collection A without adjusting it in collection B, too?

It does and it will be your job to ensure that data updates are performed in all collections that use a certain data piece.

But the huge advantage is that you don’t need to work with complex (and at some point slow) join statements. All the data is already stored in the structure you need it in.

This is especially great for data that doesn’t change a dozen times every second.

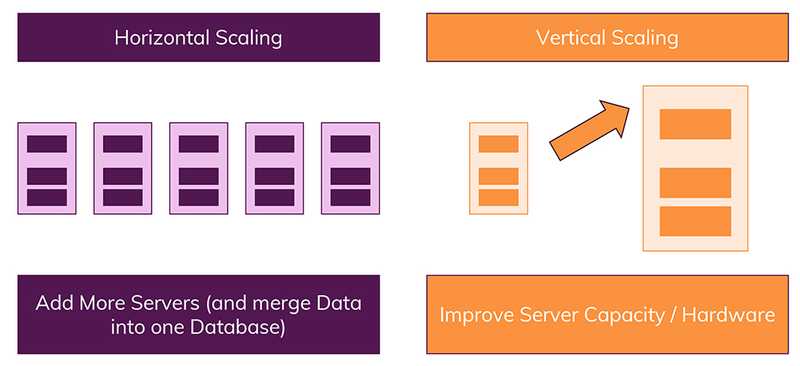
# [#](https://academind.com/learn/web-dev/sql-vs-nosql/#vertical-horizontal-scaling)Vertical & Horizontal Scaling

There’s also one other important concept we have to have a look at when comparing databases: **Scaling**.

How well can you scale your database?

With that I mean how many read and write requests is your database able to handle. Which amounts of data can it work with?

When it comes to scaling, we can differentiate between **vertical** and **horizontal** scaling.

* **Vertical Scaling** means that we simply increase the power of the database server - e.g. by upgrading its CPU.
* **Horizontal Scaling** on the other hand means that more servers are added and the database is distributed across them. Hence you still work with one database but multiple servers that host it.[](https://academind.com/static/3d5e1fd10206c1c76da6214c01c7a5f4/5a312/horizontal-and-vertical-scaling.jpg)Due to the way data is stored (related tables vs unrelated collections), SQL databases generally support vertical scaling only - horizontal scaling is only possible for NoSQL databases.

SQL databases do know the concept of [sharding](https://docs.microsoft.com/en-us/azure/architecture/patterns/sharding) but it comes with certain restrictions and is typically hard to implement. NoSQL databases natively support this and therefore make it way easier to split your database across multiple servers.

# [#](https://academind.com/learn/web-dev/sql-vs-nosql/#the-right-choice)The Right Choice

With the brief overview given, the question is: **Which database solution should you work with?**

There is no clear winner!

Both SQL and NoSQL are absolutely viable solutions. But they’re solutions to different problems - it comes down to your data and application in the end.

Let’s summarize the key **advantages** of both approaches:

## SQL

* Clearly defined schema, data integrity is ensured
* Relations allow you to store each data only once - no duplicates

## NoSQL

* Absence of a schema gives you more flexibility - you can adjust your stored data at any point and introduce new “fields”
* Data is stored in the format your app needs it - this speeds up fetching the data
* Vertical **and horizontal** scaling is possible, hence your database will be able to handle any amount of read/ write requests your app throws at it

And the **disadvantages**:

## SQL

* Less flexibility, data schema needs to be known and planned in advance (adjusting it later is difficult or maybe even impossible)
* Relations can lead to very complex queries with a lot of JOIN statements
* Horizontal scaling is hard, often only vertical scaling is possible - this means that you’ll face some growth limits (regarding throughput you can handle/ performance) at some point

## NoSQL

* Increased flexibility might lead you to work sloppy and postpone data structure decisions
* Duplicate data means that you have to update multiple collections and documents if that data changes - not just one record in one table as you would do it in the SQL world

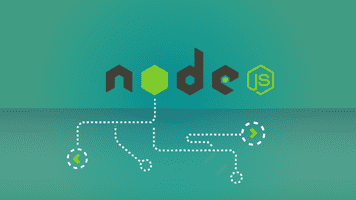
## So when might SQL be best?

* You got related data, used in different “chunks” in different parts of your app, that changes relatively often (you would have to update multiple collections all the time in a NoSQL world)
* A clear schema is important to you and your data is unlikely to change (drastically)

## When is NoSQL best?

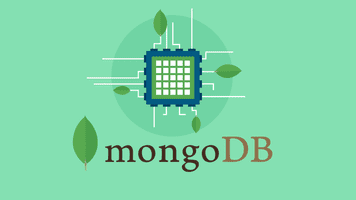
* Exact data requirements or the data itself is unknown or subject to change/ expand
* You require high (read) throughput but you won’t change your data that often (i.e. you don’t need to update dozens of documents for one change all the time)
* You need to scale your database horizontally (i.e. you store enormous amounts of data and have huge read and write throughput)

Obviously, you can structure your database in different ways. And depending on your structure, you might end up with a NoSQL database that mitigates some of the issues (e.g. you reduce the amount of duplicate write requests). The same is true for SQL databases: You can choose a structure that meets your data requirements and doesn’t lead to overly complicated JOIN statements.

[C:\Users\joona\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6FCD436B.tmp[](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W)](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W" \t "_blank)

# [Node.js - The Complete Guide](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W" \t "_blank)

[Learn Node.js from A - Z with this bestselling, high-rated complete guide!](https://www.udemy.com/nodejs-the-complete-guide/?couponCode=ACAD_W" \t "_blank)

[C:\Users\joona\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\8CEF2DFE.tmp[](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W)](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W" \t "_blank)

# [MongoDB - The Complete Guide](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W" \t "_blank)

[Every web app needs a database, MongoDB is one of the best ones you can dive into. Learn all about it from scratch!](https://www.udemy.com/mongodb-the-complete-developers-guide/?couponCode=ACAD_W" \t "_blank)

This site uses cookies. By continuing to browse the site you are agreeing to our use of cookies.

[MORE INFO](https://academind.com/info/cookies)OK

[IMPRESSUM & DATENSCHUTZ (DE)](https://academind.com/impressum)|[IMPRINT & DATA PRIVACY (EN)](https://academind.com/impressum/en)

