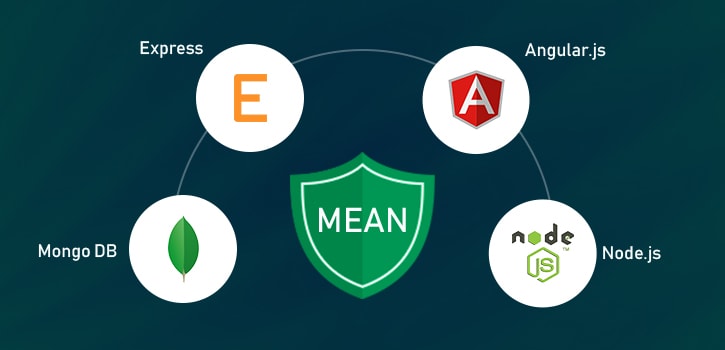
# Technology

## The MEAN Stack

Developers looking to host maximum user engagement on their web app choose the most feasible technologies and frameworks. There is a long list of technologies and the MEAN stack is a recipe to create modern web applications. Introduced in 2014, MEAN has transformed the way the web development process goes. As mentioned above, the four components in the MEAN Stack are:

* **MongoDB**, document database - used by your backend application to store its data as JSON (JavaScript Object Notation) documents.
* **Express.js**, a backend web application framework running on top of Node.js.
* **AngularJS**, a front-end [web application framework](https://www.peerbits.com/cross-platform-application-frameworks.html); runs your JavaScript code in the user's browser, allowing your application UI to be dynamic.
* **Node.js**, a JavaScript runtime environment - lets you implement your application backend in JavaScript.



The MEAN stack is all about using the same programming language for all the components of backend JavaScript development and [front-end development](https://www.peerbits.com/frontend-development.html). Access to the database is through MongoDB's idiomatic driver. Drivers facilitate interaction using typical JavaScript concepts such as objects and async execution, using either callback functions or promises.

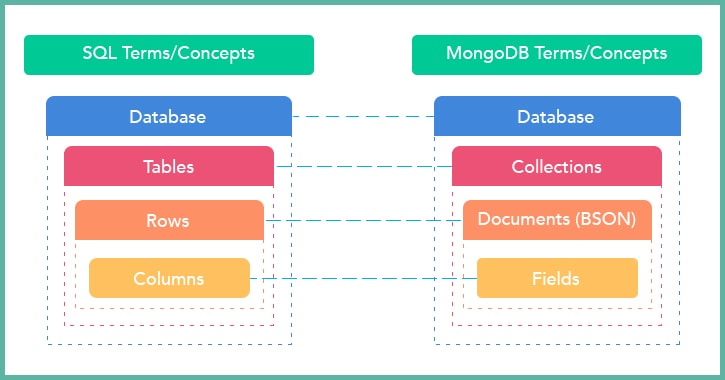
MongoDB is a NoSQL database that provides flexible data storage layers, and Node.js keeps your server running continuously. It harnesses the potential for a faster computer network. Express.js standardizes [web application development](https://www.peerbits.com/web-application-development.html) and Angular.js allows interactive gestures on web applications. Together, they form the MEAN stack.

Developers, who realized the LAMP stack (Linux, Apache, MySQL, and PHP) is not flexible enough to serve their purposes, are gradually making a move towards the MEAN stack. And it's high time you should too. There are many reasons to do so.

|  | **MEAN STACK** | **LAMP STACK** |
| --- | --- | --- |
| **Operating System** | Cross-platform | Linux OS only |
| **Database** | MongoDB, a ‘non-relational’ database | MySQL, a ‘relational’ database |
| **Programming Language(s)** | Front-end—JavaScript backend—PHP | JavaScript, back- and front-end |
| **Structure** | Slower and limited due to its blocking structure. | faster & scalable due to its non-blocking structure. |
| **Technologies** | MongoDB, Express.js, AngularJS, Node.js | Linux, Apache, MySQL, PHP |
| **Mainstream Backers** | Google, IBM, Samsung | Oracle, Zend, Linux Foundation |

## The Database System: MongoDB

MongoDB has the best of both worlds. It has the fast, scalable key-value stores of NoSQL and the rich functionality of relational databases. Unlike relational databases, MongoDB stores BSON documents inside collections with active schemas.

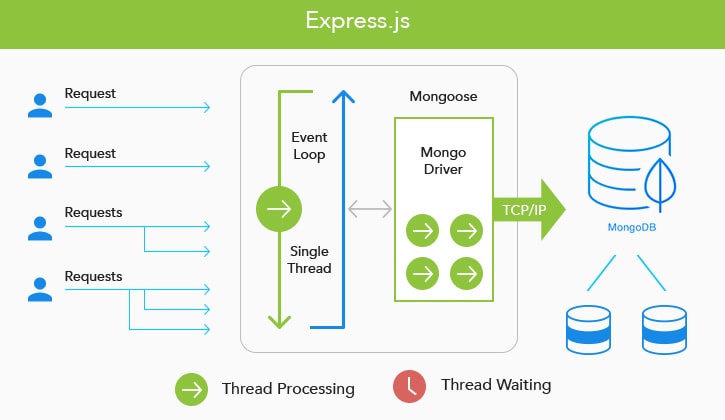


While relational databases like MySQL store data in specific rows and columns, MongoDB's document data model stores data regardless of its structure and doesn't void existing authentication rules, data access, or indexing functionality. You can actively adjust the schema without interruption.

## The Backend Web Framework: Express.js

Express is a web application framework that executes backend application (JavaScript) code. Express is a module Node.js environment and runs under it.

Express handles the allocation of requests to different parts of an application (or to various apps running in the same environment).

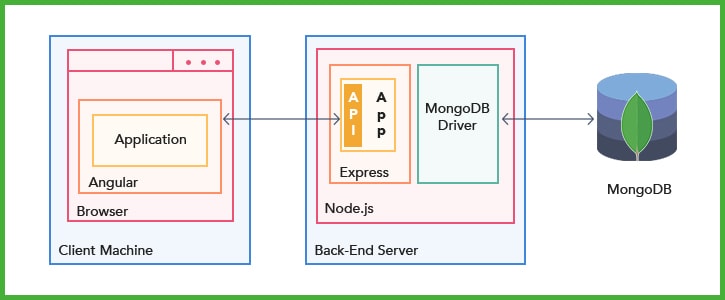


Express executes the part of a web app's code that is responsible for business logic and returns an HTML file to be compiled by a web browser. However, Express mostly gives RESTful APIs that the front-end can access with a single line of code.

## The Frontend Framework: AngularJS

Angular renders JavaScript code in a web browser's environment to make way for a reactive user interface (UI). Unlike static web forms where you have to fill in the form and hit submit, reactive UI reflects a user's input without refreshing the entire page, but only the essential components.

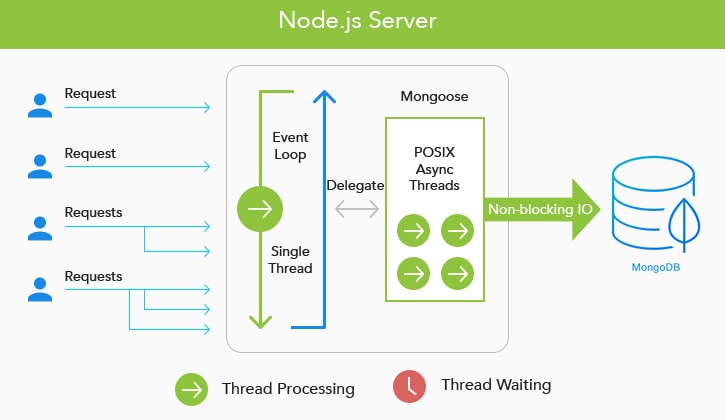
You implement your web app front-end as a group of components. JavaScript code attached to an HTML template contains hooks to execute and employ the results from your JavaScript functions. Multifaceted application front-ends can be built from many simple or nested components.



Code for Angular applications can compile in the backend server rather than in the front-end browser, or as a native desktop or mobile app.

## The Backend Runtime Environment: Node.js

A JavaScript runtime environment to run your backend application with the help of Express, Node.js is based on Google's V8 JavaScript engine, which powers Google Chrome and the open-source Chromium browser. Many of its modules are essential for deploying web applications on the MEAN stack - including HTTP. Third-party modules can be installed with npm.



Node.js is asynchronous and event-driven. That is, an application doesn't poll an acknowledgment to a request. It resumes other tasks in the meantime. Upon completion, a callback request informs the application and it can start other tasks for optimal resource utilization, which enables many parallel operations while scaling applications. MongoDB can work asynchronously too and, thus, works well with Node.js applications.

If you are still not sold on the MEAN stack, we will take you through why you should be using it for your next web development project.

## MEAN Stack Advantages

* **One Language to Rule Them All: JavaScript**

Every piece of code in the MEAN stack is JavaScript. On the other hand, in the LAMP stack, the server-side is coded in PHP, the client-side in JavaScript, and queries are sent using MySQL. With MEAN, MongoDB allows you to work with JavaScript - it's the binary version of JSON. In a nutshell, the same language works, on both the client- and server-side.

* **Cost-Effective While Fast and Scalable**

A team of JavaScript developers can code for the server-side and client-side using the MEAN stack. So, no need to hire separate teams for different technologies. The same developer can switch roles between a front-end and backend developer. This will save you a lot of money. For the first time in the history of web development, a full-stack developer needs to know only one language.

* **Advantages of Node.js**

Node.js is a crucial part of the MEAN stack and is a very scalable and robust framework. Node.js is the runtime environment and is compliant with non-blocking input and output. Thus, a new query is handled efficiently. In the LAMP stack, the client had to direct a request for the query to be created. With Node.js, this all happens automatically.

* **The MongoDB Advantage**

MongoDB is the database web developers need while managing huge amounts of data. The whole thing works like OOPs and makes way for easy, effective handling.

* **Open-Source-Edit the Source, Compile, and Install**

The technologies at use in the MEAN stack are all open source. That is, developers get access to the code for each technology and can modify it to meet their individual needs. The technologies don't cost a buck to use and distribute, which means, not only can you use it for personal use but you can install them on your employees' personal workspaces.

* **Good Community Support**

A large community of individual and corporate contributors and sponsors back the technologies in the MEAN stack and the MEAN stack itself. When you are unable to debug a nasty piece of code, there are people there to help.

**BUILDING A RESTFUL API USING NODE AND EXPRESS**

**CONFIGURING OUR NODE / EXPRESS APPLICATION**

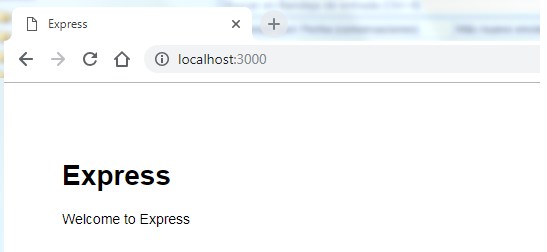
*> cd <workspace>>*

*<workspace>> express com.syss.app.percint.backend*

*<workspace>> cd com.syss.app.percint.backend*

*<project>> npm install*

*<project>>* SET DEBUG=com.syss.app.percint.backend:\* & *npm start*

**



If you are behind a proxy, you have to add its configuration before to use the npm command:

npm config set proxy http://proxy.company.com:8080

npm config set https-proxy http://proxy.company.com:8080

**Note**: The https-proxy doesn't have https as the protocol, but http.

Here, you have a .npmrc example file:

registry=http://registry.npmjs.org/

strict-ssl=false

proxy=http://<username>:<password>@proxy:80

https-proxy= http://<username>:<password>@proxy:80

**CONFIGURING MONGOOSE**

*< project>> npm install -g mongoose*

*< project>> npm install -g body-parser*

In order to handle PUT and DELETE request, you must install method-override:

*< project>> npm i --save method-override*

*< project>> npm install*

**UPLOADING YOUR PROJECT TO GIT**

*< project>> git init*

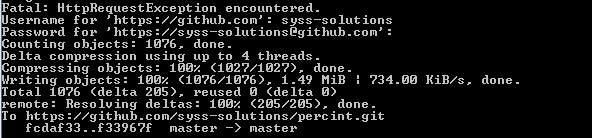
*< project>> git add .*

*< project>> git commit -m "20181212 - First commit"*

*< project>> git remote add origin* [*https://github.com/syss-solutions/base.git*](https://github.com/syss-solutions/base.git)

*< project>> git remote -v*

*< project>> git push origin master*



**Note:** If your updates were rejected because the remote contains work that you dont have locally, you must firstly integrate the remote changes with 'git pull origin master' command.

**Note:** If you need config a proxy server, use this command:

*git config --global http.proxy http://<username>:<password>@<proxy\_url>:<proxy\_port>*

*git config --global https.proxy http://<username>:<password>@<proxy\_url>:<proxy\_port>*

To disable this configuration:

*git config --global --unset-all http.proxy*

*git config --global --unset-all https.proxy*

**AUTHENTICATING WITH JWT (JSON WEB TOKENS)**

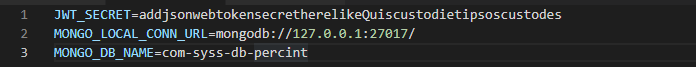
To authenticate our RESTFUL API, we need to install a set of npm packages apart from the current ones:

* bcrypt: We'll use this to hash our passwords before we save them our database.
* dotenv: We'll use this to load all the environment variables we keep secret in our .env file.
* jsonwebtoken: This will be used to sign and verify JSON web tokens.
* morgan: This will log all the requests we make to the console whilst in our development environment.
* nodemon: We'll use this to restart our server automatically whenever we make changes to our files.
* cross-env: This will make all our bash commands compatible with machines running windows.

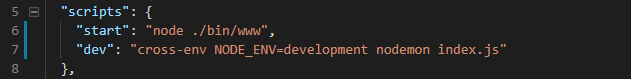
*< project>> npm install bcrypt dotenv jsonwebtoken --save*

*< project>> npm install morgan nodemon cross-env --save-dev*

Now, let’s create our .env file (in the project root) with our environment variables:



Inside the already existing scripts object in the package.json file, add the dev key value pair:



We'll now start our server with the **npm run dev** command.

Every time we do this, development is automatically set as a value for the NODE\_ENV key in our [process](https://nodejs.org/docs/latest/api/process.html#process_process_env) object. The command nodemon index.js will allow nodemon to restart our server every time we make changes in our folder structure.

Let's define the port we'll have our server listen to in the config file (config.js, in the project root):

**TAGGING**

git tag -a v1.4 -m "my version 1.4" Create an annotated tag

git tag v1.4-lw Lightweight Tag - This is basically the commit checksum stored in a file, no other information is kept.

git show v1.4

Listing Your Tags

git tag

git tag -l "v1.8.5\*"

git log --pretty=oneline

15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'

a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support

0d52aaab4479697da7686c15f77a3d64d9165190 one more thing

6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'

0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc added a commit function

4682c3261057305bdd616e23b64b0857d832627b added a todo file

166ae0c4d3f420721acbb115cc33848dfcc2121a started write support

9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile

964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo

8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme

Now, suppose you forgot to tag the project at v1.2, which was at the “updated rakefile” commit. You can add it after the fact.

To tag that commit, you specify the commit checksum (or part of it) at the end of the command:

git tag -a v\_0.1 807c73f5fdfe5c9657f7c2b23c2677edf71d938c -m "Version without JWT Auth"

**Sharing Tags** - By default, the git push command doesn’t transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them. This process is just like sharing remote branches — you can run git push origin <tagname>.

git push origin v1.5

If you have a lot of tags that you want to push up at once, you can also use the --tags option to the git push command. This will transfer all of your tags to the remote server that are not already there.

git push origin --tags

Checking out Tags - If you want to view the versions of files a tag is pointing to, you can do a git checkout, though this puts your repository in “detached HEAD” state, which has some ill side effects:

git checkout 2.0.0

In “detached HEAD” state, if you make changes and then create a commit, the tag will stay the same, but your new commit won’t belong to any branch and will be unreachable, except by the exact commit hash. Thus, if you need to make changes — say you’re fixing a bug on an older version, for instance — you will generally want to create a branch:

git checkout -b version2 v2.0.0

If you do this and make a commit, your version2 branch will be slightly different than your v2.0.0 tag since it will move forward with your new changes, so do be careful.

**Deploying Applications: DOCKER**

Docker unlocks the potential of your organization by giving developers and IT the freedom to build, manage and secure business-critical applications without the fear of technology or infrastructure lock-in.

By combining its industry-leading [container engine](https://www.docker.com/products/docker-engine) technology, an[enterprise-grade container platform](https://www.docker.com/products/docker-enterprise) and [world-class services](https://success.docker.com/support), Docker enables you to bring traditional and cloud native applications built on [Windows Server](https://www.docker.com/products/windows-containers), Linux and mainframe into an automated and secure supply chain, advancing dev to ops collaboration and reducing time to value.

Because Docker increases productivity and reduces the time it takes to bring applications to market, you now have the resources needed to invest in key digitization projects that cut across the entire value chain, such as [application modernization](https://www.docker.com/solutions/mta), [cloud migration](https://www.docker.com/solutions/cloud-migration) and server consolidation. With Docker, you have the solution that helps you manage the diverse applications, clouds and infrastructure you have today while providing your business a path forward to future applications.

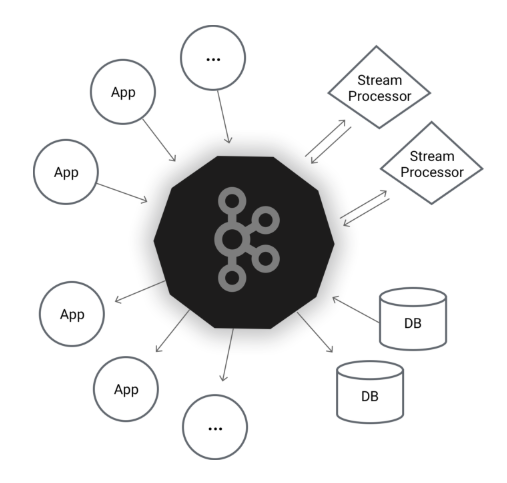
<https://docs.docker.com/>

<https://scotch.io/tutorials/getting-started-with-docker>

<https://scotch.io/tutorials/create-a-mean-app-with-angular-2-and-docker-compose#toc-dockerize-the-express-server-api>

## APACHE KAFKA

Kafka® is used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.



<https://kafka.apache.org/intro>

TODO

Integración con Apache Cordova - https://onsen.io/

Apache kafka