MEAN technology pros and cons:

MEAN pros;

* Fast, very very fast for non blocking interactions. A REST API response with a 20k package in MEAN is on average between 20ms - 40ms, compared to a LAMP response, on the same server with the same 20k package, of about 200ms - 400ms (both without caching).
* Same language throughout the entire stack.
* MEAN stack encourages a distributed architecture which, when properly structured, vastly reduces backend hardware requirements at scale.
* Loose typing or strongly typed you decide (and with multiple typing flavors).
* You get to work with Angular, Node, and Mongo together, imo the very best of their respective classes. (React is not a front-end framework it is only a rendering library)

MEAN cons;

* Node is not good for blocking, or heavy computational, back-end logic.
* Node is single threaded.
* The stack is a very different paradigm for backend developers coming from existing stacks and requires a different architectural approach to fully take advantage of its performance benefits.

LAMP stack pros and cons:

| Non-proprietary and open-source |  |
| --- | --- |
| Long history of support | Limited scalability makes it unsuitable for large applications |
| Components of this stack can be swapped out as needed, making it highly flexible | MySQL is relatively easy to learn but isn’t the best option for high-performance applications |
| Easy to use and set up, especially for beginners | Shared servers can cause performance issues |
| [MySQL](https://www.educative.io/blog/mysql-tutorial) and [PHP](https://www.educative.io/blog/php-tutorial-from-scratch) are widely supported across many web hosts |  |
| Cost-effective |  |

MERN technology cons and pros:

#### **Pros**

* Rendering and efficiency of the user interface
* Budget-friendly
* Free
* Quickly move between server and client mode

#### **Cons**

* Inappropriate for critical systems
* Reduced productivity

Angular

Pros:

**Reusability**. Components of similar nature are well encapsulated, in other words, self-sufficient. Developers can reuse them across different parts of an application. This is particularly useful in enterprise-scope projects where different systems may have many similar elements like search boxes, date pickers, sorting lists, etc.

**Readability**. Encapsulation also ensures that new developers – who’ve been recently onboarded to a project – can read code better and eventually reach their plateau of productivity faster.

**Unit-test friendly**. The independent nature of components simplifies unit tests, and [quality assurance](https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/) procedures aimed at verifying the performance of the smallest parts of the application, units.

**Maintainability**. Components that are easily decoupled from each other can be easily replaced with better implementations. Your engineering team will be more efficient in maintaining and updating the code within the iterative development workflow.

* Angular offers faster load time and increased security by using a brilliant concept commonly known as “ahead-of-time compiler." Angular compiles HTML and TypeScript into JavaScript during the development, which means all of the code is compiled before the browser even loads your web app.
* With Angular elements and modules, this framework is designed to be fully customizable, giving more power to the developer and designer. Angular elements can also easily be added to projects that were built using another framework, which just adds to the appeal of this program.
* The dependency injection capabilities of the Angular framework can be a huge advantage—if you know how to use it. Creating the various dependencies can be time-consuming, but well worth it if you’re working on a large-scale project that is expected to perform a number of different functions.

Cons:

### Angular is verbose and complex

The most frequent complaint that you hear from the Angular development community is the verbosity of the instrument. This problem hasn’t changed since AngularJS.

Although we’ve mentioned the component-based architecture as the major Angular benefit, the way components are managed is too complicated. For instance, you may need up to five files for a single component in Angular, have to inject dependencies, and declare the component lifecycle interfaces. Other points of concern are Angular-specific 3rd party libraries and Angular syntax. Much of the development time in Angular is spent doing repetitive things.

### Θ Hard learning curve

If you onboard new developers familiar with JavaScript to learn and use new Angular, they will be challenged compared to similar React or Vue onboarding. The array of topics and aspects to be covered is large: modules, dependency injection that we mentioned before, components, services, templates, etc.

Another barrier is RxJS, a reactive programming library for asynchronous programming. Learning it, at least on the basic level, is mandatory for using Angular. Engineers complain about error messages that are too cryptic to grasp without additional research followed by trial-and-error manipulations.

### **The Complexity of the Framework**

Angular is complex to understand despite the astounding features that make it an excellent tool for web development. Using its features calls for specialized support from experts who have mastered the framework and understand it inside-out.

* One of the major drawbacks to using Angular is the platform’s limited SEO options and poor accessibility for search engine crawlers. Being that Google is the number one search engine on the planet, we can’t imagine this problem will continue to persist for much longer before a solution is presented in another update.
* Another glaring con of using the Angular framework is that it can be quite difficult to learn. With such a complex web of modules, coding languages, integrations and customizing capabilities, understanding Angular definitely takes some time. Fortunately, Angular provides phenomenal support and there are a number of [online tutorials](https://www.pluralsight.com/blog/tutorials/angularjs-step-by-step-services) and courses you can take to catch up and start taking full advantage of all the program has to offer.
* Unless you’re working on a project of appropriate size and complexity, Angular can easily weigh you down. Save this one for the big-ticket assignments and you’ll fare just fine.

Python + Django

Pros:

[Django](https://www.netguru.com/services/django-development) helps developers by speeding up the development process. It includes its own Object Relation Mapping (ORM) layer for handling database access, sessions, routing, and multi-language support. It also takes care of security while handling requests. It includes an admin panel (called django-admin) for managing models data by default.

### **Security**

Django includes prevention of common attacks like Cross-site request forgery (CSRF) and SQL Injections. More technical details can be found in [the official security overview guide](https://docs.djangoproject.com/en/2.0/topics/security/).

### **Python inside**

Everything is explicit, following the mentioned Zen of Python, where explicit is always better than implicit. In the case of Django it means: you have to configure it first. It’s the opposite of convention over configuration. It’s the perfect framework for [developers used to Python](https://www.netguru.com/services/python-development). **Django inherits all Python’s benefits, like great support for external libraries and programmer productivity boost**. It significantly speeds up development.

Cons:

### **Speed**

Badly-designed architecture combined with Python, which is not the fastest language around, may lead to slow websites. So **make sure that your app is properly optimized**. Django offers its own benchmarks to check the speed of internals and spot all bottlenecks. Caching and a bunch of different optimizations can be applied. Check out [a complete guide](https://docs.djangoproject.com/en/2.0/topics/performance/).

Ensuring a well-optimized and scalable architecture from the beginning should save you a lot of problems with speed in the future. Django powers huge web applications - experiencing speed issues is not a problem of Django itself, but rather a question of proper configuration and architecture design.

### **Lack of convention**

In comparison to frameworks like Ruby on Rails (which is a perfect example of the Convention Over Configuration approach), everything has to be explicitly defined, **which leads to configuration boilerplate that may slow down the development process.** On the other hand, relying on configuration is a common practice for the Python ecosystem.

#### Not good for simpler projects

Django is a high-level framework, and so it lends itself more to complicated features. Handily, if you are looking for the right tool for smaller, easier projects that have potential to grow rapidly, then the micro-framework Flask is the one for web developers. If you’re interested in learning more about it, we’ve written a full beginner’s guide to the Flask microframework.

#### Can lead to slow websites

For all the speed at which you can build things with the Django framework, sometimes it can result in your website running quite slowly. It’s important to note that this isn’t due to anything particular to the Python language or Django itself, more to the amount of resources you’re accessing with the framework. The more databases and so on that you’re sending requests to, the slower the performance will become.

All this means is that while it’s fun to be able to scale quickly and change direction at the drop of a hat, it’s important to keep one eye on the overall architecture of the project. It could result in speed problems further down the line. That said, this can only start becoming apparent at a truly large scale for your app, and the wealth of Django documentation and thriving community will provide you with the resources to solve these performance issues.

#### You need to know everything in order to work it

A problem with this can be that Django is too vast at times, and requires you to know the plan of the whole structure before you can actually start building it. That said, the wide range of support resources and community can make this easier, as there is guidance and documentation to follow.

#### Lack of conventions

A coding convention is a group of guiding principles to follow when using a web framework—a rulebook of sorts. Whereas Rails has “Convention over Configuration”, Django doesn’t use any conventions, which can put some programmers off, and lead to slower progress at times.

MySQL

Pros:

* **Faster in processing queries:** [SQL allows](https://cloudinfrastructureservices.co.uk/postgresql-vs-sql-server/) to retrieve a large amount of data within seconds, making the process quick and efficient. Data Insertion, manipulation, and deletion are just a matter of seconds with SQL.
* **Multiple Data Views**: SQL let to create multiple views of the database structure, customizing the view for users and their use cases.
* **No Coding:** SQL takes away the burden of writing bulky codes to perform operations. There is a set of keywords that can be used in SQL such as SELECT, INSERT INTO, DELETE, CREATE, DROP, and UPDATE. Therefore SQL is considered a user friendly language.
* **Standardized Language:** SQL has been developed and used over the years. Therefore it is very popular within the community and features rich documentation and extensive community support. Furthermore, SQL is a query language recognized by ISO and ANSI.
* **Language Intractability**: The SQL language is highly interactive. It is also easy to learn and understand. Therefore, it will only take seconds to communicate with databases and receive feedback on even complex queries.
* **Portability:** SQL can be used on many devices like PCs, servers, and laptops. It is also independent of the platform, such as the Operating System. Moreover, SQL can be embedded with other programs or applications depending on the requirement. In simple words, it is a “code once, runs forever” kind of language.

Cons:

* **Lack of Control:** SQL is designed to write queries with pre-defined code snippets/keywords. So, developers may face some problems when developing queries with SQL as they may not have full access to the database. Complete access to databases is forbidden for users due to some hidden business rules.
* **Complexity in understanding:** New users/ developers may find it difficult to understand some specific queries.
* **Complex Interface:** Users might feel uncomfortable when using the interface as it is a bit complex to find operations and manage databases.
* **Cost:** Some versions and features are not free and can be costly.
* MySQL is not very efficient in handling very large databases.
* MySQL doesn’t have as good a developing and debugging tool as compared to paid databases.
* MySQL versions less than 5.0 do not support COMMIT, stored procedure and ROLE.
* MySQL is prone to data corruption as it inefficient in handling transactions.
* MySQL does not support SQL check constraints.

**Explanation for each point on the slide**

**Pros For Angular:**

* Offers great reusability=self-sufficiency.

Explanation: Developers can reuse them across different parts of an application. This is particularly useful in enterprise-scope projects where different systems may have many similar elements like search boxes, date pickers, sorting lists, etc

* Maintainability.

Explanation: Components that are easily decoupled from each other can be easily replaced with better implementations

* Customizability.

Explanation:this framework is designed to be fully customizable, giving more power to the developer and designer. Angular elements can also easily be added to projects that were built using another framework, which just adds to the appeal of this program

* Two-way data binding is when data automatically reflates once the value is changed.

Explanation:

**Cons For Angular:**

* The language is verbose and complex.

Explanation: The most frequent complaint that you hear from the Angular development community is the verbosity of the instrument. This problem hasn’t changed since AngularJS.

Although we’ve mentioned component-based architecture as the major Angular benefit, the way components are managed is too complicated.

* Learning curve issues due to AngularJS.

Explanation:

If you onboard new developers familiar with JavaScript to learn and use new Angular, they will be challenged compared to similar React or Vue onboarding. The array of topics and aspects to be covered is large: modules, dependency injection that we mentioned before, components, services, templates, etc.

Another barrier is RxJS, a reactive programming library for asynchronous programming. Learning it, at least on the basic level, is mandatory for using Angular.

* Limited SEO(search engine optimization) capabilities.

Explanation:

Angular is the platform’s limited SEO options and poor accessibility for search engine crawlers. Being that Google is the number one search engine on the planet, we can’t imagine this problem will continue to persist for much longer before a solution is presented in another update.

**Pros For Python+Django:**

* Usability with integrating Python.

Explanation:

**Django inherits all Python’s benefits, like great support for external libraries and programmer productivity boost**. It significantly speeds up development.

* Improved Security.

Explanation:

Django includes prevention of common attacks like Cross-site request forgery (CSRF) and SQL Injections.

* Time-and-cost effective.

Explanation:

**Cons For Python+Django:**

* Degrading performance issues.

Explanation: Badly-designed architecture combined with Python, which is not the fastest language around, may lead to slow websites. So **make sure that your app is properly optimized**.

* Can’t handle multiple requests.

Explanation:

* Relying on the ORM system.

Explanation:

Django ORM, created before SQLAlchemy existed, is now much inferior to SQLAlchemy. It is based on the Active Record pattern which is worse than the Unit of Work pattern adopted by SQLAlchemy. This means, in Django, models can “save” themselves and transactions are off by default, they are an afterthought.

* Lack of Convention

Explanation: In comparison to frameworks like Ruby on Rails (which is a perfect example of the Convention Over Configuration approach), everything has to be explicitly defined, **which leads to configuration boilerplate that may slow down the development process.** On the other hand, relying on configuration is a common practice for the Python ecosystem.

**Pros For MySQL:**

* Portability among vast amounts of devices.

Explanation:

SQL can be used on many devices like PCs, servers, and laptops. It is also independent of the platform, such as the Operating System. Moreover, SQL can be embedded with other programs or applications depending on the requirement. In simple words, it is a “code once, runs forever” kind of language.

* Language Intractability.

Explanation: The SQL language is highly interactive. It is also easy to learn and understand. Therefore, it will only take seconds to communicate with databases and receive feedback on even complex queries.

* Simple operational coding.

Explanation:SQL takes away the burden of writing bulky codes to perform operations. There is a set of keywords that can be used in SQL such as SELECT, INSERT INTO, DELETE, CREATE, DROP, and UPDATE. Therefore SQL is considered a user friendly language.

**Cons For MySQL:**

* Lack of Control.

Explanation:SQL is designed to write queries with pre-defined code snippets/keywords. So, developers may face some problems when developing queries with SQL as they may not have full access to the database. Complete access to databases is forbidden for users due to some hidden business rules.

* Degrade of efficiency at large scale.

Explanation:

* Chance of Data corruption.

Explanation:

* Experiences of lack of developing and debugging tools.

Explanation:

**For the login screen, he doesn't want to see all the tabs.**

**Automatically the time will be updated and given start then will fit up until finished. Specify the time from beginning to end.**

**I Want a reference of practice plan drills on the side while creating the practice plan. Open to drop down menu of all the options/search bar pop up to easy find the drill**

**HE SUPPORTS the MERN stack and will choose the Bluehost web hosting server.**

**Database costs could occur.**

**We need to do research into hosting the database where we can store the data to be accessed.**