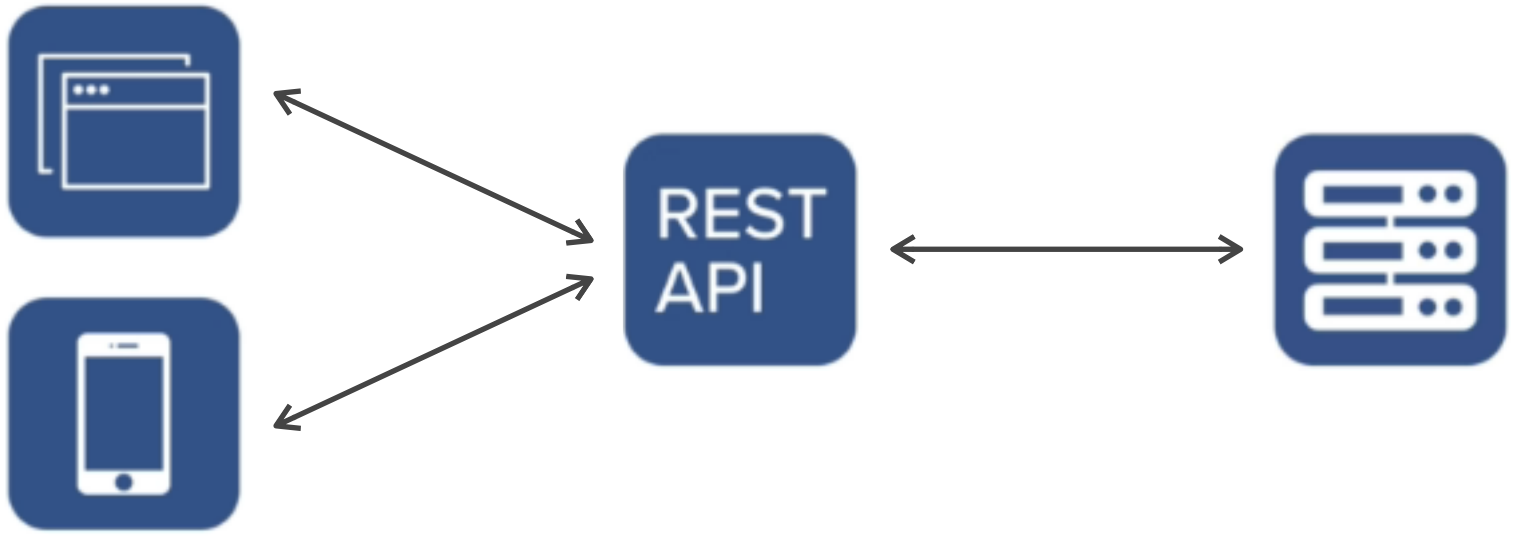
GraphQL with Node.js and Apollo

Section 1: Introduction

Why GraphQL & Installing Node.js and Visual Studio Code

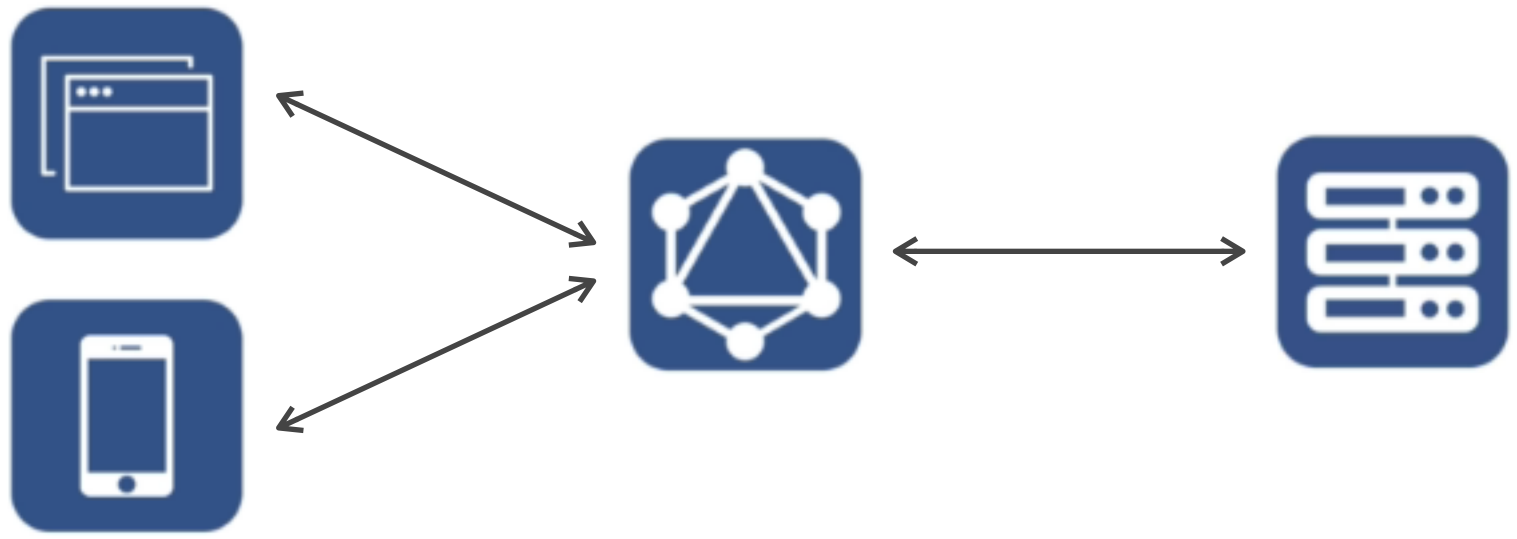
Most current applications likely uses some form of a REST API where we are making HTTP requests between the client and the server to send and receive data. Below is a visual representation of a REST API application where we have a client and a backend database/server.



The REST API would have multiple different endpoints/URLs that can be used to communicate with the server. For example: one endpoint for signing up, another for logging in, creating new data, reading all data and so on.

When we introduce GraphQL not a lot will change. We will continue to be able to use any client and any server but we will replace the REST API (with many endpoints) with a GraphQL API that has a single endpoint exposed.

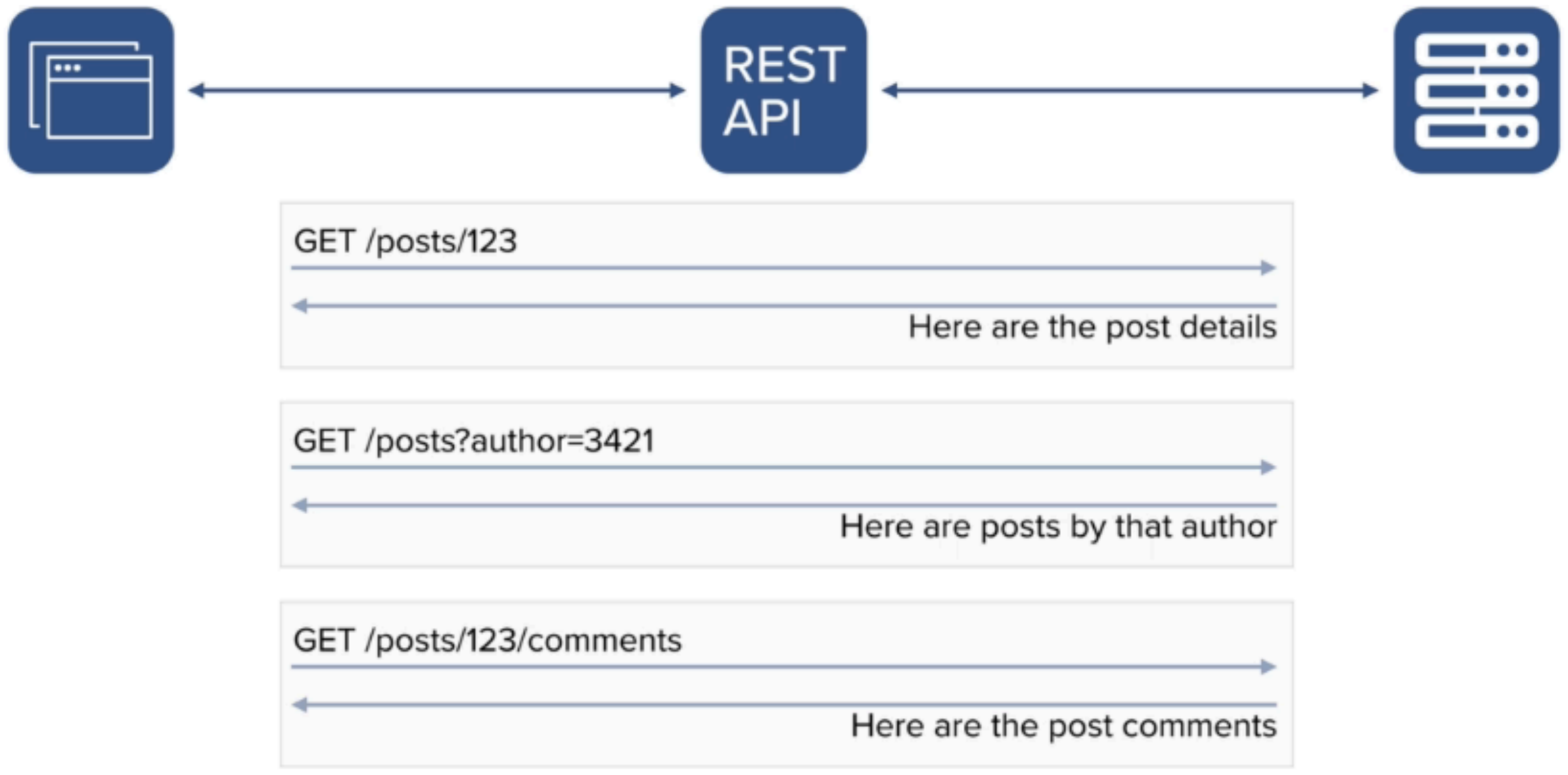
GraphQL stands for the Graph Query Language and is something that operates over HTTP and therefore we can use any backend language and any database we want as well as any client (web, mobile, server) we want.



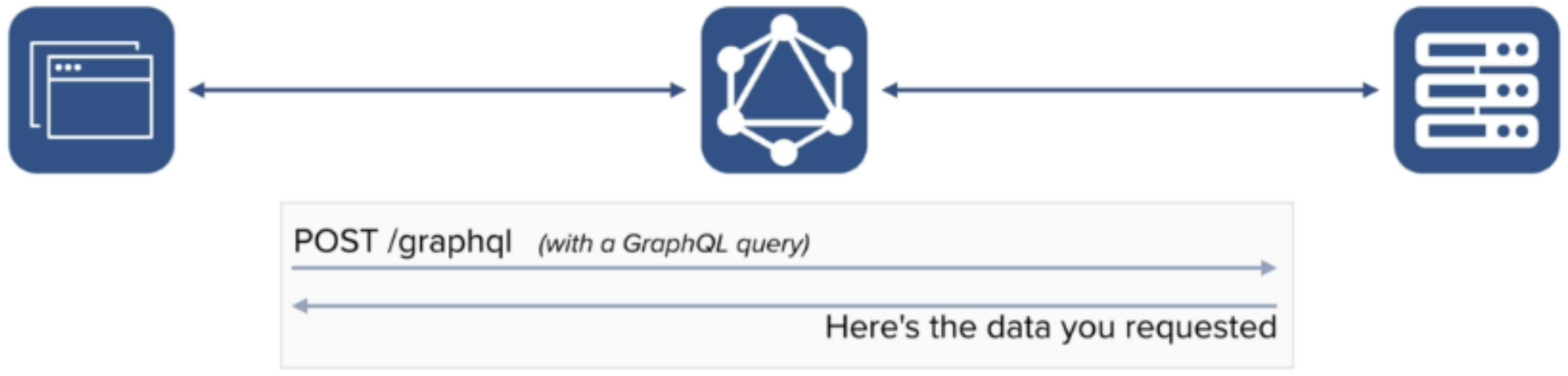
Why use GraphQL? GraphQL is fast, flexible, easy to use and simple to maintain. Below is a visual representation of a typical example of loading the necessary data for a page with a REST API and then with a GraphQL API.

We have a client and a server but the glue between the two is either a REST API or a GraphQL API. In both examples we would make a HTTP request to fetch the data necessary to render a page which displays a bog post i.e. the post details, comments made on the post and other blog posts made by that author.

Starting with the REST API we have a dozen or so endpoints for managing our application data. We would make HTTP requests to the various endpoints to get all the necessary data from the server. The server would determine what data to retrieve and send back to the client based on the endpoint requests made to the server.


REST API
GET /posts/123
Here are the post details
GET /posts?author=3421
Here are posts by that author
GET /posts/123/comments
Here are the post comments

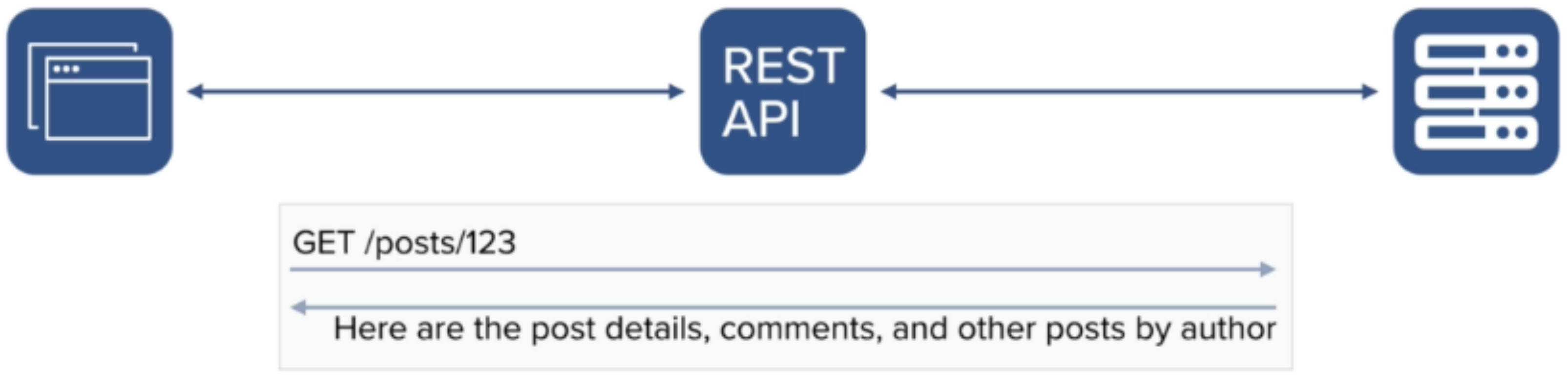
The GraphQL API exposes a single endpoint which is a very important piece to the puzzle. This endpoint could be called anything we like (in the example below it is named /graphql). When making a HTTP request to this single endpoint we would also send along a GraphQL query.


POST /graphql (with a GraphQL query)
Here's the data you requested

A GraphQL query allows the client to describe exactly what data it needs from a server, the server gets all the data ready to send back to the client. This is the second most powerful piece to the puzzle. Instead of the server determining what data gets sent back, it is up to the client to request all of the data it needs. Therefore in the GraphQL case, it can request the post details, the post comments and other posts by that author all within a single GraphQL request.

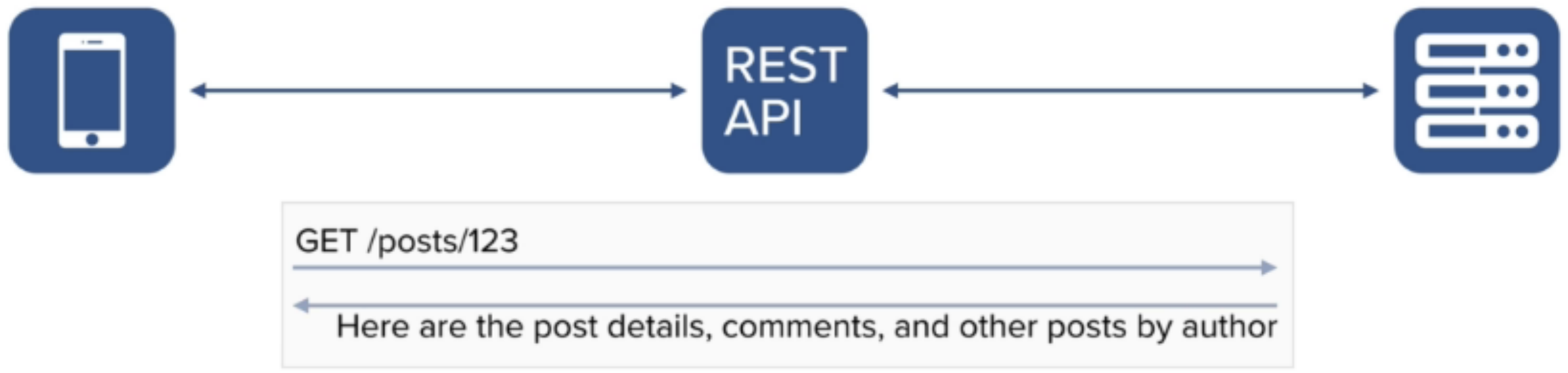
The important difference between a REST API and a GraphQL API is that the latter allows the client to determine what data it gets back as apposed to the former HTTP endpoint which allows the server to determine what data comes back from and endpoint. Clearly 3 HTTP requests is more than a single HTTP request and therefore GraphQL is going to be more faster.

The greatest advantage of GraphQL is the flexibility. We could argue that for the REST API we could have had one endpoint that would return everything.

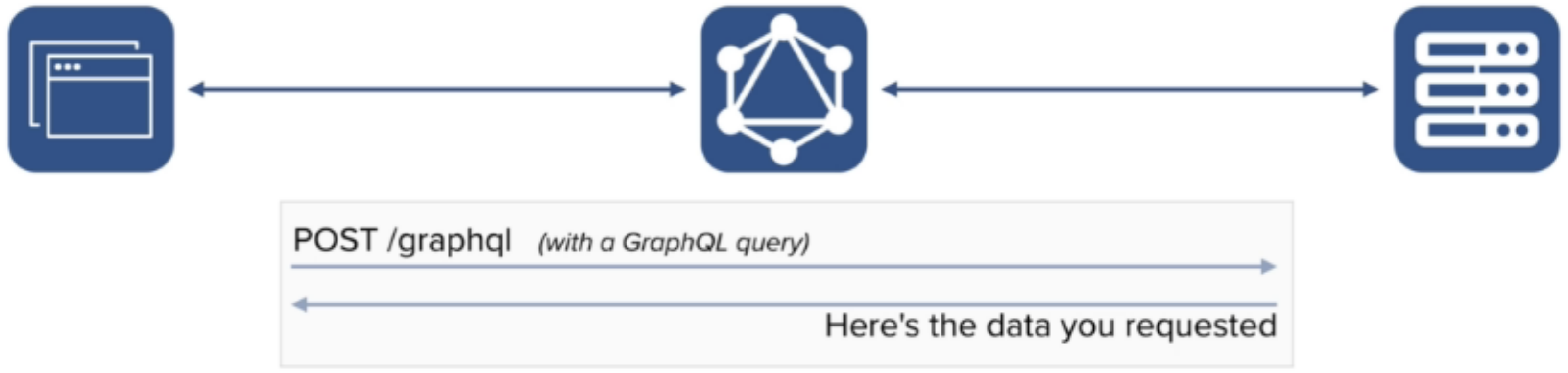

REST API
GET /posts/123
Here are the post details, comments, and other posts by author

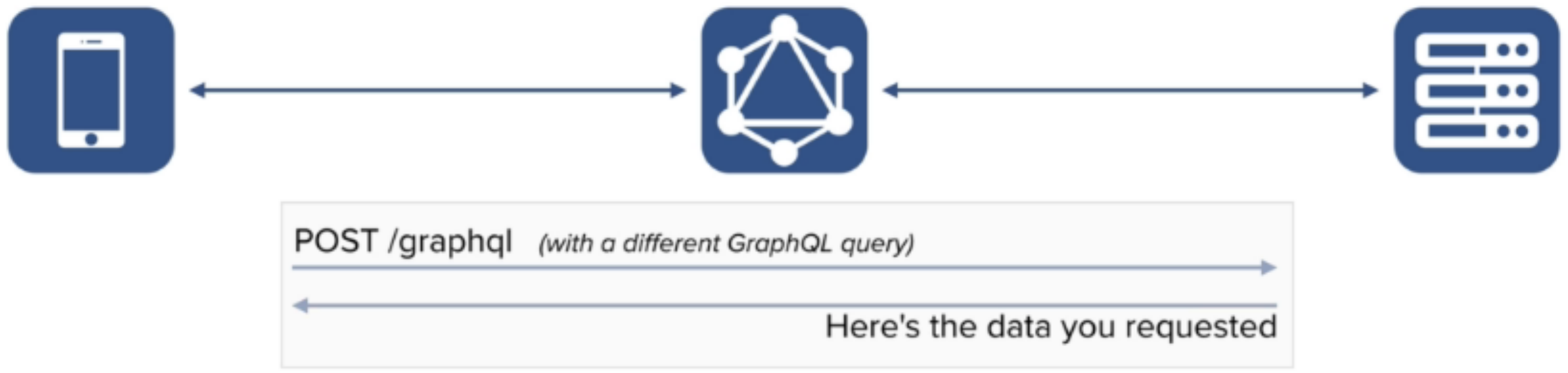
The above would be a perfectly fine approach and it would give the client everything it needs to render that page with just a single HTTP request. However, the problem with this solution is that we now have this one endpoint which is making way more database request than it was before.

It is getting big and slow i.e. from one database request to three database request in order to get all of the data necessary. For the desktop version of the application this may be perfectly fine as we may use the data right away. However, if we have a mobile version of our application to use the same backend. The problem with this is that the mobile application cannot change the data it gets back.


REST
API
GET /posts/123
Here are the post details, comments, and other posts by author

On mobile devices we have a whole set of considerations to take into account such as screen real estate, battery life, hardware and slow/expensive data. We want to make sure that we do not abuse the device else the users are going to get a poor user experience and unlikely to use the mobile application. This was the original reason for why GraphQL was created i.e. a flexible way for the individual clients to request exactly the data that it requires to use i.e. nothing more and nothing less. This is not an issue we would run into with the GraphQL API because it is more flexible.


POST /graphql (with a GraphQL query)
Here's the data you requested


POST /graphql (with a different GraphQL query)
Here's the data you requested

The GraphQL API exposes a single endpoint for both desktop and mobile applications; however, the GraphQL query allows us to specify what data we would like to get back for example the desktop application would want everything while the mobile application would only want the post details.

The desktop query would require the server to do a lot more work i.e. make three database requests to get all of the data. However, for mobile the server does less work as the queery requests less data i.e. one database request to get only the post details.

The REST API does not provide the same flexibility as we get the same response with both clients. With GraphQL it is for the individual client that determines what data it gets back from the server while with a REST API it is the server that determines what data it gets back to what endpoints.

Finally, with a REST API if the client requires different data this typically requires us to add a new endpoint or change an existing one. Using a GraphQL API, the client would just need to change its query making GraphQL API's much simpler to maintain compared to REST API.

To conclude, GraphQL creates fast and flexible APIs, giving clients complete control to ask for just the data they need. This results in fewer HTTP requests, flexible data querying and in general less code to manage.

Before we can dive into looking at GraphQL we would need to install some applications/packages on our machines.

1) Node.js

We can visit https://nodejs.org/en/ homepage to download node.js for our operating system. There are two versions: The LTS version (Long Term Support) and the Current release version. It is advisable to download the Current release, as this will contain all the latest features of Node.js. As at January 2020, the latest version is 13.6.0 Current.

We can test by running a simple terminal command on our machine to see if node.js was actually installed correctly:

```
:~$ node -v
```

If node.js was installed on the machine correctly the command would return the version of node that is running on the machine. If "command not found" is returned, this means node was not installed on the machine.

2) Visual Studio Code

A text editor is required so that we can start writing out our node.js scripts. There are a couple of open source code editors (i.e. free) that we can use such as Atom, Sublime, notepad++ and Visual Studio Code to name a few.

It is highly recommended to download and install Visual Studio Code as your code editor of choice as it contains many great features, themes and extensions to really customise the editor to fit your needs. We can also debug our node.js scripts inside of the editor which is also a great feature.

Links:

https://code.visualstudio.com/

https://atom.io

https://www.sublimetext.com/

Some useful extensions to install in Visual Studio Code are:

Babel ES6/7 (dzannotti), Beautify (HookyQR), Docker (Microsoft), Duplicate action (mrminc), GraphQL for VSCode (Kumar Harsh), npm (egamma), and npm Intellisense (Christian Kohler).