Contents

[Proposed Solution 2](#_Toc57819071)

[1. API (Laravel) 2](#_Toc57819072)

[1.1 Setup 2](#_Toc57819073)

[1.2 Models, Controllers and Routes 3](#_Toc57819074)

[1.3 Results 5](#_Toc57819075)

[2. Frontend App (React) 6](#_Toc57819076)

[Components 6](#_Toc57819077)

[App.js 6](#_Toc57819078)

[PropertyGrid.js 7](#_Toc57819079)

[PropertyItem.js 8](#_Toc57819080)

[Search.js 8](#_Toc57819081)

[Header.js 9](#_Toc57819082)

[Styling (CSS) 10](#_Toc57819083)

[App.css 10](#_Toc57819084)

[The Result 10](#_Toc57819085)

# Proposed Solution

We were asked to design and implement a search function for the properties focusing on the bookings made for them. In the following sections, we will describe our solution and how we reach to conclusions based on the results

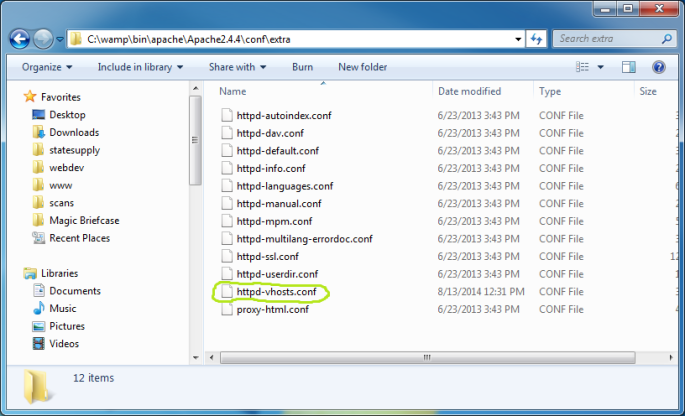
# API (Laravel)

We were given a very simple database to work with but we wanted to bring everything to life by using the latest web technologies. For this, we took everything and we wrote a simple API using Laravel to serve the data which will be used to assist our Frontend application.

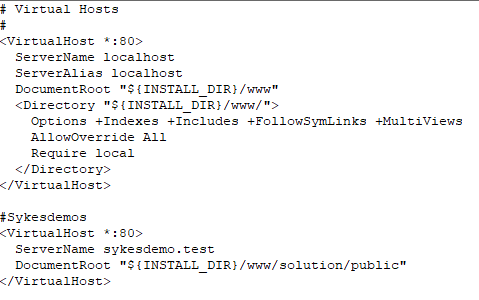
One of the most popular ways of sharing data between web applications is APIs which can be used over nearly any protocol, when used for web APIs it typically takes advantage of HTTP.

## 1.1 Setup

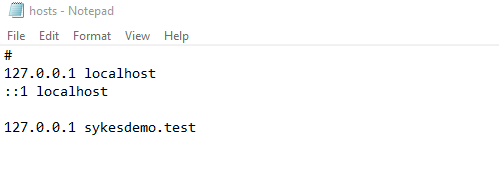
Since we will be working on a local server, we chose to use the **WAMP** suite which offers Apache Web server. Then, we had to setup our Virtual Host Configuration File as well as our hosts file.

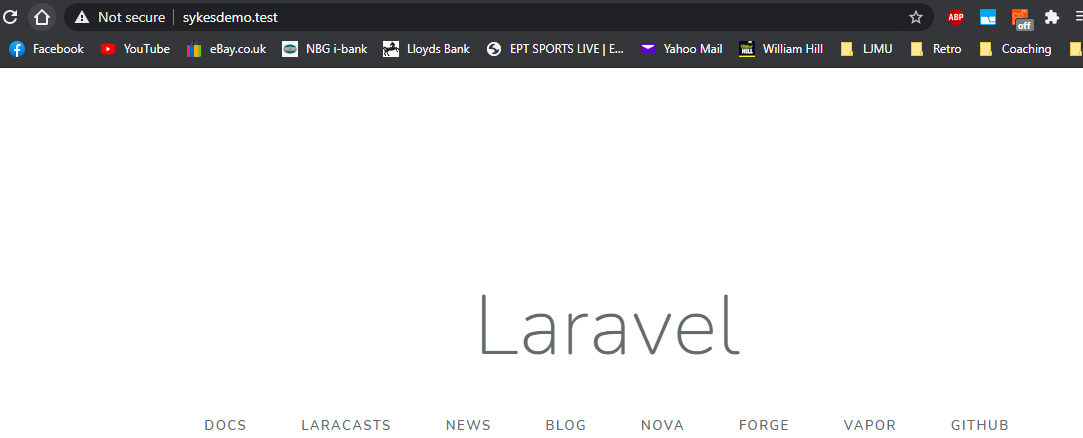


We added a virtual server named **Sykesdemo.test**.



Finally, we had to update our Windows hosts file located in C:\Windows\System32\drivers\etc\hosts file.



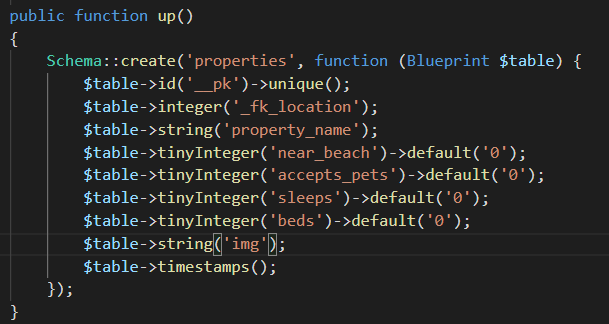


## 1.2 Models, Controllers and Routes

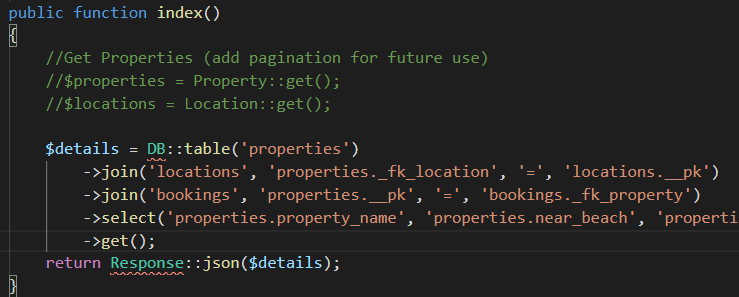
We started off by creating 3 Models named **Property, Booking, Location** and their respective controllers. Furthermore, we added the correct routes to the **api.php**. These routes will guide the data to be displayed correctly.



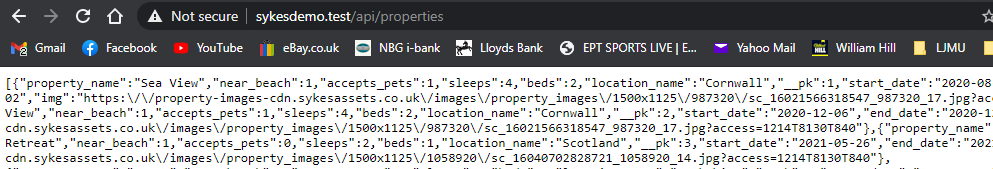
We also recreated the database using migrations.



In the index method of the **PropertyController** we run a join query which will fetch the data from the Properties, Bookings and Locations based on which property appears on which location.

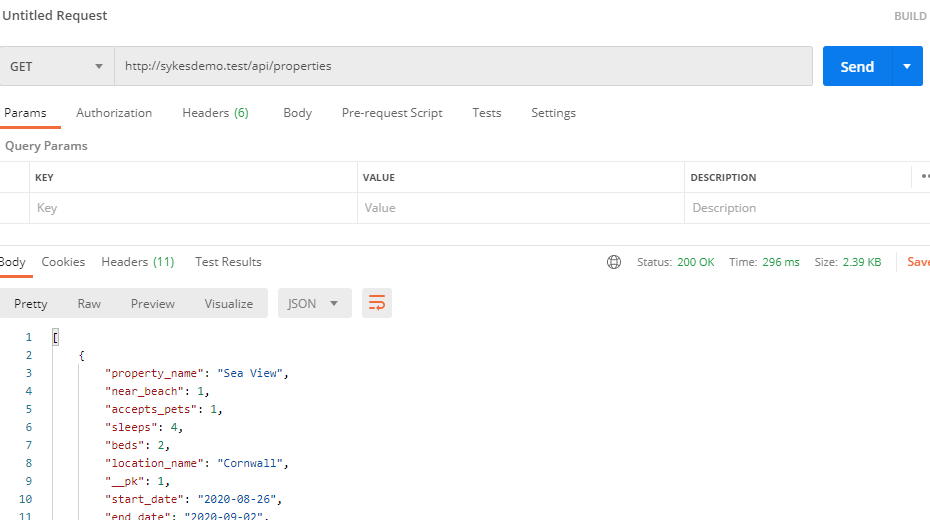


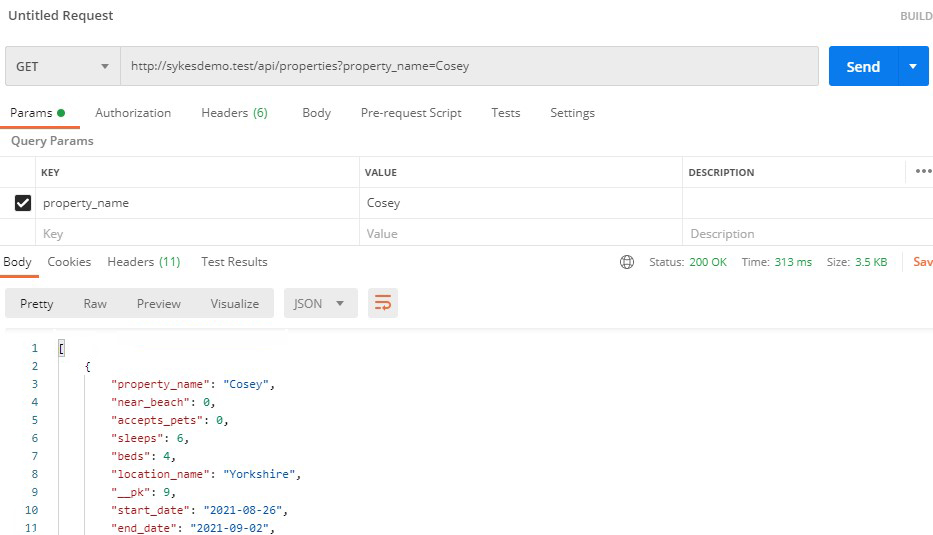
Finally, we used our web browser to test that the data is being returned correctly as a **JSON** object.



## 1.3 Results

To test the functionality of our API we used **Postman** to make requests to our server which returned the results successfully.





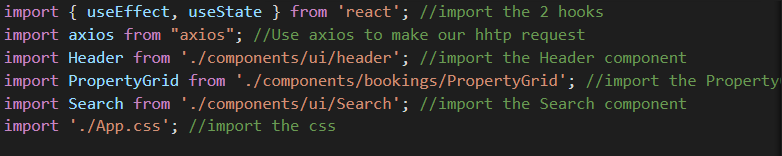
# Frontend App (React)

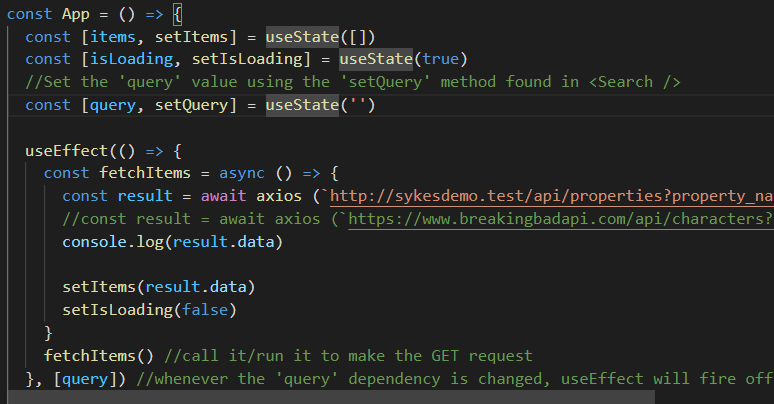
We wrote an application to supply us with the data of the properties, bookings and locations so now it’s time to create a frontend application to make of that data. We used React to create an application which would connect to the API and get all the data with a view of serving it to the customers. In our application we use a couple of components to achieve having a dynamic search function with which the user can search through the bookings. We will be using **Axios** HTTP library to make http requests and fetch the data we need.

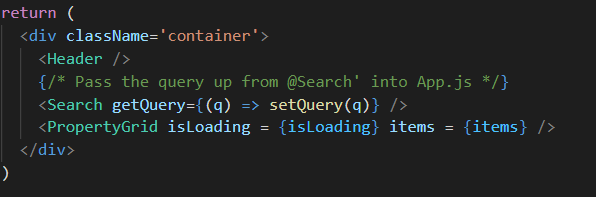
## Components

In this section, we explain our components. We separated the property stuff from the ui by creating the appropriate folders to hold our components.

### App.js

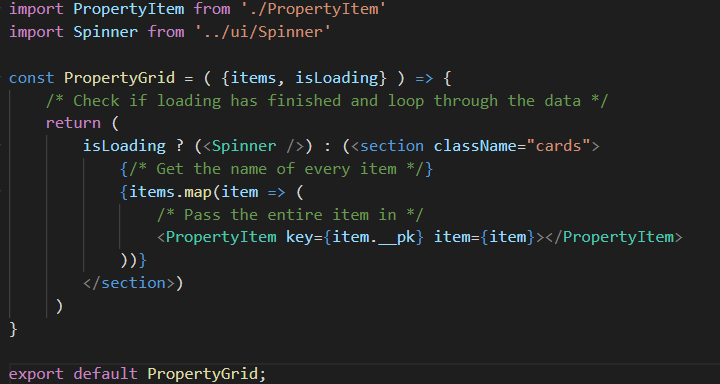
The App.js is the main component where everything happens. We import the components, hooks as well as Axios which will help us interact with our API. Moreover, we declare that we will be using the **useEffect** which will enable us to use states within our functions and **useState** hooks which will allow us to fire off when our component loads to make an HTTP request to get the data.  


In the App component we use the **useState** hook defining the items which will represent the bookings that come from the API and the **setItems** which will be the function we use to change the state whenever the user types something. The **isLoading** state is used to let us now if the loading of the data from the API was completed and successful. The **useEffect** hook uses **Axios** to make the HTTP requests to the API and in the url we add a **property\_name** parameter in order to help us search for a specific booking made for a specific property.  


This is the wrapping div where we display things on the frontpage.  


### PropertyGrid.js

The PropertyGrid.js serves… We pass the **isLoading** and **items** state as props and which catch them in the **PropertyGrid**. Then, we check is the data has finished loading and upon confirmation we use the map() method which is high order array method to loop through the results. Finally, we bring in the **PropertyItem** component and we pass the entire item in. We also had to set a unique key as React will through a warning if it finds items that have the same id.



### PropertyItem.js

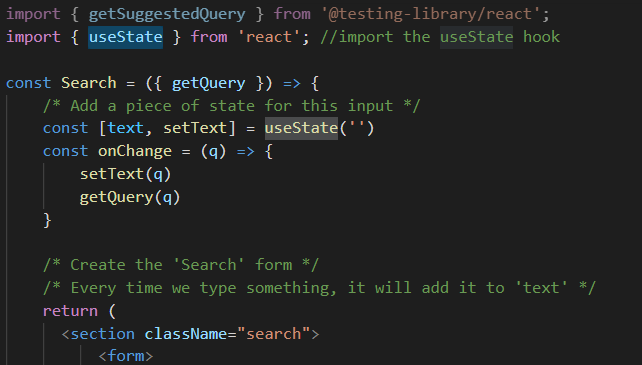
The PropertyItem.js holds all the functionality and the styling of the data that resulted from the API and we use it to render every item we got from the **PropertyGrid.js** component. We use divs and styling to help us with the look we want to achieve.



### Search.js

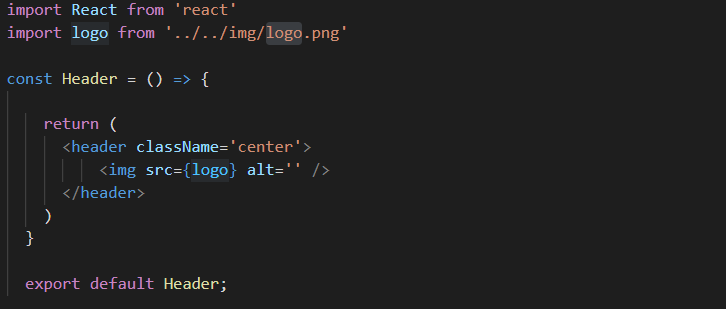
The Search.js component handles the dynamic search which includes everything from the moment the user starts typing the first letter of a property to the time that the search has finished and the results must be displayed on the screen. We bring in the **useState** hook as we will be working with a form and in a form every input is going to have its own piece of state. We take the value of the input text and we use the **onChange** event which will be triggered every time the user types a new character in the search box and we add an arrow function to catch the changes and pass them to the **text** state.

The next thing is to get whatever the user has typed to the main App.js component, pass it as a **property\_name** into our API query and filter out the properties by name. To achieve that we created an **onChange** function which will the take the value name **q** and call **setText** and set it to that value. Furthermore, we create a **getQuery** function which will take the query q and also set the **setQuery** to it to pass the query from our search to the main DOM.



### Header.js

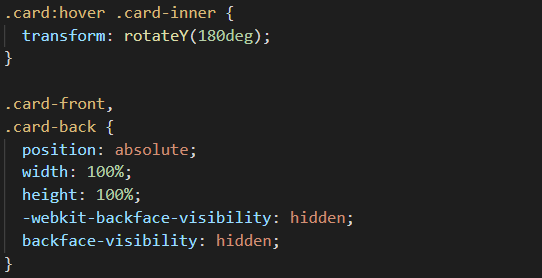
The Header.js components’ functionality is very simple as it registers the logo image we will be using and then displays it, adding little bits to the UI.



## Styling (CSS)

### App.css

This file handles all the css styling. It is made of pretty simple rules that define the colors, background images and positions as well as the card flip effect.



## The Result

Our application is successfully retrieving the data from the API and displays it on the screen. Furthermore, it will return a specific booking made for a specific property.

