**Build and run instructions for barometer**

**Summary**

This barometer is based on the PHP framework Laravel and Valinlla JavaScript. Air pressure and weather data is retrieved from the OpenWeather API service (https://openweathermap.org/) using the Guzzle HTTP client library. Extracted and calculated necessary variables stored in Maria DB database. Data from the database via Controller to Blade is provided when the client loads the page. Data is updated every minute using a Laravel Scheduler. When database update occurs, Laravel Event is activated to send weather and pressure data to Laravel Echo (a JavaScript library) via websockets using the open source Pusher Channels driver.  Laravel Echo receive broadcasted data on the client-side and reloads received values of the variables.

**Preparation**

1. Coposer was downloaded and installed from here

**https://getcomposer.org/download/**

1. Laravel project created via command line:

**composer create-project laravel/laravel barometer-app**

(*go in to this folder using command line:*

***cd barometer-app*)**

3) Then the model for a city's barometric air pressure was created using the command line:

**php artisan make:model Barometer -mc**

This command line for Barometer makes:

* **Barometer** model is in Barometer.php (directory: app/Http/Models/)
* **BarometerController** fileBarometerController.php (directory: app/Http/Controllers/)
* **Barometer migration table** file 2023\_02\_07\_165602\_create\_barometers\_table.php (directory: database/migrations/)

1. In Laravel is MVC(Model-View-Controler) architecture. Models are always conected to the tables in database. Controllers use data that they get from Models and transfers to **View**, that are in directory resources/views/

**Blades** are in this directory. Thet generate html, variables are added using syntax very similar to Twig

1. **Routes** you can find in file web.php file routes.
2. So I then deleted the route to welcom.blade.php as well as Blade itself, both are already in Laravel after installation. Then created function route to index() from BarometerController.php which returns newly created blade barometer.blade .php.
3. To have a better development experience, I usually prefer to create a page layout, add styles, choose SVGs and fonts, add some animations and functions available without data while hard-coding values:

* **Styles** you can find in directory: recources/sass/app.scss
* **Javascript** can be found in directory: recourses/js/app.js

In recourses/js/bootstrap.js directory are already Echo library configurations, commented on, that will be uncommented and used in the future.

1. Javascript and styles was connected via Vite using syntax @vite(['resources/sass/app.scss', 'resources/js/app.js'])

in barometer.blade.php head.

1. My svg was taken from: <https://svgsilh.com/>;The responsive style was mainly built via @media queries, although in some places I've included the latest fashion of setting font size or element width with calculations of vw or vh.
2. Some animations made using @keyframes rule and @property feature, the second one is suported for Chrome and, by extension, Edge

And Action!

**Front-end**

1. Most of the action here happens in app.js file and some Laravel Echo parameters in bootstrap.js file. It also includes some explanation of how data is passed from or to the barometer.blade.php file as well as some calculations in the BarometerController and the Barometer model;
2. Functionality for navigalion buttons (Vilnius, Nida, Both) you can find in app.js:

* **2-6 lines**: DOM
* **10-21 lines:** the function assigned to the constant
* **57-59 lines:** call for function ( if cities data is availible and displayed)

1. Message is called when connection with API or data validation fails. Message already built in barometer.blade.php file 28 line, but is styled not to be displayd. In app.js file message is in:
   * 7 line DOM
   * **22-33 lines:** the function assigned to the constant, thet controls when a message should appear and animation of its appearance;
   * **33-34 lines:** the function assigned to the constant, thet controls when message should disappear and animation of its disappearance;
2. If statement in case there is no data in the database when the client loads (usually at the very beginning of the project or when data is not updated for 3 hors therefor deleted (*calculations in Batometer model*). In this statement is:

* **45-53 lines:** Echo is listening for the back-end events and reloads page if data is created;

1. Else statement in case there is data in database when client is loading (most of the time):

* **57-59 lines** already mentioned. There are calls for navigation function
* **61-75 lines:** javascript eventListener listening to load and then:
  + If statement in case data in dalabase is last updated later then 15minutes (but earlyer than 3 hour) ago calls for message add funcion. *Data about last update received from Model,where comparisons are made and passed to view blade, where variable is passed to javascript via dataset attribute (blade 28 line).*
  + Animation both cities weather appearence
  + Call for funtions that sets weather condition and air pressure data (see below)

1. Functions for information about weather data upload:

* **76-87 lines:** the function assigned to the constant, that takes data from dataset in Blade and rotates the pressure arrow up or down, that indicates whether the pressure is increasing and also inserts text about the increasing or decreasing pressure. If statement included to determine if pressure condition is already calculated and passed to blade (at the beginning of the project there is no previous value of the pressure or it is equal to the current pressure value, therefore no conclusions can be drawn about the pressure state)
* **88-95 lines:** the function assigned to the constant, that takes data from dataset in Blade and determines witch picture to use, also inserts text about weather condition.
* **96-109 lines:** the function assigned to the constant, that takes data from dataset in Blade and determines rotation of the arrow in barometer image. Formula calculated based on visual data, seeing that 0deg is when pressure is 1015 hPa and 90deg is when pressure is 1048hPa. 1048-1015 = 33hPa, so when arrow rotates 90deg, pressure changes 33hPa and when pressure changes in 1deg arrow should rotate 90/33hPa.
* **110-114 lines**: the function assigned to the constant, that calls for all of these three functions above.

1. In **bootstrap.js**

* **21-35 lines**: you can find Echo library parameters. These parameters are already written when you install Laravel. All you need to do in here is to uncomment them. Most of the data in these parameters are written in .env file.
* ATTENTION: without back-end configuration, and Event firing, you will not be able to use Echo library. I will intraduce you to information about these configurations in the next section.

1. Echo is listening when the data is updated ir database and Event is triggered in back-end, takes the broadcasted data and

* **116-148 lines :** includes it in Blade dataset value and calls for function described just above (**110-114 lines).** If API request or data validation previously failed therefore message was displayed, it will be removed.

1. Echo is listening, when API request fails and data was updated 15 minutes ago and

* **149-158 lines:** if it is updated later than 15 minutes ago, but earlyer that 3 hour ago, calls for function adding the message about last update, if last update is longer that 3 hours, data is deleted from database in a Barometer.php model file and in app.js file calls page to reload. Comparisons between current time and last update made in model;

**Back-end**

* Database created Maria DB, simply adding tha name of DB and saving it. In **.env 14 line** file BD\_DATABASE value changed to name I created in DB (barometer\_dara), so the data could be migrated.
* **18-22 lines** of the migration table **2023\_02\_07\_165602\_create\_barometers\_table.php.,** database/migrations directory, that creation is mentioned in preparation section**,** between id and timestamps, other neccesery columns added: city, weather\_condition, pressure, previous\_pressure and is\_rising,
* previous\_pressure and is\_rising is nullable, because in the beginning and in case of failed api\_call, there will be no data about previous pressure, and it will be impossible to count if the pressure is rising
* city column must be unique, so thet upserter function upadateOrCreate() could work. This upserter updates values of the one model, if its unique column variable is already in database and create if no matching unique variable exists.
* Migration was made via command line:

**php artisan migrate**

* In Barometer.php model you can find a function (not upserter) createOrUpdate, where you can find
* **16 line**: citys array declaration to constant;
* **18 line** protected variable $fillable, it is needed upserter updateOrCreate, to ensure that only the attributes you want to update can be mass-assigned. (*Mass assignment in Laravel refers to the ability to set multiple attributes on a model at once, rather than setting each attribute individually.)*
* request for API via Guzzle HTTP client library
* logic, what to do if the request fails,
* necessary data extraction
* this data validation,
* validation failure logic,
* saving them to database
* activation of Event WeatherUpdated when data is updated
* activation of Events WeatherUpdateFail when API request fails
* This function is activated every minute using Scheduler.
* **17-15 lines** of Kernel.php file in directory app/Console is schedule() function, where you can schedule tasks. Here is the call to the createOrUpdate() barometer model function scheduled every minute.
* Command line for make it run for once (convenient for developement process):

**php artisan schedule:run**

* using Laravel's scheduler, we only need to add a single cron configuration entry to our server that runs the schedule:run command every minute. Command line:

**\* \* \* \* \* cd /path-to-your-project && php artisan schedule:run >> /dev/null 2>&1**

Command line to make it run continuasly:

**php artisan schedule:work**

* Broadcastin via websockets.
* Installing websockets command line:

composer require beyondcode/laravel-websockets

* Creating migration file for storing statistic information while running your WebSocket server command line (in one line, i have a lack of space):

php artisan vendor:publish

--provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider"

--tag="migrations"

* Migration ran via command line:

php artisan migrate

* Publish the WebSocket configuration file via command line. After creating this file you will have it in config/websockets.php (one line, actually, lack of space):

php artisan vendor:publish

--provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider"

--tag="config"

* Now you need to install Pusher. Pusher is a real-time communication service that allows you to send and receive data between client and server through WebSockets. The communication between client and server happens through publish-subscribe (pub/sub) pattern where clients subscribe to channels to receive messages and servers publish messages to channels.
* The Pusher Channels PHP SDK using the Composer package manager installed via command line ( 7.2 version is not compatible with beyondcode/laravel-websockets, if I got it right):

composer require pusher/pusher-php-server:7.0.2

* Now you have to make some configurations:
  + 194 line of app.php of config file you can find         App\Providers\BroadcastServiceProvider::class, which was uncommented
  + In **18 line** of **broadcasting.php** file in config folder you should have to change the default BROADCAST DRIVER to pusher.
  + **18 line** of .**env** file: BROADCAST\_DRIVER changed to=pusher
  + **21 line** of .**env** file:cheked if QUEUE\_CONNECTION=sync as well as in 16 line of file queue.php in directory config QUEUE\_CONNECTION is sync
  + If you want to use pusher host for websockets, you should register to <https://dashboard.pusher.com>, then **46-48, 25 lines** of .**env** file PUSHER\_APP\_ID, PUSHER\_APP\_KEY, PUSHER\_APP\_SECRET, PUSHER\_APP\_CLUSTER could be filled with data that you get after registering in
  + Or during development process, you can make it locally, then in
    - **46-48 lines** of .**env** file create data for by yourself
    - **49 line** of .**env** file PUSHER\_HOST=localhost
    - **50 line** of .**env** file PUSHER\_PORT=6001
    - **51 line** of .**env** file PUSHER\_SCHEME=http
    - **42 line** of **broadcasting.php** in config directory 'encrypted' should be changed to false
    - **43 line** of **broadcasting.php** in config directory 'useTLS' should be changed to false
* Command line for running:

php artisan websockets:serve

* Echo javascript library installed via command line (Echo configurations in javascript is already uncommentd):

npm install --save-dev laravel-echo pusher-js

npm install

npm run dev

* In app/Events directory you can find 2 events. One event is for case, when database is updated, the other one is when update fails. Both function should implement ShouldBroadcast and uses public channel, that it is named simply Channel; in function broadcastOn() this Channel is returned and named. In javascript takes data from this Channel and in my case listens to event named by class name. All public variables or data in function broadcastWith is passed to Echo.
  + In **122 line** of **Barometer.php model** you can find when WeatherUpdated event is triggered. In this case data is changed in the blades;
  + **49 and 113 line** of **Barometer model**: WeatherUpdateFail event is triggered in case it is already 15 minutes since the last update happened, message with data of last update appears. In case it is 3 hours since the last data was updated, data is deleted from database, not to confuse clients and page (if open) reloads via javascript function.
* When the client loads the page or for those few occasions when page is forced to reload in javascript file, data is received via BarometerController. BarometerController gets gata from database (if it exists), evaluates, if data last update in the same way that it is evaluated in validator or API fail case, and pass the data to blade.

**Testing**

In the files app.js, Barometer.php model and BarometerController.php you will find comments with the word TEST in capital letters. Those comments you have to uncomment and delete word TEST, as well as comment on lines above – this way you will change the real values and see the changes in user interface. To trigger events, to activate changes you will have to run command line php artisan schedule:run. Sometimes you will need to migrate database with command php artisan migrate:fresh or reload the page. Sometimes you will have to uncomment few lines at the same time. After test is done, bring back it to previous condition.

I recommend starting with javascript testing, because during testing, backend calculations, whether the weather pressure is increasing or decreasing, can be done only after some time of program work when there is enough data to draw conclusions, and because these calculations can be ruined during testing.

So firstly write a command-line: php artisan schedule:work (to let it work every minute, so that with time previous weather pressure could differ from current pressure and the arrow would appear;

**recources/js/app.js**

CHANGES ON LOADING

* So take a closer look at barometer arrow and a weather condition, then comment on **73 or 74 line** of file and look twice.
* Take a closer look at pressure arrow that shows if pressure is increasing or decresing **(**if it already exists, if no, do this test anyway**) 83 line:** uncomment (you can change value to 1 or 0) **84 line:** comment on it and look twice.
* Take a look at weather picture and the text under it and uncomment **92 line**, comment on **91 line** and look twice.**.** Instead of snow, you can write dry, fair, stormy, change, rain.
* **100 and 103 line:** if you uncomment them you will see in values in console of air pressure and how many degrees should the arrow rotate;
* Take a closer look at barometer arrow and uncomment **105 line**, comment on **106 line** and look twice.

CHANGES VIA WEBSOCKETS

* Take a closer look at pressure arrow that shows if pressure is increasing or decresing **(**if it already exists, if no, do this test anyway**) 126 line:** uncomment (you can change value to 1 or 0) **125 line:** comment on it, then run command line **php artisan schedule:run** and look twice.
* Take a look at weather picture and the text under it and uncomment **133 line,** uncomment **132 line**, then run command line **php artisan schedule:run** and look twice.
* Take a closer look at barometer arrow and uncomment **105 line**, comment on **106 line**, then run command line **php artisan schedule:run** and look twice.
* Take a closer look at barometer arrow and uncomment **141 line**, comment on **140 line**, then run command line **php artisan schedule:run** and look twice.

**app/Http/Models/Barometer.php**

CHANGES VIA WEBSOCKET

* Follow the link :

<https://api.openweathermap.org/data/2.5/weather?q=nida,LT&appid=4073f66b93000ba7712dff2f2f0628a5>

* Here you will find and object. In this object there is a named array ‘weather‘, in this array is value ‘main‘. This value determines, wether condtition and which picture to include. You can see which values are asigned to dry, fair, snow, change, stormy or rain in **55-62 lines**, check if it maches.
* Press ctyrl+f and find ‚pressure‘ – this is our current pressure value. You can compare it to data you see in user interface or console.log the data in **app.js 100 line.**
* From now we will be ruining the data. This time wait for message. It appears in 3 seconds. Uncomment **26 and 40 lines,** comment on **38 line** and run command line: **php artisan schedule:run** and wait for message.
* Uncomment **26, 40 and 41lines,** comment on **38, 39 line** and run command line: **php artisan schedule:run** and watch the page reload and data disappear; After this test, bring back the previous data and run the commant line: **php artisan schedule:run.** Data is needed for future tests.
* Take a look at weather pictures and texts under them and uncomment **61 line** and run command line: **php artisan schedule:run** and look twice;
* Take a look at barometer arrows, uncomment **63 line** and run command line: **php artisan schedule:run** and look twice;
* Take a look at the weather pressure arrows. If it still did not appear (what is probable), at this test it wil appear with false data; anyway, after some time data will become correct. So uncomment **70 line** and run command line: **php artisan schedule:run** and look at the arrow, the same should be done with **71 line.** After second line being uncommented the arrow shoud go up; You can do it visa-verca.
* Validation test, wait for message. It appears in 3 seconds. Uncomment **87 and 102 lines,** comment on **104 line** and run command line: **php artisan schedule:run** and wait for message.
* Uncomment **87 and 102, 103 lines,** comment on **104, 105 lines** and run command line: **php artisan schedule:run** and watch page reload and data disappear
* You can also try to add

If the command line php artisan schedule:work is not running, maby it would be better to run it now, so data would fix in time;

**\*\* you can also test controller, but there you will find the same function as in the model, exept there is no call for event inside, because controller only wakes up on reload.**