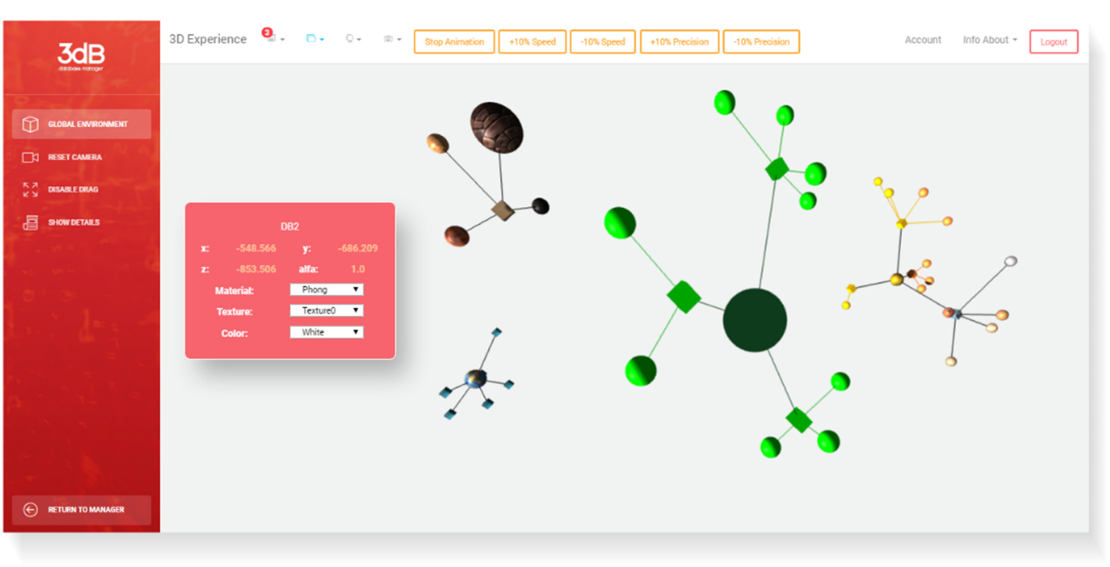
As we can see also lights and camera are initialized inside the function. **PHP** code is used to print and create the javascript code required to build all the complex objects relative to registered user. The function **createObject** is delegated to effectively create the specific object inside the DatabaseStruct while the function **connect** ,belonging to the same class, create a line object that connects father and children in the Scene. When an object is connected to another one, that object is inserted inside the children of the **Mesh** Object so that in DatabaseStruct's objects set is mantained only one **Mesh** that is the reference to a specified Database created thorough the interfaces in the administration panel.

DatabaseStruct object has other functions with which can be generated a spefied number of children for a specific object. It also manage the interaction between its structure and the data stored inside the physical storage through the call of the right **API endpoint**.



When an object is clicked or dragged a information panel is displayed. It contrains informations about object position, alpha value, used material, used texture and used color. Through this panel you can also change material, texture and color of the specified object.
A complete vision of how much objects are present in scene and how they are organized is also showed when the first button in the top bar is clicked. Scene details are also showed in another panel that appears when the **show details** voice is clicked (principal navigation menu).

The user can intact directly with all the object in the scene changing the position that is asynchronous updated in the physical storage. When an object is moved all his children, in autonomous way, are updated because of the hierarchy previous explained. Only object's children are updated when their father is moving, moreover line connecting objects are recreated to respect new objects position.

When an object is clicked, the object and his children are underlined in two different selectable kind: **Object Animation** and **Camera Animation**.
If **Object animation** is selected the **animation_type** variable is updated and selected object is underlines by a **Gravity** simulation so that its children start to gravitate around the object that rotate on x,y and z axis. The **Gravity** simulation was easily created by changing the rotation of the selected object in the **render** function.

```
function render() {
  if(animated_object != null){
    if(animation_type == "camera"){
      if(!(camera.position.y <= animated_object.position.y+interval_limit &&  camera.position.y >= animated_object.position.y-interval_limit)){
        if(camera.position.y > animated_object.position.y){
          setCameraPostion(camera.position.x, camera.position.y -= increment, camera.position.z);
        }else{
          setCameraPostion(camera.position.x, camera.position.y += increment, camera.position.z);
        }
      }else{
        founded[1] = true;
      }
      if(!(camera.position.z <= animated_object.position.z+interval_limit &&  camera.position.z >= animated_object.position.z-interval_limit)){
        if(camera.position.z > animated_object.position.z){
          setCameraPostion(camera.position.x, camera.position.y, camera.position.z-= increment);
        }else{
          setCameraPostion(camera.position.x, camera.position.y, camera.position.z+= increment);
        }
      }else{
        founded[2] = true;
      }
      if(!(camera.position.x <= animated_object.position.x+interval_limit &&  camera.position.x >= animated_object.position.x-interval_limit)){
        if(camera.position.x > animated_object.position.x){
          setCameraPostion(camera.position.x -= increment, camera.position.y, camera.position.z);
        }else{
          setCameraPostion(camera.position.x += increment, camera.position.y, camera.position.z);
        }
      }else{
        founded[0] = true;
      }

      if(founded[0] && founded[1] && founded[2]){
        animated_object = null;
        founded = [false,false,false];
        //camera.updateProjectionMatrix();
        $('#Button-Stop').addClass("disabled");
      }
    }else if(animation_type == "object"){
      animated_object.rotation.z += 0.02;
      animated_object.rotation.z += 0.02;
      animated_object.rotation.y += 0.02;
    }

  }
  if(display_type == "gravity"){
    if(animated_objects != null){
      animated_objects.forEach(function(animated_object){
        animated_object.rotation.x += 0.013;
        animated_object.rotation.y += 0.013;
        animated_object.rotation.z += 0.013;
      });
    }
  }
  controls.update();
  renderer.render( scene, camera );
}
```

In the other case if **Camera Animation** is selected the camera is moved until the object fit the screen. This animation is managed by two variables: **interval limit** and **increment**; The first is used to set a space interval in which stop the camera so that to fit the object in the **camera view** while the increment is the factor added to the camera position coordinate between one renderization step and the next.
The user can manage these variables by clicking **Speed** and **Precision** buttons on the top of the screen. When one of this buttons are clicked a panel showing the parameters value is displayed.

If **No Animation** is choosed, the mesh was selected is not underlined and user can navigate the scene through W-S-A-D keys on his keyboard.

The user can also reset camera position by clicking the **Reset Camera** voice in the left side menu. When it is clicked the function **setCameraPosition** is called and it change the camera coordinates to the initial position, resetting the fov and all camera's parameters to default values. By clicking **Disable Drag** voice, global drag controls are disabled by disablig drag controls of each complex structure present in the scene. Disabling the drag, user can navigate the scene witout modifing any object position.

Because of Javascript limitations, **Textures** are precalculated and renderized inside specified materials. The materials are applied to the selected Mesh only when the user changes the field relative to Texture application.
As we can notice in the init function at line 1716, after that all DatabaseStrct are builded the function **createTextureMaterial** is called. This function create the material after that the specified texture is loaded by the **Texture loader**. From the fact that javascript runs on a single thread and images uploading process is an expensive operation in term of resource and time, texture are inserted initially in a global array and are applied to specific mesh only after that all texture are loaded. Until all texture are not applied, global interface is blocked with a graphic loader.

Created Databases could be displayed with two different visualization: **Graph vh iew** or **Gravity view**.
The first was is showed in the previous images while the second apply the **Gravity** effect explained before to the whole structures in the scene. If both **Gravity view** and **Object animation** are selected the rotation effect on the axis is cumulated;

Selecting the world icon on the top section menu, user can change scene background color choosing between a light or a darker version. The global information about Scene configuration are not stored in any storage but could be reapplied in simple way. If **Dark** theme is selected and an object is dragged or clicked, line's material are checked: if material color is not in contrast with the theme a contrast color is selected and lines that connects the specific object are updated. The same behaviour is applied when **Lighter** theme is selected.

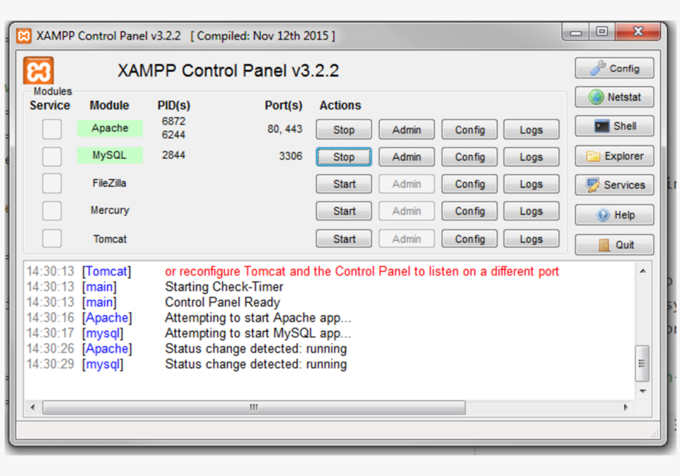
Setup and Run

This section explains how to setup the project and run it on a the local machine, the needed technologies and versions.

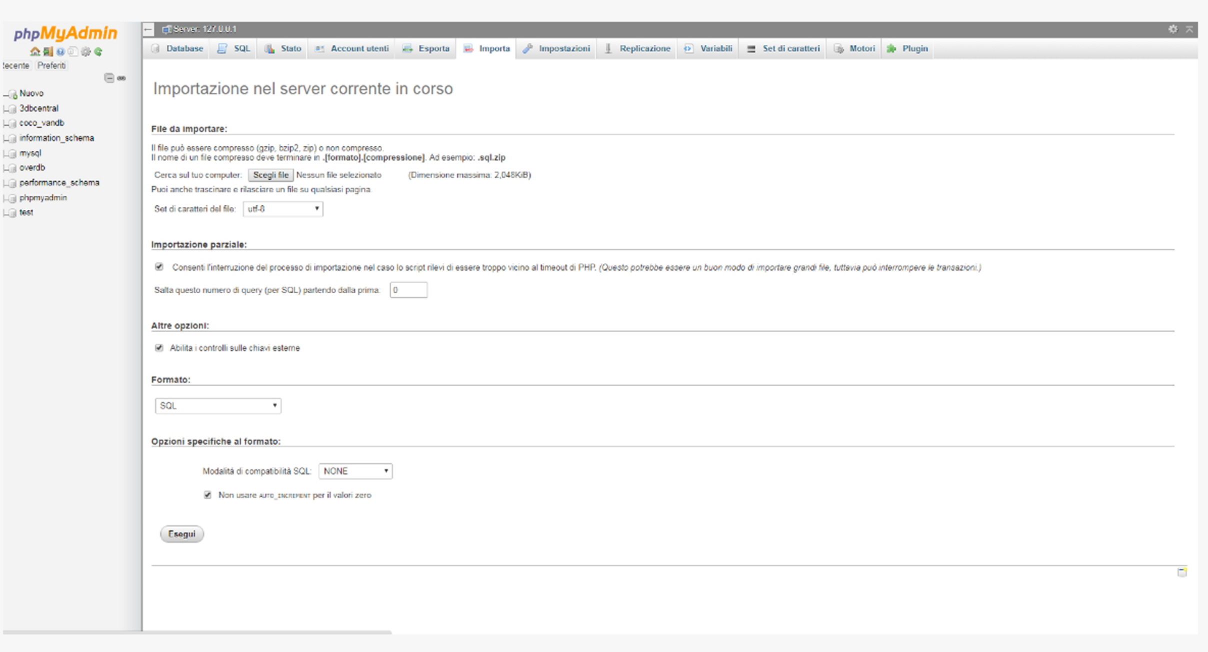
XAMPP - All you need

If you work on Windows, Linux or OS X you can use XAMPP to setup all you need to run this project on your local machine. XAMPP is the most popular PHP development environment, is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.
If you don't know how to install XAMPP on your machine you can easily foud the procedure on the official site.

If you have successfully installed XAMPP now you can copy files contained in the directory called **3dB** inside the local server directory (on Windows usually at C:\xampp\htdocs).
After this step you have to run **XAMPP.exe** and starts **Apache** and **MySql** services through XAMPP interface.



In the project directory you can found a subdirectory called **Documentation**, inside this directory you can found the **3dbcentral.sql** file that you have to import through **phpMyAdmin** interface.
To open **phpMyAdmin** you have to visit in your browser the address http://localhost/dashboard/. If you server is running you should see the panel presented in the first image of this section.
By clicking on **phpMyAdmin** you can address the interface through which you have the possibility to import **3dbcentral** database.



At the end of this simple procedure you can see the site at the location http://localhost/3dB/index.php. In this page first of all you have to register a new Account that allows to you to enter in your private section (dashboard.php). In your private section you can add new Databases, Tables and Columns, change your informations and see the tracked history of your account.

Recommendation

MySQL version 5.0.12
phpMyAdmin version 4.6.5.2
PHP version 7.1.1

3dB
database manager