# Reference guide

This guide is made with the intention so a new future developer can easily adapt to the project. We will be discussing the backend's framework and it's features that make coding look simple.

The framework we are using is called vibe.d. It allowes us to make a personalised backend server that works by passing JSON objects with which we can later scale the application. The backend is segmented in 3 seperate files. db.d helps us connect and use the mysql database, app.d is the startup file in which we set the default settings and rest.d which defines all the routes, passing parameters and code flow.

In rest.d we can quickly spot an interface that has a collection of all of the routes we can use. Each function is made from 4 parts that define the recieving parameters, returning parameters, route name and route HTTP verb. For example ‘string postLogin(string username, string password)‘ tells us this function will return a JSON string, the substring "post" in the name tells us it will call this function on the HTTP verb POST, the rest of the name tells us the route of the function and the parameters tell us what exactly do we request from the client (in JSON format). We can also see an odd looking function ‘@authCookie‘. This function is an alias which simply gets the clients cookie which we will later parse to see if the user is authenticated and retrieve his data. It passes the information to the variable called "auth" which is defined as an optional parameter. The reason we need it that way is so we can respond appropriately without the framework throwing an error at the client.

All of our functions are later defined in a class that extends the interface we were talking about. We can also set our own functions in this class, that were not made for the clients to call directly, so we simply mark those functions as private. A few examples for these functions are ‘getSalt()‘, ‘parseCookie()‘ and ‘hashFunction()‘. The other functions must look the same as in the interface and it's also recommended that they work autonomously. They are also more or less self-documenting, since we split the work into other functions and because of the frameworks design.

Overall the code is really neat and you can quickly spot out the bugs and errors. And furthermore, even if that happens, vibe.d knows how to appropriately react so your code keeps running and only one user leaves the site slightly unhappy. There is also always room for improvement. In the future I'd like add that instead of returning bad variables, to mark that the user is not authorised, I'd return an HTTP status code with a user friendly message.