

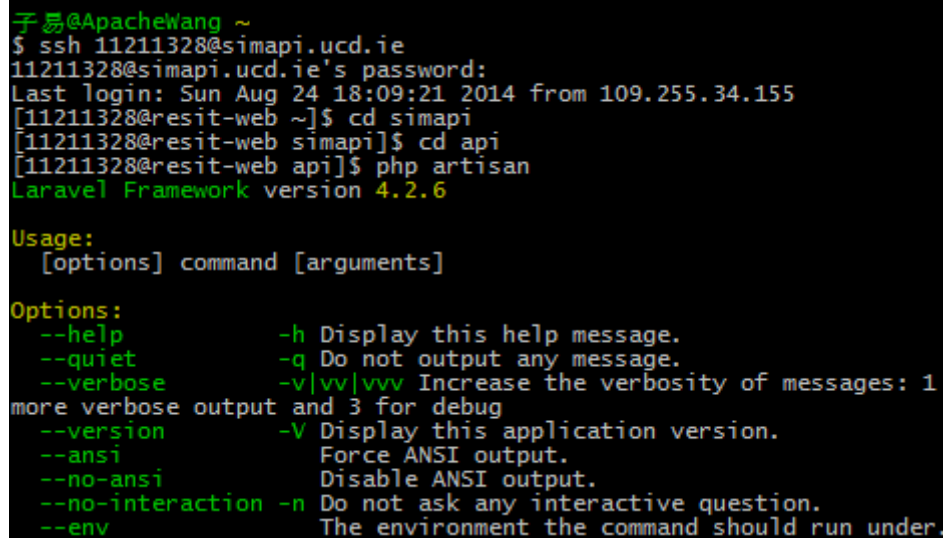
# SimAPI – Web Platform step by step guide

## Laravel environment

Laravel is a PHP framework which makes it much easier to write a RESTful API or design a website. With the help of artisan, the command-line interface included with Laravel, I can get a number of helpful commands while developing. And php blade makes it easier to build different web interfaces with a same title content. Using composer it is convenient to install Laravel on the local computer. However installing Laravel on a Apache server requires PHP environment (PHP>=5.4). In order to check whether the server is ready I need to get SSH connected using Cygwin. When SSH is connected, type in command:

1. `composer global require "laravel / installer = ~1.1"`
2. `composer create - project laravel / laravel--prefer - dist`

Now laravel has been installed on the server. When type in command: `php artisan`, I can get the version of the laravel framework that I am using.



```
子易@ApacheWang ~
$ ssh 11211328@simapi.ucd.ie
11211328@simapi.ucd.ie's password:
Last login: Sun Aug 24 18:09:21 2014 from 109.255.34.155
[11211328@resit-web ~]$ cd simapi
[11211328@resit-web simapi]$ cd api
[11211328@resit-web api]$ php artisan
Laravel Framework version 4.2.6

Usage:
 [options] command [arguments]

Options:
 --help           -h Display this help message.
 --quiet         -q Do not output any message.
 --verbose       -v|vv|vvv Increase the verbosity of messages: 1
more verbose output and 3 for debug
 --version       -V Display this application version.
 --ansi          Force ANSI output.
 --no-ansi       Disable ANSI output.
 --no-interaction -n Do not ask any interactive question.
 --env           The environment the command should run under.
```

Figure 5.1.1 Laravel Install

## Set up developing environment

In order to move and create file easily, I used Cyberduck to get access to SFTP. Under the new project folder `simapi`, I create a new folder called `api` to install laravel. In the folder of `api`, there are `start`, `models`, `commands`, `lang`, `database`, `storage`, `tests`, `config`, `views`, `contollers`, `bootstrap` as well as a PHP file named `routes`. By default the server will consider `index.php` in the folder of `htdocs` as entry point. So that I need to set the proper path in `paths.php` under the folder of `bootstrap`.

Firstly, I defined the path to the application directory. I did not make any changes here because it will find the correct directory by the original code as followed.

1. `'app' => __DIR__.'../../app'`

Secondly, I need to set the public path which contains the assets for my API such as JavaScript and CSS files, and also contains the primary entry point for web requests.

1. `'public' => __DIR__.'../../htdocs'`

Thirdly the base path which is the root of Laravel installation and the storage path.

```
1. 'base' => __DIR__.'/'
2. 'storage' => __DIR__.'../app/storage'
```

## Route testing

For testing I set the default route with a return of “Hello! ”.

```
1. Route::get('/', ['as' => 'home', function() {
2.     Return "Hello!";
3. }]);
```

When type in the entry url in web browser , it shows “Hello!”.



Figure 5.1.3 Home page test

Now that Laravel is working, it is time for developing the real homepage. In routes.php, set the entry point return with a view named "hello".

```
1. Route::get('/', ['as' => 'home', function() {
2.     return View::make('hello');
3. }]);
```

In order to avoid writing the same code duplicately, I decided to use php blade. In php blade, a web page has three parts:

- extends: use the template layouts
- section: main content
- stop: the end of a page

Before building the homepage, a template layout is required. In the layout, I used CSS Bootstrap to make a simple and beautiful web page. Laravel blade php is able to catch flash message and global message by using sessions. In the code fragment above , I get the support of latest compiled and minified CSS , optional theme and compiled and minified JavaScript through the links. A fixed navbar is built on the top of the page as well as the links of API documentation and API blueprint .In the drop down menu , Laravel Auth() is used to display different content judging by whether a user has already logged in or not.

If Auth::check() returns true , “sign out” and “change password” will be displayed. If Auth::check() returns false , then “sign in” ,”sign up” and “forgot password” will show up once the user click on the drop down menu “Profile”.

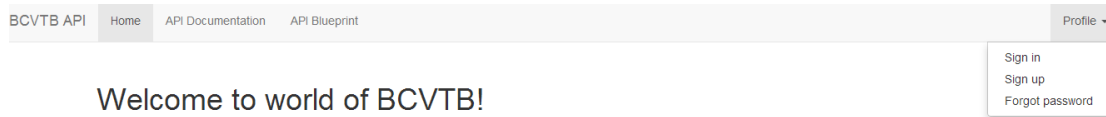


Figure 5.1.3 Home Page

## Database

Since the simple home page has been set up , in order to build the authentication system , the connection between the web server and MySQL database should be ready in the first place.

In laravel, the connection is built in the file of database.php under the folder of app/config.

```
'mysql' => array(
    'driver' => 'mysql',
    'host' => 'resit-db.ucd.ie',
    'database' => 'simapi_db1',
    'username' => 'simapiuser',
    'password' => 'pg31bd2!',
    'charset' => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix' => ''
)
```

Once the connected, it is possible for users to get access to the database remotely through the web server. The database has already been designed and built in MySQL , however there is still another step before building the authentication system. In Laravel,since primary key and foreign key constrains can not be read directly, structure of the data base is required to be interpreted. What is more, it is also required to make it clear that which columns Laravel could change and which columns should be protected from Laravel. Take User table for an example:

```
1.
2. protected $table = 'users';
3.
4. protected $primaryKey = 'idUser';
5.
6. protected $fillable = array('email', 'username', 'password', 'code', 'password_temp', 'active');
7.
8. protected $hidden = array('password', 'remember_token');
9.
10. public function getAuthIdentifier() {
11.     return $this -> getKey();
12. }
13. public function getAuthPassword() {
14.     return $this -> password;
15. }
16. public function getReminderEmail() {
17.     return $this -> email;
```

```

18. }
19. public function Instances() {
20.     return $this - > hasMany('Instance', 'User_idUser', 'idUser');
21. }
22. }

```

User.php is created in the folder of app/models. In User.php, it selects the table named "users", sets the primary key "idUser" and make columns "email","username","password","code","password\_temp" and "active" fillable. At the same time, for security concern, columns "password" and "remember\_token" are hidden. Functions getAuthIdentifier() and getAuthPassword() are modified for authentication. GetReminderEmail() can get the e-mail address where password reminders are sent. Lastly, Instances() explain the one to many relationship between user and Instance which is one user can have many instances. Besides User.php , all the other tables are described similarly following the rules in the database.

## Authentication

Since data base is ready now, it is time for developing authentication endpoints. According to the API documentation, the basic authentication including create a new account , login and logout.

### Create a new account

To build a endpoint, I need to set the proper route first.Because create a new accounts need post data like username and hashed password , I create a post route as followed:

```

1. Route::post('/accounts/create',
2.     array('as' => 'accounts-store',
3.         'uses' => 'AccountsController@postcreate')
4. );

```

Laravel is using MVC which are model,view and controller. Here I need to create a new controller called AccountsController to control all the accounts.In the controller , I also need to create a method called postcreate to get the input data , insert them into data base and return with the user id of the new user.

In a web browser,it is usually not hard to test a "get" endpoint, but it can be pretty tricky when it comes to "post" ones. Postman is a chrome application which is created to help developers be more efficient while working with APIs.

simapi.ucd.ie/accounts/create POST

form-data x-www-form-urlencoded raw

email	email@gmail.com	✕
username	testUser	✕
password	123456	✕
Key	Value	

Send Preview Add to collection

Body Cookies (3) Headers (7) STATUS 200 OK TIME 816 ms

Pretty Raw Preview

```
{"user_id":142}
```

Figure 5.1.5.1 Create a new account

Create a new account with username ,email and password.Once created, it returns the id for the new account.It takes 816 ms in total.

## Login

Set the route for login as simapi.ucd.ie/login:

1. `Route::post('/login', array('as' => 'login', 'uses' => 'AccountsController@login'));`

In AccountsController create a new method called login which can get the input username and password and use Auth::attempt in Laravel to check whether the user exists or not. If username and password is valid ,return user id and user name. If username not exists or password dose not match with the username, then return error with message Unauthenticated.

http://simapi.ucd.ie/login POST

form-data x-www-form-urlencoded raw

username testUser ✕

password 123456 ✕

Key Value

Send Preview Add to collection

Body Cookies (3) Headers (7) STATUS 200 OK TIME 569 ms

Pretty Raw Preview

```
{
  "user_id": "142",
  "username": "testUser"
}
```

Figure 5.1.5.2 Login(1)

Login with the username and password created before. It returns the id of the user as well as the username. It takes 569 ms to check and login.

http://simapi.ucd.ie/login POST

form-data x-www-form-urlencoded raw

username testUserfake ✕

password 123456 ✕

Key Value

Send Preview Add to collection

Body Cookies (3) Headers (7) STATUS 401 Unauthorized TIME 347 ms

Pretty Raw Preview

```
{"status": "error", "message": "Unauthenticated"}
```

Figure 5.1.5.2 Login(2)

When login with wrong username or invalid password, API returns an error saying Unauthenticated. It takes 347 ms.

## Logout

Set the route for logout as simapi.ucd.ie/logout:

1. `Route::get('/logout', array('as' => 'logout', 'uses' => 'AccountsController@logout'));`

Simply use the `Auth::logout()` in Laravel to log out

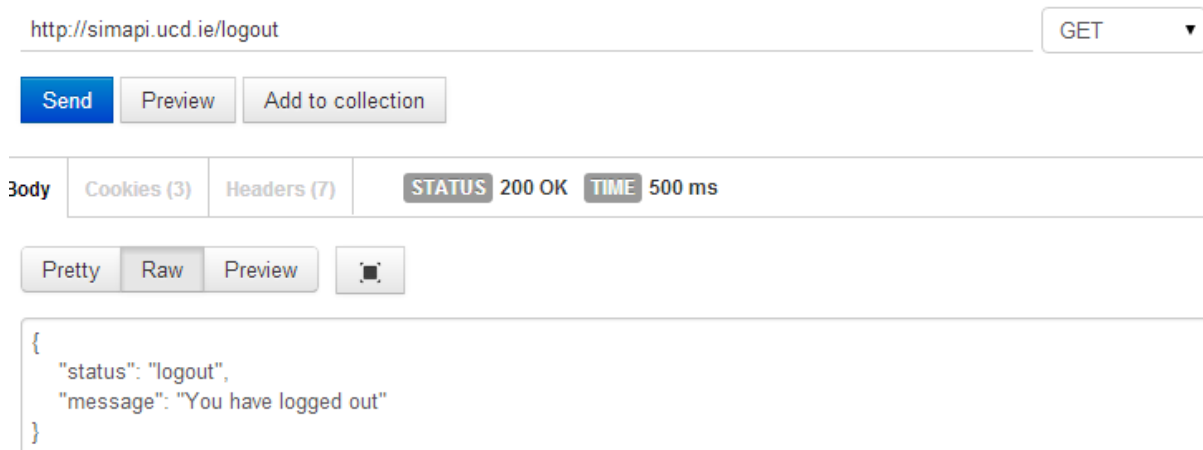


Figure 5.1.5.3 Logout

When logout , it return the message:"You have logged out". Logout takes 500 ms.

## Create instance

When a user login , he or she should be able to create a instance representing the simulation that he or she wants to run. Since one user could create many instances, user id should be used in URL as a parameters.

1. Route::post('{user\_id}/create\_instance',  
array('as' => 'create-instance-post',  
'uses' => 'InstanceController@postCreate'));

Create instance needs a new controller called InstanceController. In the controller , I created a new method called postCreate. In postCreate, it takes the user id from URL , inserts the user id and instance name into table Instance. Meanwhile it sets begin to 0 and returns with the new instance id.

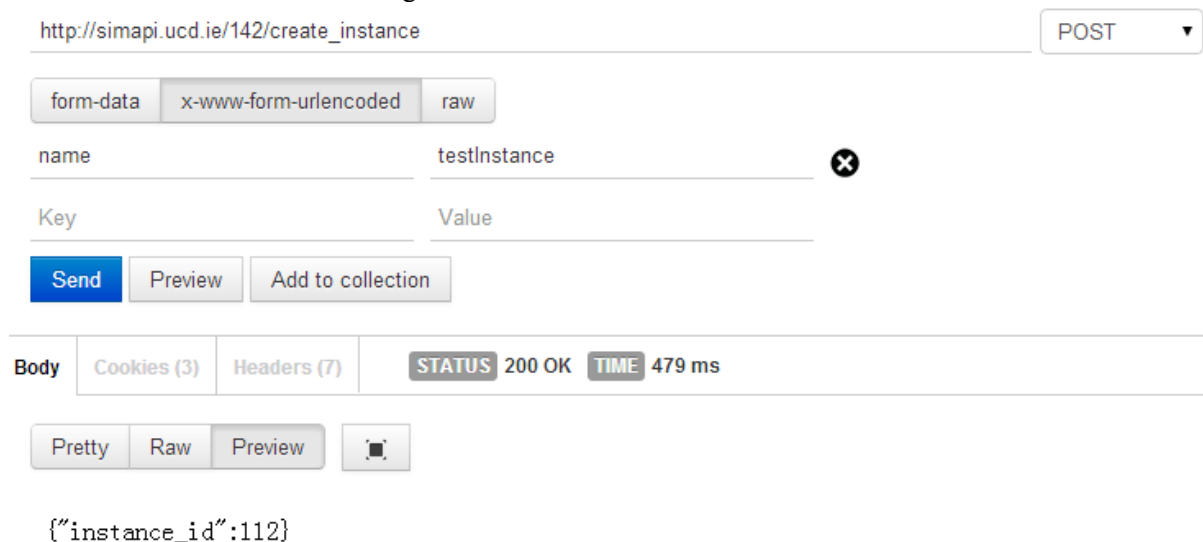


Figure 5.1.6.1 Create Instance

Create a new instance for the user which user id equals 142. Get the response of the new instance id 112 in 479 ms.

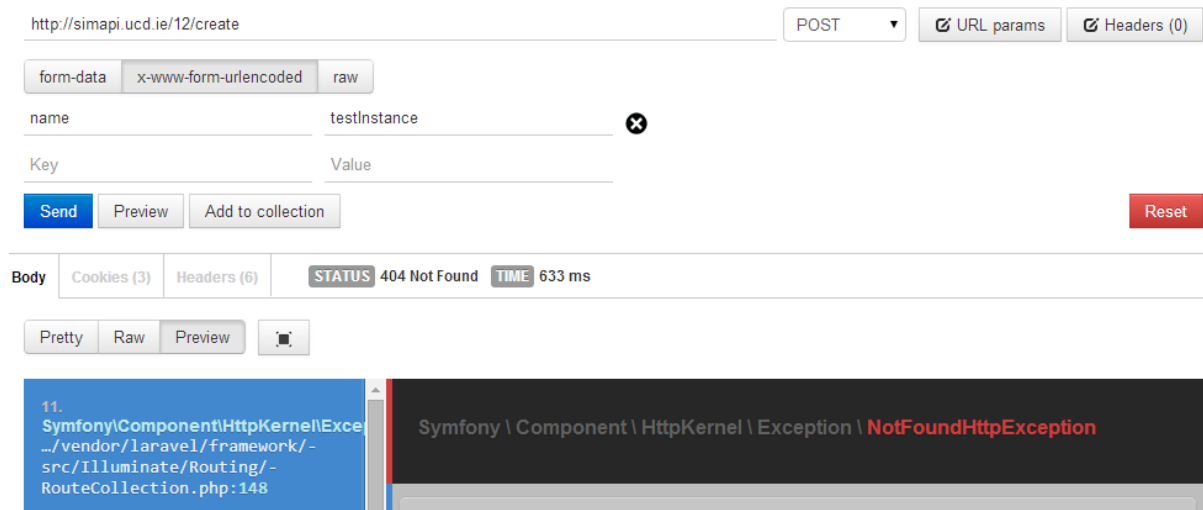


Figure 5.1.6.2 Create Instance error test

When create a instance using a invalid user id, it returns not found http exception. It takes 633 ms.

## Begin Simulation

At the beginning, begin is set to 0. To start simulation, API need get the instance id to see which instance the user wants to start and then set begin to 1.

1. `Route::post('/{instance_id}/begin',  
array('as' => 'postbegin',  
'uses' => 'InstanceController@postBegin'));`

Function `postBegin` is created in `InstanceController`. In the method `postbegin`, API selects instance by `instance_id`. Once found, `begin` is set to 1 and instance is saved.

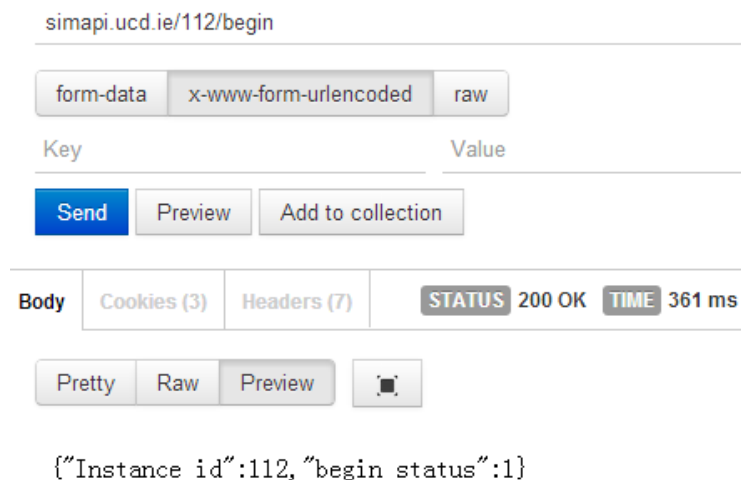


Figure 5.1.7 .1Begin simulation

Set the begin column of selected instance to 1. It takes 361 ms.



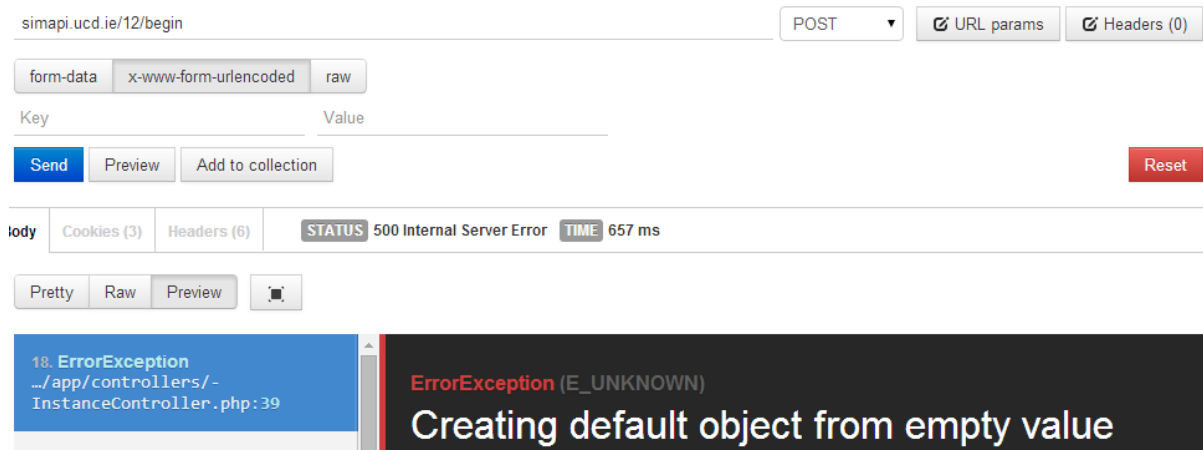


Figure 5.1.7 .2 Begin simulation error test

When begin a instance using a invalid instance id, it returns 500 internal server error with a message saying creating default object from empty value error. It takes 657 ms.

## Create action

In the table of action, there are two columns named `TSetHea` and `TSetCoo` which represents the range of temperature that is considered to be comfortable. If temperature goes higher than `TSetCoo`, the system will start to make it cooler. If temperature drops lower than `TSetHea`, the system will start the building heating system. In short, the change of these two columns may change the action.

1. `Route::post('/{instance_id}/{timestep_id}/setTem',  
array('as' => 'setTem',  
'uses' => 'ActionController@postSetTem'));`

Method `postSetTem` in `ActionController` is created to get the `TSetHea` and `TSetCoo` values and insert them into `Actions` table. Insert input `TSetHea` and `TSetCoo` as well as `timestep_id`, `instance_id` into `Actions` table and take the `Action_id` as response.

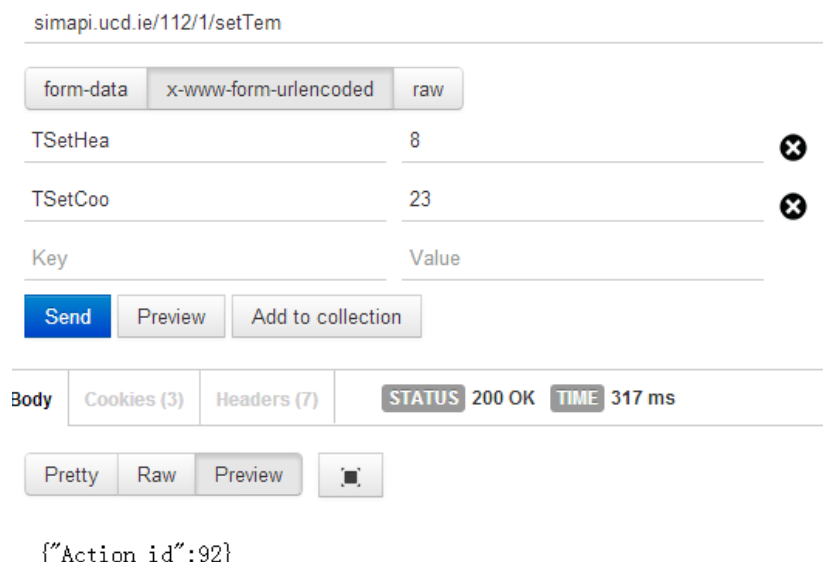


Figure 5.1.8.1 Create an action

Post heating temperature and cooling temperature of time step 1 of instance 112 to API . Get the action id 92 in 317 ms.

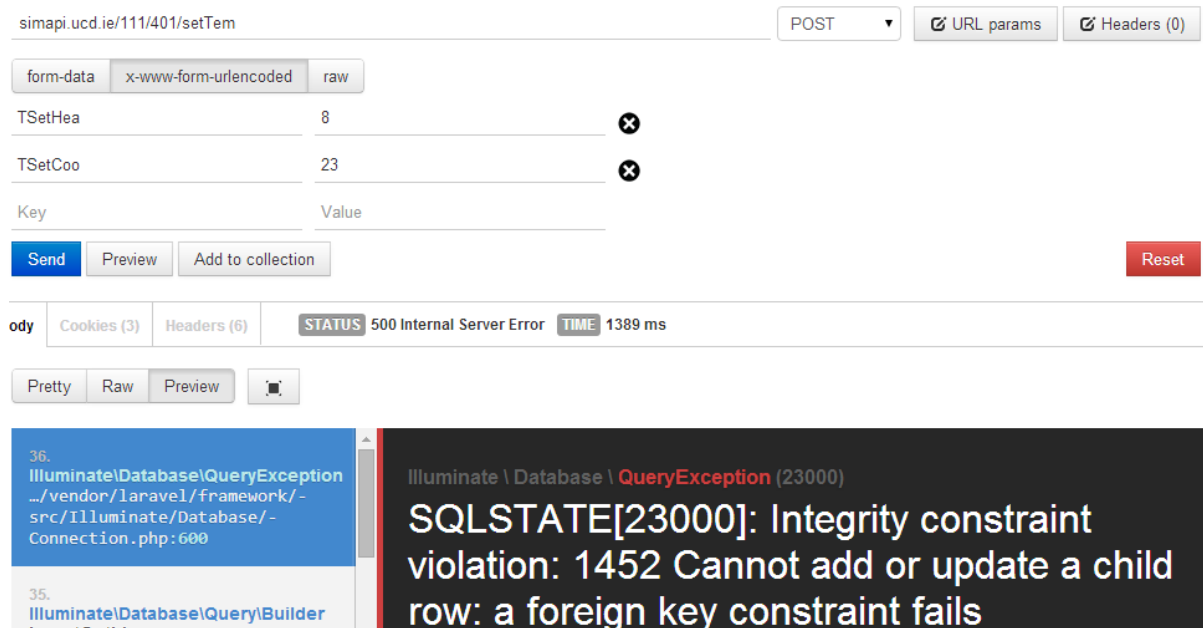


Figure 5.1.8.1 Create an action with error

When create an action using a invalid time step id, it returns 500 internal server error. It takes 1389 ms.

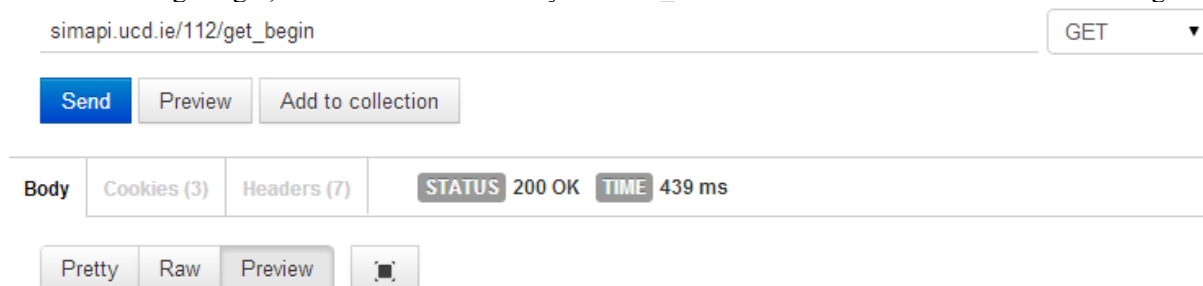
## Get Status of Instance

In the table of Instance ,there is a column called begin.When begin is 0, it means the simulation has not started yet. When begin is 1, it means the simulation is running.When begin is 2, it means the simulation has finished and stoped. It is important to make it possible for users to know which status the instance is in.So that when there is no new data, users would be able to know whether they should wait or not.

1. 

```
Route::get('/{instance_id}/get_begin',
array('as' => 'getbegin',
'uses' => 'InstanceController@getBegin'));
```

In order to get the status of a instance, it is obvious that instance id should be put into the URL as a parameter. In the method getbegin, API searches instance by instance\_id and return the value in the column of begin.



1

Figure 5.1.9.1 Get begin status

Get the begin status of instance 112 in 439 ms.

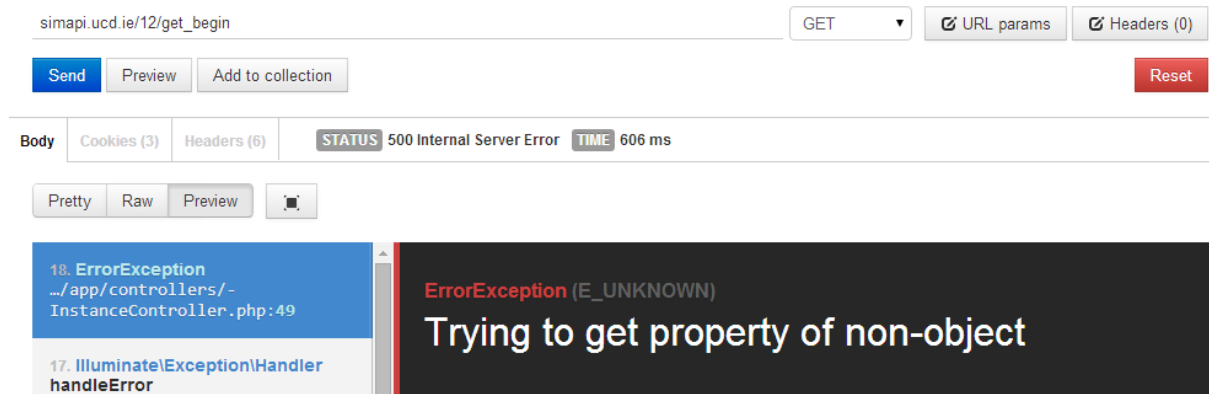


Figure 5.1.9.2 Get begin status with error

When check the status of a instance with an invalid instance id, it returns 500 internal server error. It takes 606ms.

## Get Sensor

Users can get values from sensor table whenever they want during the simulation as long as they have the timestep id , instance id and sensor id.

1. 

```
Route::get('/{instance_id}/{sensor_id}/{timestep_id}/get_sensor',
array('as' => 'getSensor',
'uses' => 'SensorController@getSensor'));
```

Since the sensor table have three primary keys, there are three parameters in the URL to build a RESTful API. SensorController and function `getSensor` are described as followed:

In the function, API gets the specific row in the table of Sensor by `sensor_id`, `timestep_id` and `instance_id`. If the row is found, API would return the temperature outside (TOut) , temperature in room (TRoom) along with the sensor id and the timestep id as a response. If the row is not found, API returns null.

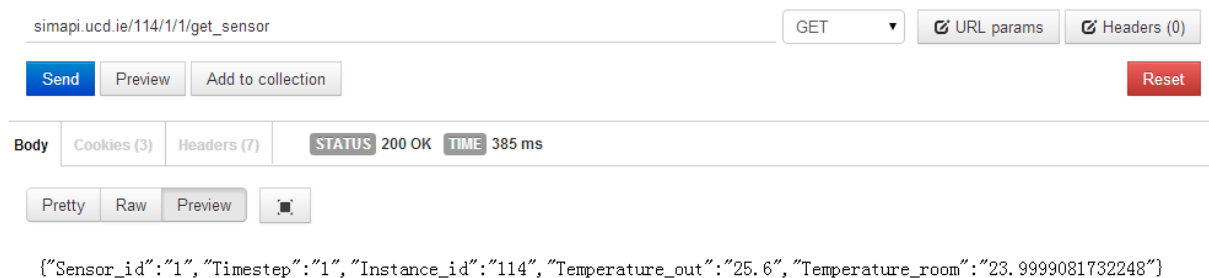


Figure 5.1.10.1 Get Sensor value

Get the sensor value of instance 114 ,sensor 1 , timestep 1. Get the response of json code contains `sensor_id`, `timestep_id`, `Instance_id` , `Temperature_out` and `Temperature_room` in 385 ms.

simapi.ucd.ie/144/1/1/get\_sensor GET

Send Preview Add to collection

---

Body Cookies (3) Headers (7) STATUS 200 OK TIME 364 ms

Pretty Raw Preview

Sensor Not found

Figure 5.1.10.2 Get Sensor value with invalid instance id

simapi.ucd.ie/114/2/1/get\_sensor GET

Send Preview Add to collection

---

Body Cookies (3) Headers (7) STATUS 200 OK TIME 310 ms

Pretty Raw Preview

Sensor Not found

Figure 5.1.10.3 Get Sensor value with invalid sensor id

simapi.ucd.ie/114/1/400/get\_sensor GET

Send Preview Add to collection

---

Body Cookies (3) Headers (7) STATUS 200 OK TIME 343 ms

Pretty Raw Preview

Sensor Not found

Figure 5.1.10.3 Get Sensor value with invalid time step id

When check a sensor value with invalid instance id , sensor id or time step id, it returns “Sensor Not found”.It takes around 340 ms.

## BCVTB API website

As described previously in 5.1.3 Homepage, I build the BCVTB API website including API documentation and basic authentication system. The authentication system allows users to change the password, recover the password by e-mail and so on. At the beginning, I would like to test the API by the website as well, but later I find out that on one hand it is better to make the website focus on the API documentation, on the other hand, functions like activating a new account is not the key point in this project. Thus I decided to write a java program to test the API. However I still build the basic authentication system on the website as well.

## MVC UML

MVC is used in laravel to build a web site. There are accounts controller, action controller , sensor controller and instance controller. So far , only accounts controller and instance controller have views to control.

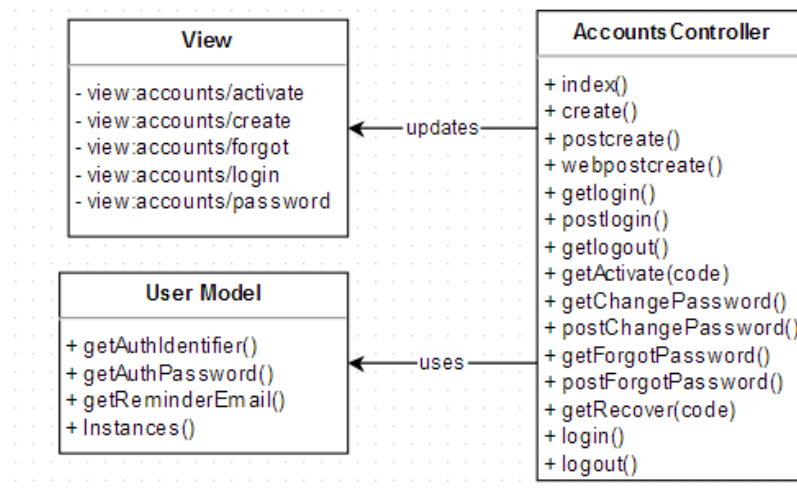


Figure 5.2.1.1 Accounts Controller

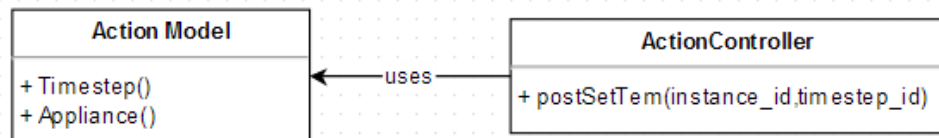


Figure 5.2.1.2 Action Controller

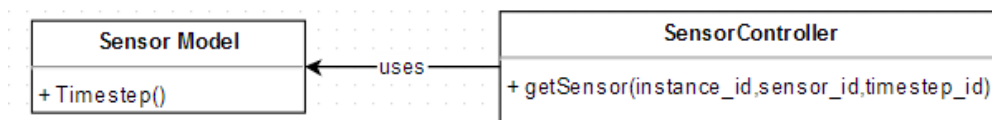


Figure 5.2.1.3 Sensor Controller

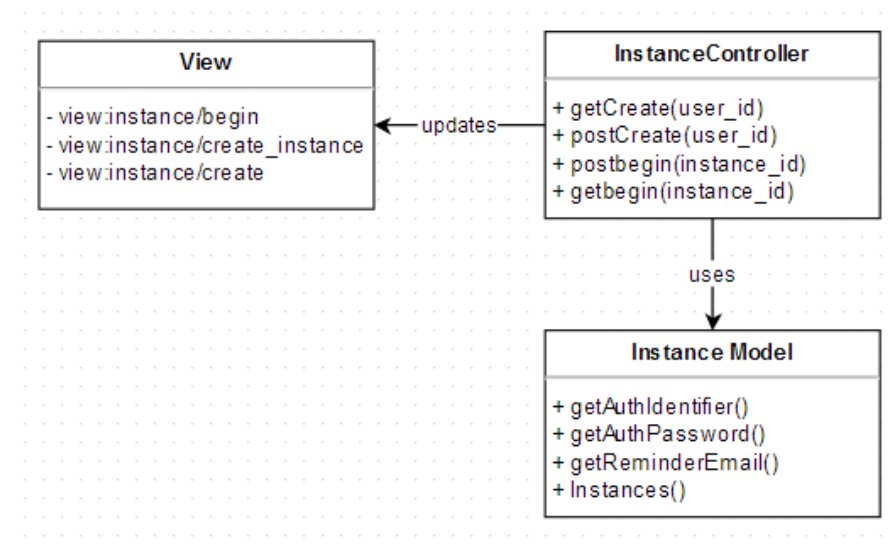


Figure 5.2.1.4 Instance Controller

## Web demo



performances of buildings. Coupling the building simulation environment with an advanced control system requires additional and specialised knowledge of both control development and building simulation software modelling. There is a gap between developers who build software controls for buildings (like home automation control systems) and building simulation software experts. The possibility to validate and benchmark sophisticated control algorithms on detailed building software models with a reduced modelling experience and knowledge could benefit building users, energy system designers and software engineers.

The objective is to develop a smart building test environment that will allow to compare and benchmark different control algorithms and store the results in a database a display comparison benchmark graphs.

Figure 5.2.1 Home page

## Create Account

- Username:
- Email:
- Password:
- Password Again:
- 

Figure 5.2.2 Create Account



Congratulations! You have successfully created your account! We have sent you an email

## Welcome to world of BCVTB!

Figure 5.2.3 Email activation(1)

Hello newWang,

Please activate your account using the following link.

---

<http://simapi.ucd.ie/accounts/activate/E6SssNuPbRO5xGbV6Jge6FmMOBLGS6769MAeMYofrq3bQtFUIR1SHo40urLOpENDBwqD9K>

---

Figure 5.2.4 Email activation(2)

Activated! You can now sign in!

## Welcome to world of BCVTB!

Figure 5.2.5 Activated

- Username:
- Password:
- ☐ Remember me
- 

Figure 5.2.6 Login

BCVTB API

You have been logged in!

# Welcome to world of BCVTB!

Figure 5.2.7 Logged in

BCVTB API

You have logged out

# Welcome to world of BCVTB!

Figure 5.2.8 Log out

Home

API Documentation

API Blueprint

- Old password:
- New Password:
- New Password again:
- 

Figure 5.2.9 Change password

- Old password:  The old password must be at least 6 characters.
- New Password:  The password must be at least 6 characters.
- New Password again:
- 

Figure 5.2.10 Change password failed example 1

- Old password:
- New Password:
- New Password again:  The password again and password must match.
- 

Figure 5.2.11 Change password failed example 2

BCVTB API

Home

API Documentation

API Blueprint

Email:

Figure 5.2.12 Forgot password



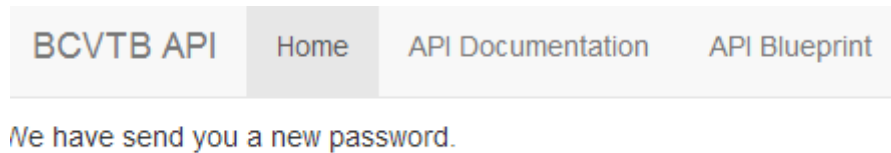


Figure 5.2.13 Recover password 1

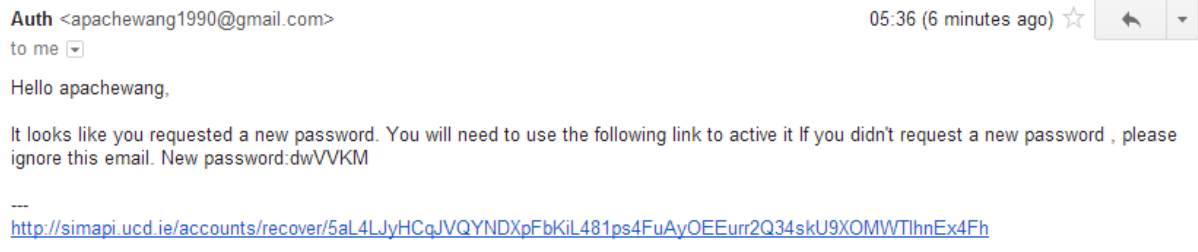


Figure 5.2.14 Recover password 2

Screen shots above show the basic process of creating a new account and activating account by the link sent to user's e-mail address. There are also functions of change password, forgot password and recover password by generating a new password and sending it to the user's email. Each input form has a validator to check whether input is validate or not. If input is not validate, an error message will appear following the form which has invalid input.

## Java Testing program

In the java testing program, I mainly test endpoints of creating a new account, creating a new instance, creating a new action, beginning simulation and getting sensor values. To test a API in java, HttpClient is powerful and easy to use.

### Create HttpClient

At the beginning, I set timestep to 0, create two strings user\_id, instance\_id which are set null by default and create a new DefaultHttpClient.

### Create a new account

Create a new string entity contains email, username and password and posts them to API using httpClient execute. After the execution, it can get the response with the help of HttpResponse. To read the response in json, BufferedReader and JSONObject is imported which makes it possible for me to get the user id of the new account so that I can use the user id to create a new instance belongs to the new account.

### Create a new instance

Create a new instance belongs to the account created above. Get and display the response from API.

### Create a new action

Set Heating temperature 10 and cooling temperate 26, wrap the values in json and post them to API. Get and

display the response from API.

## Begin Simulation

Set the selected instance begin value to 1. Get and display the response from API.

## Get Sensor

Get sensor is most tricky part of my java testing program in concern of synchronization. The main idea is to get all the sensor values of the new created instance by a loop using timestep.

At the beginning I set timestep to 1 .I also create a int parameter called wait status which tells the program whether it should wait for more data or go to an end.

Then if the API dose not throw a HTTP error, I check whether the output is null or not. If the output is null which means there are no new data inserted, I check whether begin is set to 2, in other words ,I check whether the simulation has come to an end. If so, output a message saying “ End of the simulation”.The reason why I do not check begin value first is that, it is possible that when BCVTB is faster than API even though begin is set to 2 there are still many rows that API have not read yet.

If there is no new data inserted and begin is still 1 , which means simulation is still running and could insert new rows in any time, I set wait\_status to 1 and display the message :”Waiting for new sensor value”.

When there is a new row found, output the new row and set wait\_status to 0. In the end, shutdown the httpClient.

## Output

```
Output from Server:

Create a new user completed
Details :
{"user_id":144}
-----
Create instance completed
Details:
{"instance_id":113}
-----
Create Action completed
Details:
{"Action_id":87}
-----
Ok Simulation starts
{"Instance id":113,"begin status":1}
-----
Waiting for new sensor value
```

Figure 5.3.7 Output (1)

When begin is set to 1, program start waiting for a new row .

```

Out put of sensor:
Time step <383>:

{"Sensor_id":"1","Timestep":"383","Instance_id":"116","Temperature_out":"24.7","Temperature_room":"25.99992960425817"}

Out put of sensor:
Time step <384>:

{"Sensor_id":"1","Timestep":"384","Instance_id":"116","Temperature_out":"24.55","Temperature_room":"25.999936166937204"}

Out put of sensor:
Time step <385>:

{"Sensor_id":"1","Timestep":"385","Instance_id":"116","Temperature_out":"24.4","Temperature_room":"25.999942605654"}

-----

End of simulation

```

Figure 5.3.7 Output (2)

When there is no new row and simulation is checked to be stoped, output “End of simulation”.

## API Evaluation

PHP\_Depend is a program that performs static code analysis on a given source base. It first takes the source code and parses it into an easily processable internal data structure. Then it takes the generated AST (Abstract Syntax Tree) and measures several values which is software metrics.

Since composer has been installed on the sever, installing PHP\_Depend could be as easy as adding a new dependency. After the installation, PHP depend is able to do the software metrics and draw charts as well.

```

[11211328@resit-web api]$ php pdepend.phar --summary-xml=/www/users/11211328/simapi/api/summary.xml \
> --jdepend-chart=/www/users/11211328/simapi/api/jdepend.svg \
> --overview-pyramid=/www/users/11211328/simapi/api/pyramid.svg \
> /www/users/11211328/simapi/api/app
PDepend 2.0.0

Parsing source files:
..... 60
. 61

Calculating Cyclomatic Complexity metrics:
..... 136

Calculating Node Loc metrics:
..... 102

Calculating NPath Complexity metrics:
..... 136

Calculating Inheritance metrics:
. 23

Calculating Node Count metrics:
... 80

Calculating Hierarchy metrics:
..... 114

Calculating Code Rank metrics:
. 26

Calculating Coupling metrics:
..... 142

Calculating Class Level metrics:
..... 129

Calculating Cohesion metrics:
..... 181

Calculating Dependency metrics:
.... 95

Generating pdepend log files, this may take a moment.
Time: 00:01; Memory: 15.50Mb

```

Figure 5.4 PHP depend calculating

A summary xml file as well as two charts are generated by PHP depend after calculation.

## Overview

Ⓐ ahh	2.4
Ⓐ andc	0.25925925925926
Ⓐ calls	153
Ⓐ ccn	76
Ⓐ ccn2	76
Ⓐ cloc	385
Ⓐ clsa	0
Ⓐ clsc	22
Ⓐ eloc	575
Ⓐ fanout	29
Ⓐ leafs	15
Ⓐ lloc	293
Ⓐ loc	1177
Ⓐ maxDIT	3
Ⓐ ncloc	792
Ⓐ noc	22
Ⓐ nof	0
Ⓐ noi	0
Ⓐ nom	57
Ⓐ nop	1
Ⓐ roots	5

Figure5.4.1.1 Overview

AHH:The Average Hierarchy Height metric is a average depth of the inheritance hierarchy. In a system of ten classes, a AHH-value of 1 can be interpreted in different ways, for example: Five classes inherit from five other classes within the analyzed application or five classes inherit from a single root class.

ANDC:The Average Number of Derived Classes metric describes the average of derived classes. In a system of ten classes an ANDC-value of 0.5 means, that every second class is derived from another class.

CALLS: Number of Method or Function Calls

CCN:Cyclomatic Complexity Number

CLOC:Comment Lines fo Code

CLSA: Number of Abstract Classes

CLSC: Number of Concrete Classes

ELOC:Executable Lines of Code

FANOUT: Number of Fanouts Referenced Classes

LEAFS: Number of Leaf Classes (final) classes

LLOC: Logical Lines Of Code

LOC:The Lines Of Code metric shows the number of executable source lines within the analyzed software system. To calculate this value PHP\_Depend counts all non white space lines and all non comment lines.

maxDIT:Max Depth of Inheritance Tree Maximum depth of inheritance

NCLOC:Non Comment Lines Of Code

NOC:The Number Of Classes metric counts the declared classes within the analyzed software system.

NOF: Number Of Functions

NOI: Number Of Interfaces

NOM:The Number Of Methods metric counts all declared methods, which in this context means class methods and simple functions.

NOP:The Number Of Packages metric counts the packages within the analyzed software system.

ROOTS: Number of Root Classes

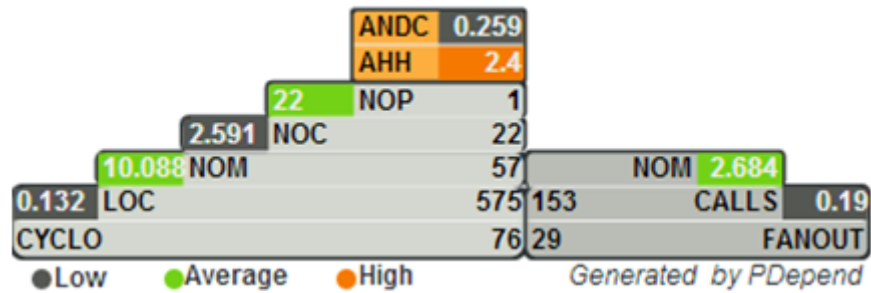


Figure 5.4.1.2 Overview Pyramid

The Overview Pyramid is used to visualize a complete software system in a really compact manner. Therefore it collects a set of metrics from the categories Inheritance, Coupling and Size & Complexity, and puts them into relation.

CYCLOC: The Cyclomatic Complexity number[5] is a software metric (measurement). It was already developed in 1976 by Thomas J. McCabe and is used to calculate the complexity of a program. It directly measures the number of linearly independent paths through a program's source code.

There are 575 lines of code, 57 methods, 22 classes in 1 package. The average depth of the inheritance hierarchy is considered to be high. The average of derived classes is low in this case.

## Abstraction Instability Chart

The A-I chart as it is generated by PHP\_Depend is based on the white paper OO Design Quality Metrics - An Analysis of Dependencies [6] published by Robert C. Martin. It shows you the quality of your design in the terms of extensibility, reusability and maintainability. All these facts are influenced by the inter-package dependencies and the package abstraction that PHP\_Depend visualizes in form of an abstract/instability chart.

Instability means the ratio between efferent coupling ( $C_e$ ) and the total package coupling ( $C_e + C_a$ ) which is based on the following formula ( $C_e / (C_e + C_a)$ ) and produces results in the range [0,1]. A value  $I=0$  indicates a maximally stable package that depends upon nothing and  $I=1$  indicates a total instable package that has no incoming dependencies but depends upon other packages.

Abstraction means the ratio between abstract classes ( $ac$ ) and the total of all classes ( $ac + cc$ ) that is calculated by this formula ( $ac / (ac + cc)$ ) that results in a value in the range [0,1].  $A=0$  means that all classes in this package are non-abstract while  $A=1$  shows a package that only consists of abstract classes and interfaces.

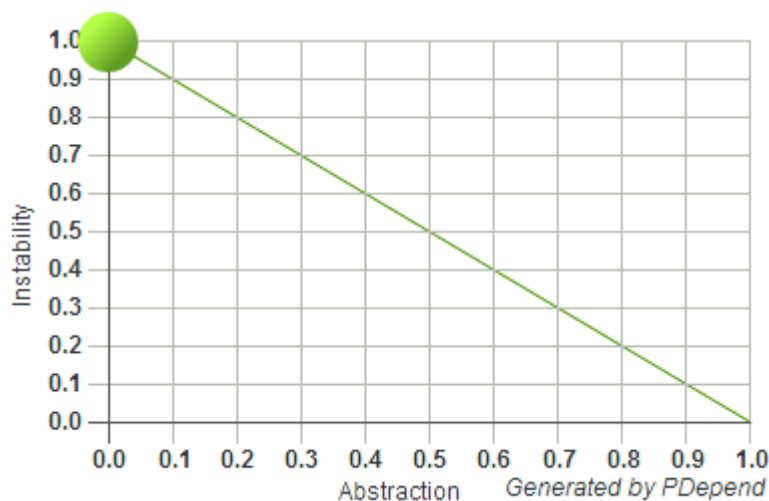


Figure 5.4.2 Abstraction Instability Chart

In terms of my case, there is only a single package. All classes in this package are non-abstract classes and interfaces. And it is a total instable package that has no incoming dependencies but depends upon other


packages.

## Monitor

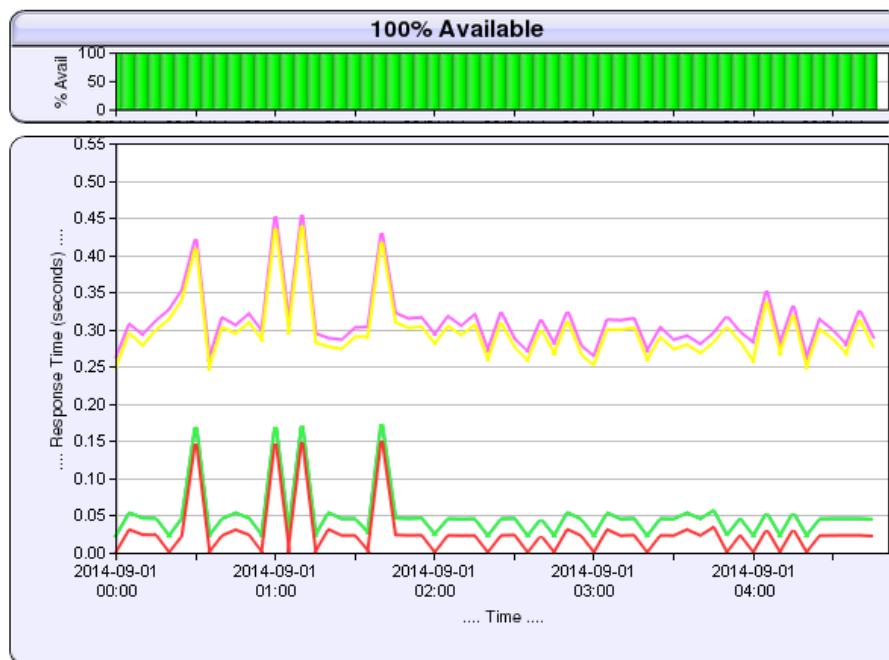
Use AlerSite to monitor the entry point of BCVTB API.



### Detail Report

Site Name:  <http://simapi.ucd.ie>  
Customer: ucd  
Locations: Birmingham

From: 2014-09-01 00:00  
Thru: 2014-09-01 04:51



Legend:

[Show Bar Chart](#)

■ DNS Lookup ■ Connect ■ Redirect ■ First Byte ■ Content

Transaction Load Analysis							
	<u>Response</u>	<u>DNS</u>	<u>Connect</u>	<u>Redirect</u>	<u>FirstByte</u>	<u>Content</u>	<u>Fullpage</u>
Avg	0.3103	0.0265	0.0226	0.0000	0.2481	0.0131	0.3918
Min	0.2593	0.0002	0.0222	0.0000	0.2214	0.0115	0.3249
Max	0.4538	0.1500	0.0248	0.0000	0.2934	0.0259	0.5352
STD	0.0409	0.0351	0.0005	0.0000	0.0179	0.0019	0.0420

Fullpage Time Distribution			
<u>Total Transaction Time</u>		<u>Individual Page Time</u>	
0 - 1 seconds	100.00%	0 - 1 seconds	100.00%
1 - 2 seconds	0.00%	1 - 2 seconds	0.00%
2 - 3 seconds	0.00%	2 - 3 seconds	0.00%
3 - 4 seconds	0.00%	3 - 4 seconds	0.00%
4 - 6 seconds	0.00%	4 - 6 seconds	0.00%
6 - 8 seconds	0.00%	6 - 8 seconds	0.00%
8 - 15 seconds	0.00%	8 - 15 seconds	0.00%
15 - 30 seconds	0.00%	15 - 30 seconds	0.00%
30.0 + seconds	0.00%	30 + seconds	0.00%

Status Summary		
<u>Status</u>	<u>Count</u>	<u>Description</u>
0	58	Site responded normally to all tests

Status Totals		
<u>Errors</u>	<u>Warnings</u>	<u>Notifications</u>
0	0	0

Figure 5.4.3 Monitor report

The average response time of the entry point is 310ms.

## Discussion on synchronization

During the test of combining BCVTB and API together , almost all the bugs we found have something to do with unexpected synchronization error. The methods might work fine either on BCVTB or in API testing , but only when put the two parts together did we find out that connecting two systems by database would lead us to serious errors if synchronization is not given enough consideration.

BCVTB usually requires a few seconds to warm up before running the simulation. That is to say when API wants to start the building simulation software , it has to wait a few seconds before checking whether the simulation started or not, or in others words are there some new values inserted or not. What is more, there might be a gap between two raw inserted into database. Which means when API finds out that there is no new data inserted , instead of returning to the user with message says it is the very end of the simulation , API should check whether the status of simulation has been set to end or not.

At the beginning, when we try to fix synchronization problem caused by software delay, we wanted to do the same in our program which is adding delay as well. But it does not take us a long time before we recognize that in order to add a reasonable delay we have to get the delay of the simulation software in the first place. Since it is hard to get a accurate delay of the simulation software, it is hard for us to build a reliable system.

The final solution for synchronization is that both BCVTB and API would keep on checking the database. If API finds outs that there is no new data inserted into database, it should wait for a new row as long as the status of simulation is running (begin column is still 1). At the same time, BCVTB keeps on running and inserts data into database when it gets new values of sensors. When there is a final log file generated , set the status of simulation to ended. As for API, only when there is new rows inserted in the database and status of simulation is set to ended should API generate a response telling the user that the simulation has been put to an end.

It is important to make full use of database as a bridge to synchronize the system properly. There is no doubt

that a well-designed and structured code would be more likely to get a better performance , but it is always better to do a test at early stage of developing. Otherwise , it would be a disaster trying to locate a single bug and judging whether it is a logic problem or a syntax one among the sea of thousands lines of code.