Create a website

Step 1 - Create the data model in Visual Paradigm

Use <http://www.visual-paradigm.com/product/vpuml/> to define your data model.

Create a class diagram.

For each object create a class

Define all the properties for each object as public

Add an ID column

Define the type of the fields. Valid types are int, stirng

Generate the php code and the database

You can use for connection string: server:localhost port:3066 databasetest user:root

Create object references:

Ex:

Echuipements belongs to Firma (each equipment is related to a specific firma)

Implys:

The relation is one to many

Echuipement table will have an Firma\_ID fiels to designate the parent object

Step 2 - Adjust database tables

Because: When generated the database tables, VisualParadigm had added field attribute as autoincrement, has not recognized data as valid data type.

Delete the “ID” fiels of all tables and rename all “Attribute” fields to “ID” One per each table.

Redefine the data type od all fields thah contain data into timestamp, with no dimmension. Use “current\_timestamp” for one data column in each table. This will be the creation timestamp.

Step 3 - Adjust PHP code

Split the code into DataModel, Controller and View.

In each DataModel file, only persistent fields will be used.

Each model file shuld include once all related model files

Ex. in Firma.php include once Equipment.php

For each foreign key, create an array in the referentiated object.

Ex:

class Equipment { $firma\_ID; }

Implys:

class Firma { $eqiupelents = Array(); }

Step 4 - Test your structures

Include model files in control files:

Ex. in Firma.php you have FirmaController class. Now, include once ..\\DataModel\\Firma.php that have Firma class.

NOTE: as each included file adopt the relative path of the file you include it into, adjust all relative paths in the included files.

Ex:

If Entity.php from DataModel use include(“.\\Entity.php”) and is included then in a Controller file, the relative path will pe interpreted as Controller.Entity.php instead of DataModel.Entity.php

Generate test code in the controllers:

$obj = new Firma();

echo (JSON-encode($obj));

Step 5 - Fill data structures from database

Create a select statement constructor for a given object.

Use the returned data to fill the structure and return the updated object

Use **get\_object\_vars** to fetch the fields of the given object.

Use **is\_array** function to exclude the related object collections (because they are not in the same table).

The ID variable can be readed from $\_GET. It is recommanded to use a constructor to initialize the ID field.

Get and Read methodes need to be implemented in each entity controller.

Read list all objects of the same type. This is an overview of the objects collection, not everithing.

Get retrieve all the information available for a selected object.

Step 6 - Define REST protocol

**READ**

Read request JSON example:

**{ “user” : { “ID” : 22, “token”: “hY3j” }, “data” : {“columns”: [ “ID”, “Name”, “Adress”, “Agent.Name” ], “filters” : [ “firma\_ID” : 5 ], “limit” : 10, “skip” : 0} }**

Read will give an array of rows. Each row is an array of values.

Read answare JSON example:

**{ “user” : { “ID” : 22, “token”: “hY3j” }, “data”:[ [ row1\_ID, row1\_value1, row1\_value2, ...], [row2\_ID, row2\_value1, row2\_value2, ... ], … ] }**

**GET**

Get request JSON example:

**{ “user” : { “ID” : 22, “token”: “hY3j” }, data : {“ID”: 5} }**

Get answare will give an object like

**{ “token”: “hY3j”, “data” : {“ID” : 5, “columnName1”: value1, “columnName1”: value1,... } }**

**SAVE**

SAVE request JSON example:

**{ “user” : { “ID” : 22, “token”: “hY3j” }, “data” : {“ID” : 5, “columnName1”: value1, “columnName1”: value1,... } }**  // An ID = -1 will force an insert, otherwise it is update

SAVE answare will give an object like

**{ “user” : { “ID” : 22, “token”: “hY3j” }, “data” :{ “ID” : 5 } , “Errors” : { “global” : {“ERROR” : “Global Message”}, “columnName1” : {“ERROR” : “Validation message” } } }**

**DELETE**

DELETE request JSON example:

**{ “user” : { “ID” : 22, “token”: “hY3j” }, “data” : {“ID” : 5, “options” : {“cascade” : true, “archive” : true } } }**

SAVE answare will give an object like

**{ “user” : { “ID” : 22, “token”: “hY3j” }, “data” :{ “ID” : 5 } , “Errors” : { “global” : {“ERROR” : “Global Message”}, “columnName1” : {“ERROR” : “Validation message” } } }**

Step 7 - Prepare View Static Pages

Prepare all fields of the pages, use same properties as in database if possible. Use related entity prefix if needen. Adjust element classes.

Step 8 - Fill View pages from JSON

Decode JSON string recieved through jQuery AJAX and populate HTML elements.

Use jQuery AJAX to

**var jqxhr = $.ajax({**

**type: "POST",**

**url: “entityController.php”,**

**data: data,**

**success: success,**

**dataType: dataType**

**})**

**.done(function(answare) { ... })**

**.fail(function(answare) { ... })**

**.always(function(answare) { ... });**

Use jQuery to update the view imput element values:

Ex:

**$(“#fieldID”).val(“newValue”);**

Set the content value of the input element defined by the ID = “fieldID” to “newValue”

**var myValue = $(“#fieldID”).val();**

Give to the variable myValue the value of the input element defined by the ID = “fieldID”

Data Flow:

URL -> Client -> JSON

JSON -> Server / Controller?param=1

**SQL HELPER**

**// reset visibility**

UPDATE `test`.`utilizator` SET `hidden` = '0';