PHP Web MVC Framework

A lightweight PHP framework for cooperative development of web applications.

Concepts

Web MVC Framework offers to developers a complete set of Classes for agile development of data intensive web applications.

Generally, it provides facilities for the system decomposition that developers can do at different levels during coding a complex web application.

- Firstly it implements services for realizing the **MVC architectural pattern**
- However, this is not the only feature provided by the Framework for making the application decomposition.

In general its goal is to organize and manage different views for decomposition of a web application

Architectural

Business scopes, roles and profiles

Content decomposition

Components and software reuse

Technologies requirements and skills

Architectural decomposition: MVC Pattern

Framework implements the **MVC pattern** allowing developers to decompose complex logic into different layers.

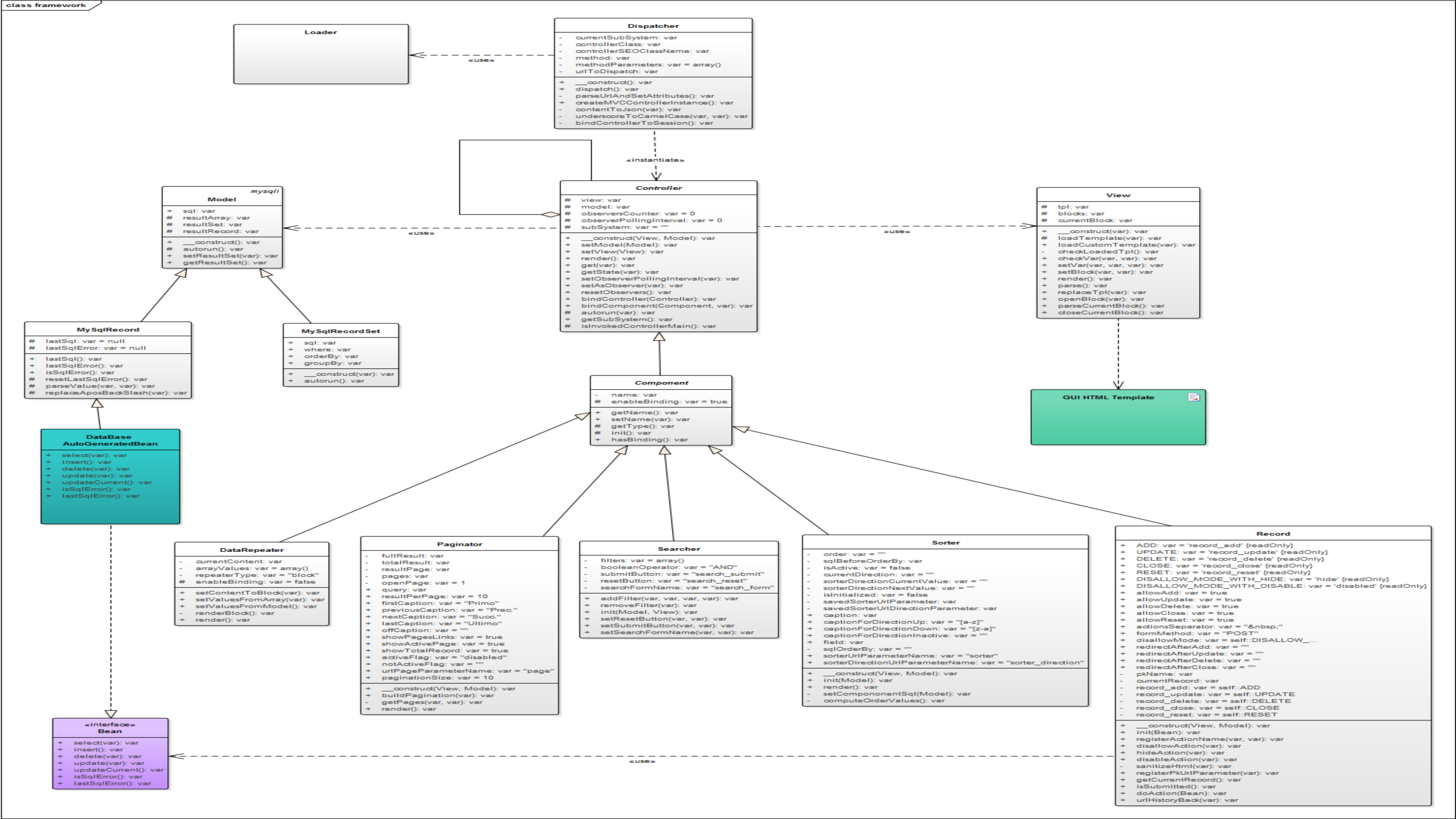
This is a specialization of the well know concept of the ***Separation of Concern,*** alias ***SOC,*** which thinks to an application like a big set **of computer functionalities (print, display, read/write data, control flow etc.)** intercommunicating each other's but having different system purposes.

SOC organizes an application by identifying its main purposes and then by grouping all its functions under these ones. MVC, which is a SOC implementation, classifies its purposes by using three high-level tiers: **data**, **presentation** and **application logic** by providing, respectively, **Model**, **View** and **Controller** (intercommunicating logic layers).

Therefore, by implementing a web application and by using the MVC pattern**, developers must decomposing and grouping its classes under this layers and must manage communications among them**.

Framework offers all base classes for building Model, View and Controller layers of a web application and for simplifying data communications among them.

Developers may quickly creates the application MVC layers just by extending Framework classes. Then the framework will provide the necessary services for the instantiation and intercommunication.


class framework
Loader
«use»
Dispatcher
- currentSubSystem: var
- controllerClass: var
- controllerSEOClassName: var
- method: var
- methodParameters: var = array()
- urlToDispatch: var
+ ___construct(): var
+ dispatch(): var
- parseUrlAndSetAttributes(): var
+ createMVCControllerInstance(): var
- contentToJson(var): var
- underscoreToCamelCase(var, var): var
- bindControllerToSession(): var
«instantiate»
mysqli
Model
+ sql: var
# resultArray: var
# resultSet: var
# resultRecord: var
+ ___construct(): var
# autorun(): var
+ setResultSet(var): var
+ getResultSet(): var
«use»
Controller
# view: var
# model: var
# observersCounter: var = 0
# observerPollingInterval: var = 0
# subSystem: var = ""
+ ___construct(View, Model): var
+ setModel(Model): var
+ setView(View): var
+ render(): var
+ get(var): var
+ getState(var): var
+ setObserverPollingInterval(var): var
+ setAsObserver(var): var
+ resetObservers(): var
+ bindController(Controller): var
+ bindComponent(Component, var): var
# autorun(var): var
+ getSubSystem(): var
# isInvokedControllerMain(): var
«use»
View
# tpl: var
# blocks: var
# currentBlock: var
+ ___construct(var): var
# loadTemplate(var): var
+ loadCustomTemplate(var): var
- checkLoadedTpl(): var
+ checkVar(var, var): var
+ setVar(var, var, var): var
+ setBlock(var, var): var
+ render(): var
+ parse(): var
+ replaceTpl(var): var
+ openBlock(var): var
+ parseCurrentBlock(): var
+ closeCurrentBlock(): var
GUI HTML Template
MySqlRecord
# lastSql: var = null
# lastSqlError: var = null
+ lastSql(): var
+ lastSqlError(): var
+ isSqlError(): var
# resetLastSqlError(): var
# parseValue(var, var): var
# replaceAposBackSlash(var): var
MySqlRecordSet
+ sql: var
+ where: var
+ orderBy: var
+ groupBy: var
+ ___construct(var): var
+ autorun(): var
Component
- name: var
# enableBinding: var = true
+ getName(): var
+ setName(var): var
# getType(): var
# init(): var
+ hasBinding(): var
DataBase
AutoGeneratedBean
+ select(var): var
+ insert(): var
+ delete(var): var
+ update(var): var
+ updateCurrent(): var
+ isSqlError(): var
+ lastSqlError(): var
DataRepeater
- currentContent: var
- arrayValues: var = array()
- repeaterType: var = "block"
# enableBinding: var = false
+ setContentToBlock(var): var
+ setValuesFromArray(var): var
+ setValuesFromModel(): var
- renderBlock(): var
+ render(): var
Paginator
- fullResult: var
- totalResult: var
- resultPage: var
- pages: var
- openPage: var = 1
+ query: var
+ resultPerPage: var = 10
+ firstCaption: var = "Primo"
+ previousCaption: var = "Prec."
+ nextCaption: var = "Succ."
+ lastCaption: var = "Ultimo"
+ offCaption: var = ""
+ showPagesLinks: var = true
+ showActivePage: var = true
+ showTotalRecord: var = true
+ activeFlag: var = "disabled"
+ notActiveFlag: var = ""
+ urlPageParameterName: var = "page"
+ paginationSize: var = 10
+ ___construct(View, Model): var
+ buildPagination(var): var
- getPages(var, var): var
+ render(): var
Searcher
- filters: var = array()
- booleanOperator: var = "AND"
- submitButton: var = "search_submit"
- resetButton: var = "search_reset"
- searchFormName: var = "search_form"
+ addFilter(var, var, var, var): var
+ removeFilter(var): var
+ init(Model, View): var
+ setResetButton(var, var): var
+ setSubmitButton(var, var): var
+ setSearchFormName(var, var): var
Sorter
- order: var = ""
- sqlBeforeOrderBy: var
- isActive: var = false
- currentDirection: var = ""
- sorterDirectionCurrentValue: var = ""
- sorterDirectionNextValue: var = ""
- isInitialized: var = false
- savedSorterUrlParameter: var
- savedSorterUrlDirectionParameter: var
+ caption: var
+ captionForDirectionUp: var = "[a-z]"
+ captionForDirectionDown: var = "[z-a]"
+ captionForDirectionInactive: var = ""
+ field: var
- sqlOrderBy: var = ""
+ sorterUrlParameterName: var = "sorter"
+ sorterDirectionUrlParameterName: var = "sorter_direction"
+ ___construct(View, Model): var
+ init(Model): var
+ render(): var
- setCompononentSql(Model): var
- computeOrderValues(): var
Record
+ ADD: var = 'record_add' {readOnly}
+ UPDATE: var = 'record_update' {readOnly}
+ DELETE: var = 'record_delete' {readOnly}
+ CLOSE: var = 'record_close' {readOnly}
+ RESET: var = 'record_reset' {readOnly}
+ DISALLOW_MODE_WITH_HIDE: var = 'hide' {readOnly}
+ DISALLOW_MODE_WITH_DISABLE: var = 'disabled' {readOnly}
+ allowAdd: var = true
+ allowUpdate: var = true
+ allowDelete: var = true
+ allowClose: var = true
+ allowReset: var = true
+ actionsSeparator: var = "&nbsp;"
+ formMethod: var = "POST"
+ disallowMode: var = self::DISALLOW_...
+ redirectAfterAdd: var = ""
+ redirectAfterUpdate: var = ""
+ redirectAfterDelete: var = ""
+ redirectAfterClose: var = ""
- pkName: var
- currentRecord: var
- record_add: var = self::ADD
- record_update: var = self::UPDATE
- record_delete: var = self::DELETE
- record_close: var = self::CLOSE
- record_reset: var = self::RESET
+ ___construct(View, Model): var
+ init(Bean): var
+ registerActionName(var, var): var
+ disallowAction(var): var
+ hideAction(var): var
+ disableAction(var): var
- sanitizeHtml(var): var
+ registerPkUrlParameter(var): var
+ getCurrentRecord(): var
+ isSubmitted(): var
+ doAction(Bean): var
+ urlHistoryBack(var): var
«interface»
Bean
+ select(var): var
+ insert(): var
+ delete(var): var
+ update(var): var
+ updateCurrent(): var
+ isSqlError(): var
+ lastSqlError(): var
«use»

Convention over configuration

Framework adopts a **Convention over Configuration** paradigm (also known as coding by convention) that is a software design paradigm that attempt to decrease the number of decisions that a developer, by using the framework, is required to make without necessarily losing flexibility.

Its conventions automatically provide a **global mechanism for handling HTTP requests** and for **routing**, **dispatching** them **to controllers (response providers).**

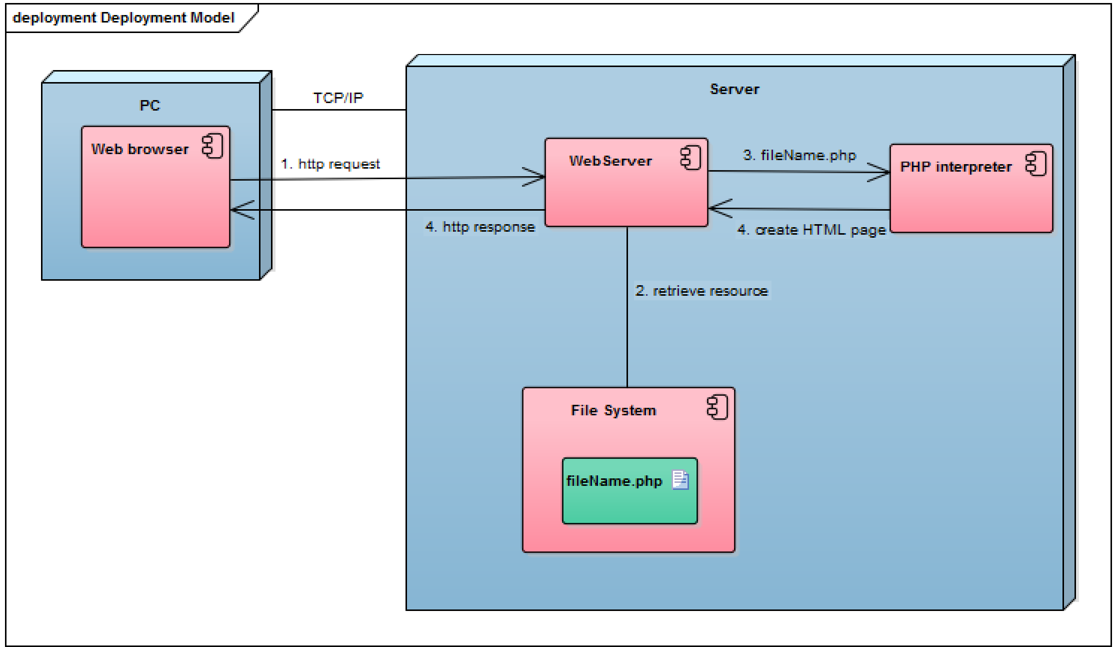
Framework conventions, in facts, are able to instantiate a user Controller without extra configurations requirements and **also permit classic PHP development approach**, where developers can write code into a single file by using both OOP or Functional paradigm.

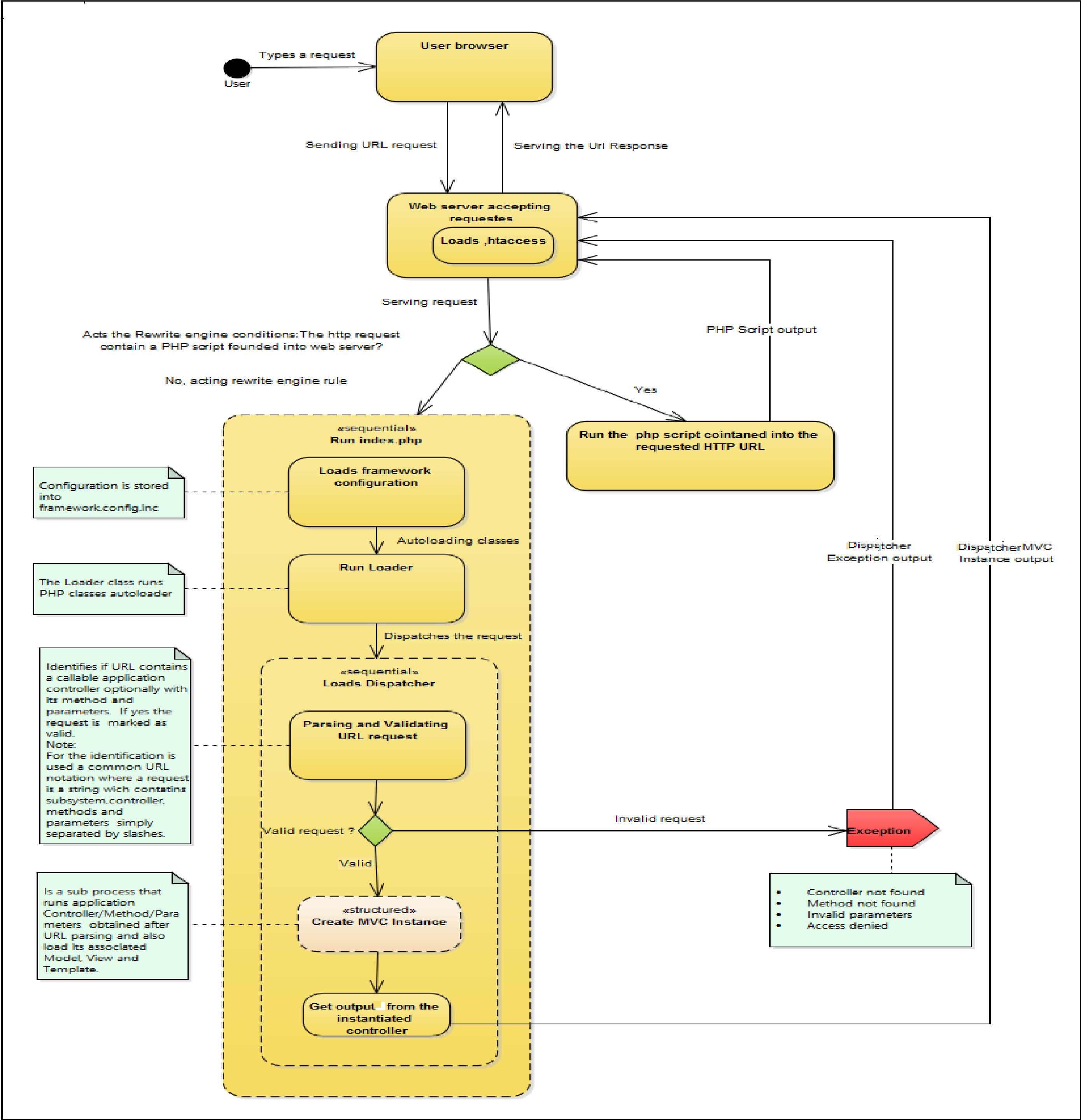
This aspect facilitate the integration of the Framework together with any existing application that use a different development technique.

Routing, dispatching and auto loading

Framework provides a mechanism for routing and dispatching user to an appropriate controller.

Classes' auto loader also simplify the objects creation by eliminating the needs of files inclusions (PHP require) or files dependencies.




User
Types a request
User browser
Sending URL request
Serving the Url Response
Web server accepting requestes
Loads .htaccess
Serving request
Acts the Rewrite engine conditions:The http request contain a PHP script founded into web server?
No, acting rewrite engine rule
Yes
Run the php script cointaned into the requested HTTP URL
PHP Script output
«sequential»
Run index.php
Loads framework configuration
Autoloading classes
Run Loader
Dispatches the request
«sequential»
Loads Dispatcher
Parsing and Validating URL request
Valid request ?
Valid
Invalid request
Exception
«structured»
Create MVC Instance
Get output from the instantiated controller
Dispatcher Exception output
Dispatcher MVC Instance output

Configuration is stored into framework.config.inc

The Loader class runs PHP classes autoloader

Identifies if URL contains a callable application controller optionally with its method and parameters. If yes the request is marked as valid.
Note:
For the identification is used a common URL notation where a request is a string wich contatins subsystem,controller, methods and parameters simply separated by slashes.

Is a sub process that runs application Controller/Method/Parameters obtained after URL parsing and also load its associated Model, View and Template.

- Controller not found
- Method not found
- Invalid parameters
- Access denied

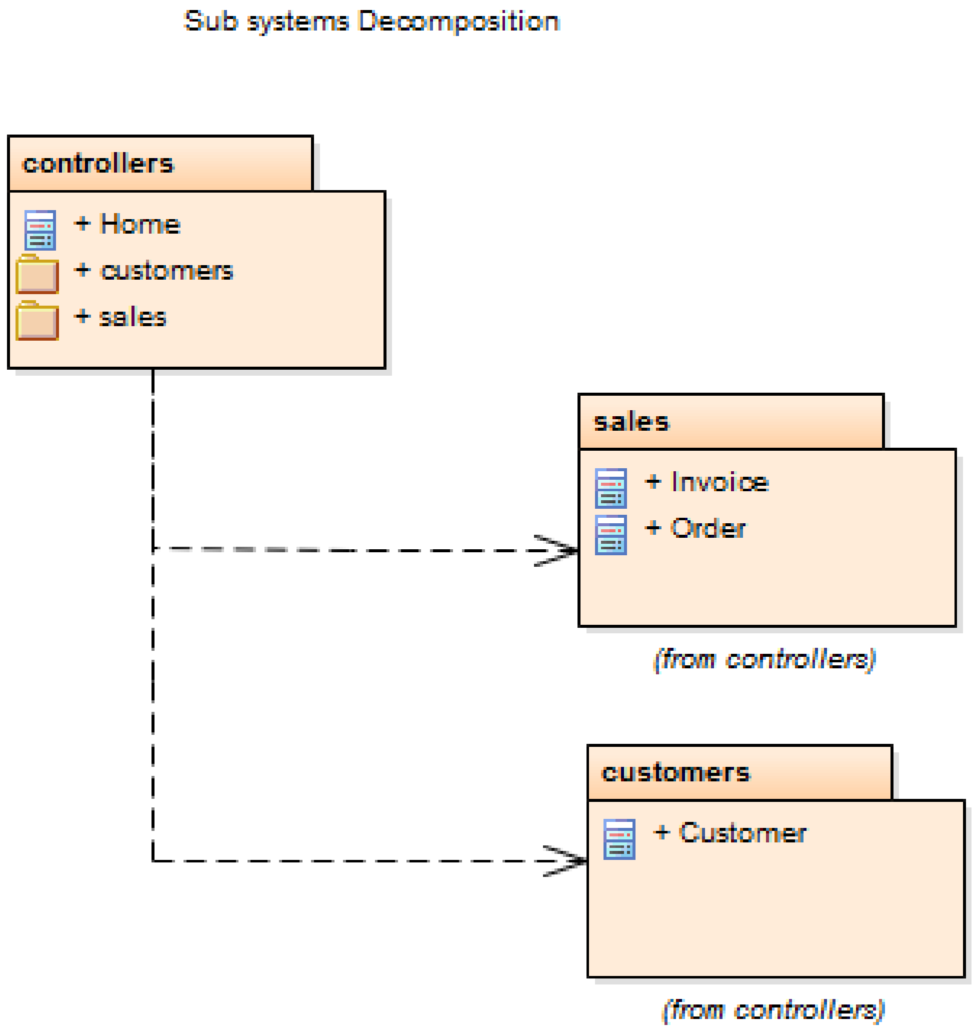
Business decomposition: Namespaces, Sub Systems and Classes

Framework, depending from application's complexity, lets developers to organize and decompose application's classes by using namespaces and sub systems.

Sub systems allows designing the application's architecture with a high level model of decomposition, well known as **Systems Decomposition**.

System Decomposition **looks at business** logic decomposition of an application.

It differs from the MVC decomposition, which is principally oriented to the computer functions concern like, for example, file system functions, memory management, database, GUI functions and so on.



Logical and physical mappings

Frameworks uses a case sensitive one to one convention for mapping:

namespace -> sub system A Namespace named *XYZ* identifies a Sub System named *XYZ* with its own variables scope

sub system ->web server folder A Sub System named *XYZ* is placed into a folder with the same name

class name -> file A Class name *MyClass* is placed into a file *MyClass.php* which is located into a sub system folder

For example:

- **namespace controllers\MySubSystem** -> **web_root/controllers/MySubSystem**
- **use controllers\MySubSystem \MyClass** -> **web_root/controllers/MySubSystem/MyClass.php**

Decomposition and separation between front end and back end technologies

Templating functionalities included into the Framework permits to decuple PHP source code used for the presentation logic from HTML/CSS source code needed for the GUI design. Both pieces of code are under the responsibility of the View layer and must coexist inside it.

By using the Framework, will be possible to avoid the mixture of different programming languages when implementing the View.

Developers can build a View by putting the GUI static design into an external and standard HTML file, which acts as a template, optionally containing some special elements: **placeholders** and **blocks**.

A placeholder is just a simple string between braces, for example, **{Placeholder}** while a block is common HTML code between special HTMLs comments like:

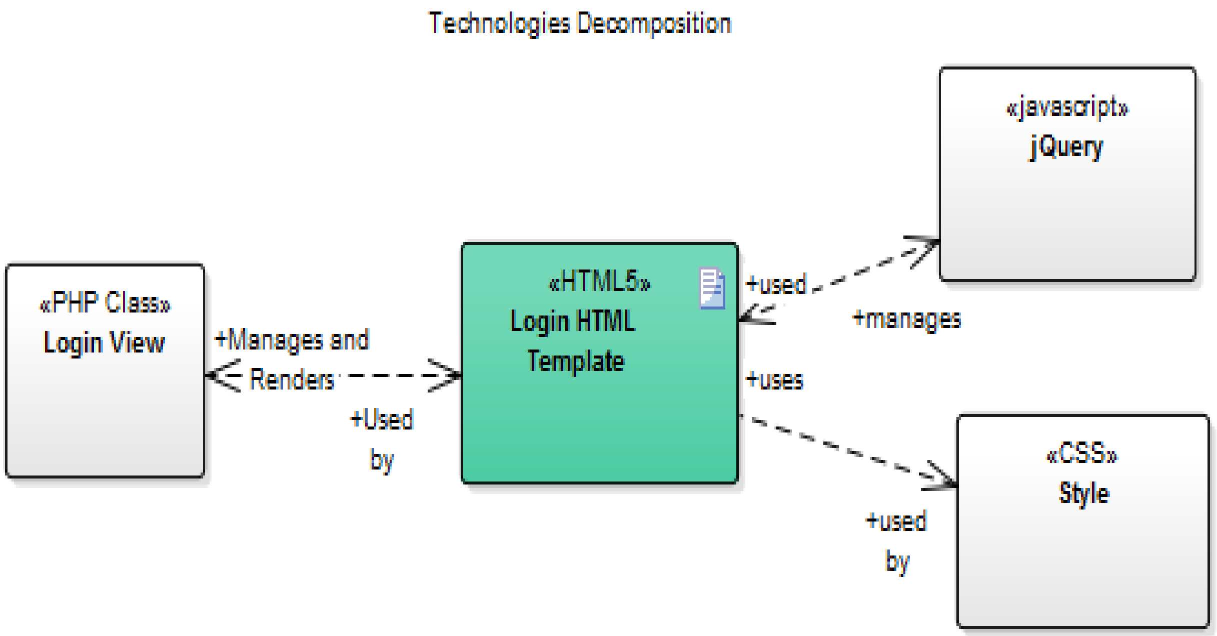
<!-- BEGIN block --> custom HTML **<!-- END block -->**

Templating

MVC pattern in conjunction with the Template Engine of the Frameworks offers another level of application decomposition oriented to **Decupling and Separation of Technologies.**

It gives to PHP View functionalities for rendering any data provided from external sources, like a database, into a GUI static design built into an external HTML file.

It avoids the mixing of programming languages into a single source code file and permits to the designers to build the application design by using only client side technologies and to developers to builds pure PHP application without the needs of mixing each other their respective artifacts.



Roles separation

Technically speaking this means that a source code file written by a developer to implement a specific layer of a MVC instance will use only one programming language or side-technology.

Framework avoids the contemporary mixture of server-side technologies such as PHP, together with client-side ones, such as HTML, during the writing of a single source code file.

Therefore, each files of an end user application will result to using only a single side technology and framework will take care of communications among files, written in different programming languages, by providing a set of server side components, classes and methods.

Entità	Finalità	Tecnologia	Gruppo di lavoro/Skill
View	• UI; • Cattura input; • Mostra output; • Gestisce l'interazione dell'utente con la UI.	XHTML per la UI. Può contenere riferimenti a: • Fogli di Stile CSS • Script in linguaggio JavaScript	Web designer con competenze primarie in grafica web, UI (Skill: HTML, CSS, JAVASCRIPT)
Controller	• Mediazione tra View e Model; • Trattamento Input/Output; • Logica applicativa.	PHP	Analisti/Programmatori (Skill: PHP)
Model	• Interfacciamento alle fonti di Dati; • Rappresentazione delle strutture e della logica di Business.	PHP, SQL	Esperti del dominio di applicazione, progettisti, ing. soft. (Skill: PHP, DB)

Hierarchical content decomposition

Framework permits developers to nesting controllers by enforcing a **Hierarchical Content Decomposition** concept.

For example, developers can build a complex application’s page by decomposing it into multiple sections implemented by different child controllers and nested into a single root controller representative of the HTML page.

Each of child controllers is, potentially, a runnable stand-alone MVC instance and, optionally, developers can reuse it inside a different root controller.

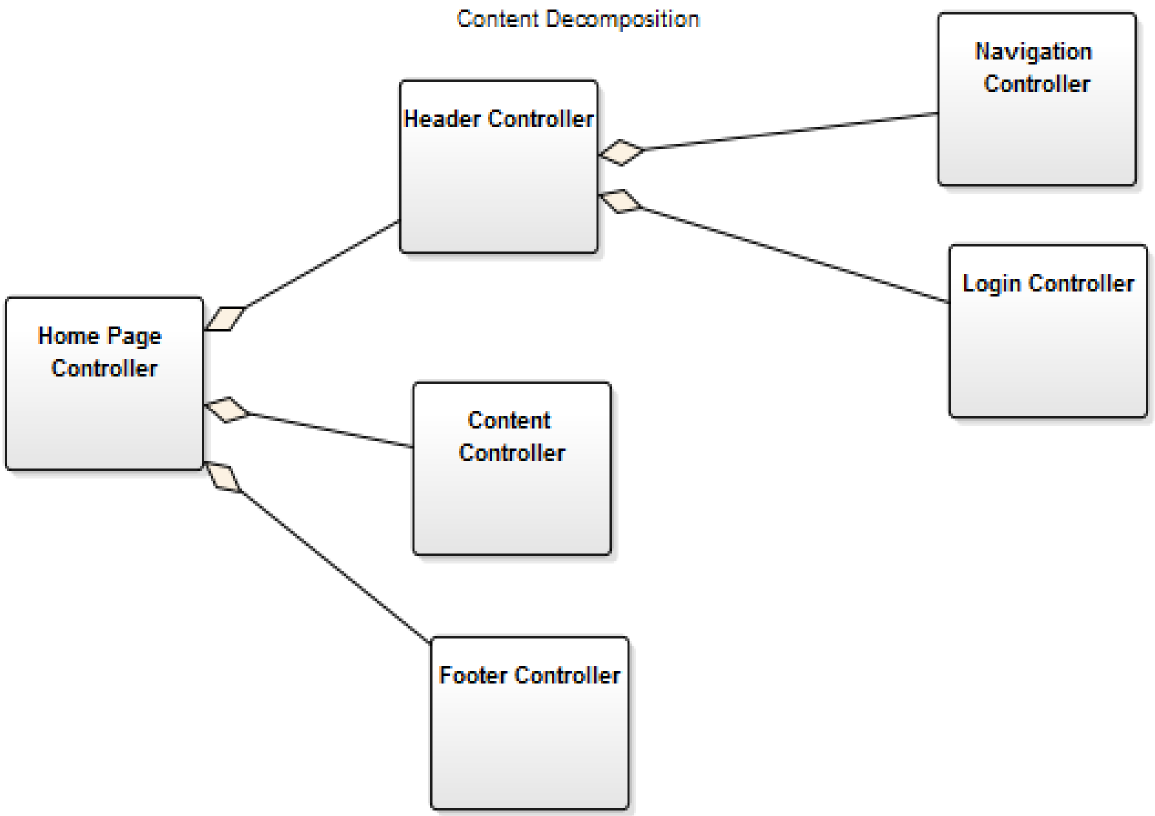
By requesting the execution of the root controller, Framework will render it automatically together with all its child and nested controllers.

There is no limit for the nesting level of controllers into the hierarchy. It exclusively depends from the application’s need or from a good decomposition analysis of the application’s scenarios.

A hierarchical MVC use case

A typical example of use (and reuse) of MVC hierarchy is an ecommerce application where common section, like the toolbar, page footer and site navigation, must be present into different application's pages like browse products or product's detail.

Developers have the facilities to build and test those child controllers individually. Then they may compose the root scenarios by assembling them into controller classes designed as root. Framework offers facilities for nesting controllers in a very simple way. In fact, developers can build a controllers' hierarchy simply by putting a special placeholder for its child controller into the View layer of the root controller. Framework will do the rest.



Component based decomposition and reuse

The **Component Based Development,** used for building many Framework's features, permits to developers another more level of applications decomposition and software reuse.

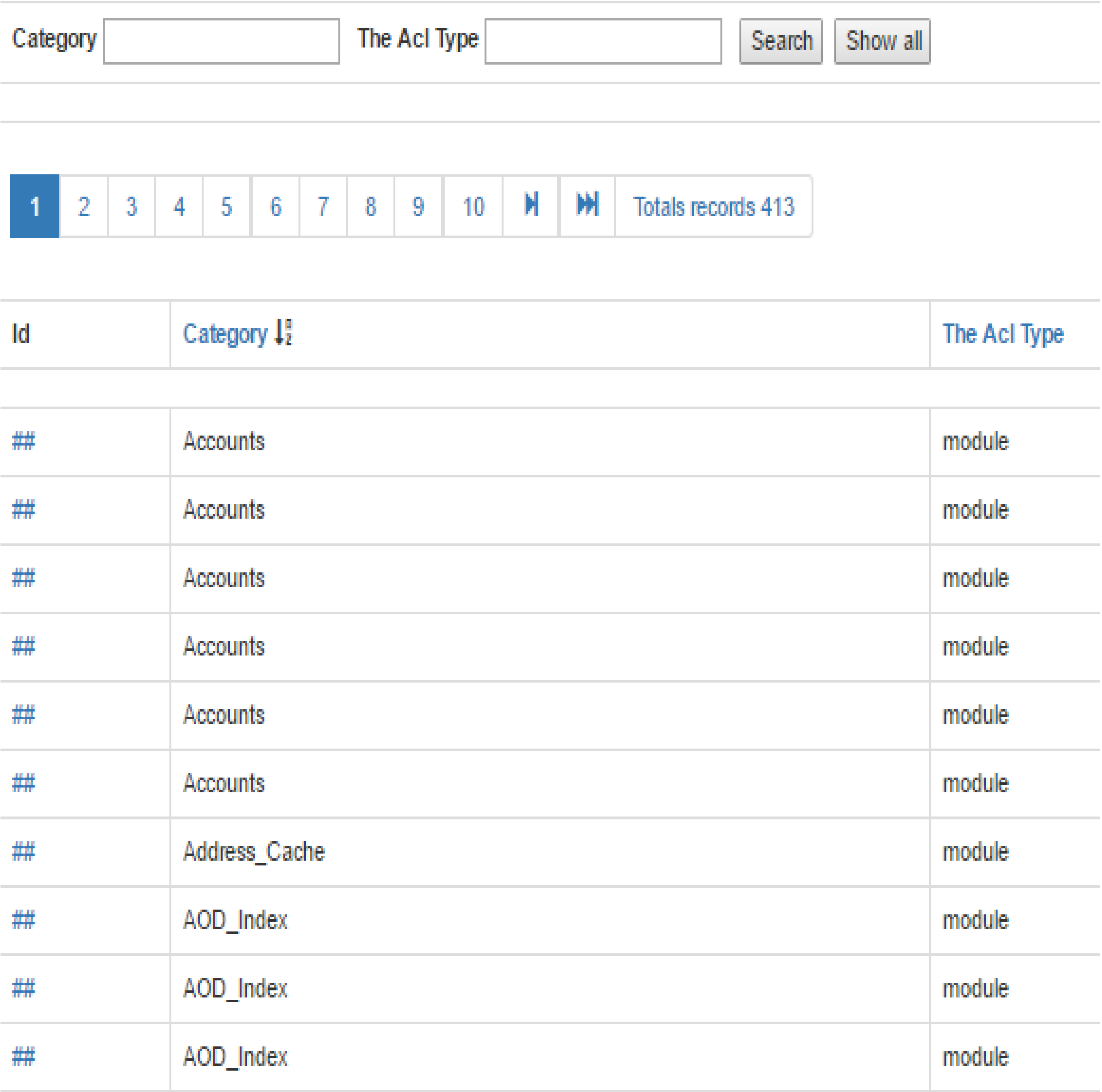
Framework's components, in fact, realizes common **Aspects** that can occurs, in a similar way, into different web applications.

Many of these aspects are **regarding database**, for example: data listing, data listing and sorting, data listing and filtering, data listing and pagination, record management and common table's operations for select, insert, delete and update records.

Framework offers a set of pre-built components for implementing the necessary server logic for these common database management aspects.

These components are itself MVC objects with a Controller, are easy to use and developers can aggregate them into a root controller by using a composite criteria for building complex application pages.

A component GUI can also easily adapted or replaced to reflect the application's experience simply by modifying or replacing its HTML template with a custom one. Component's server logic will remain fully reusable without the need of any source code modifications.

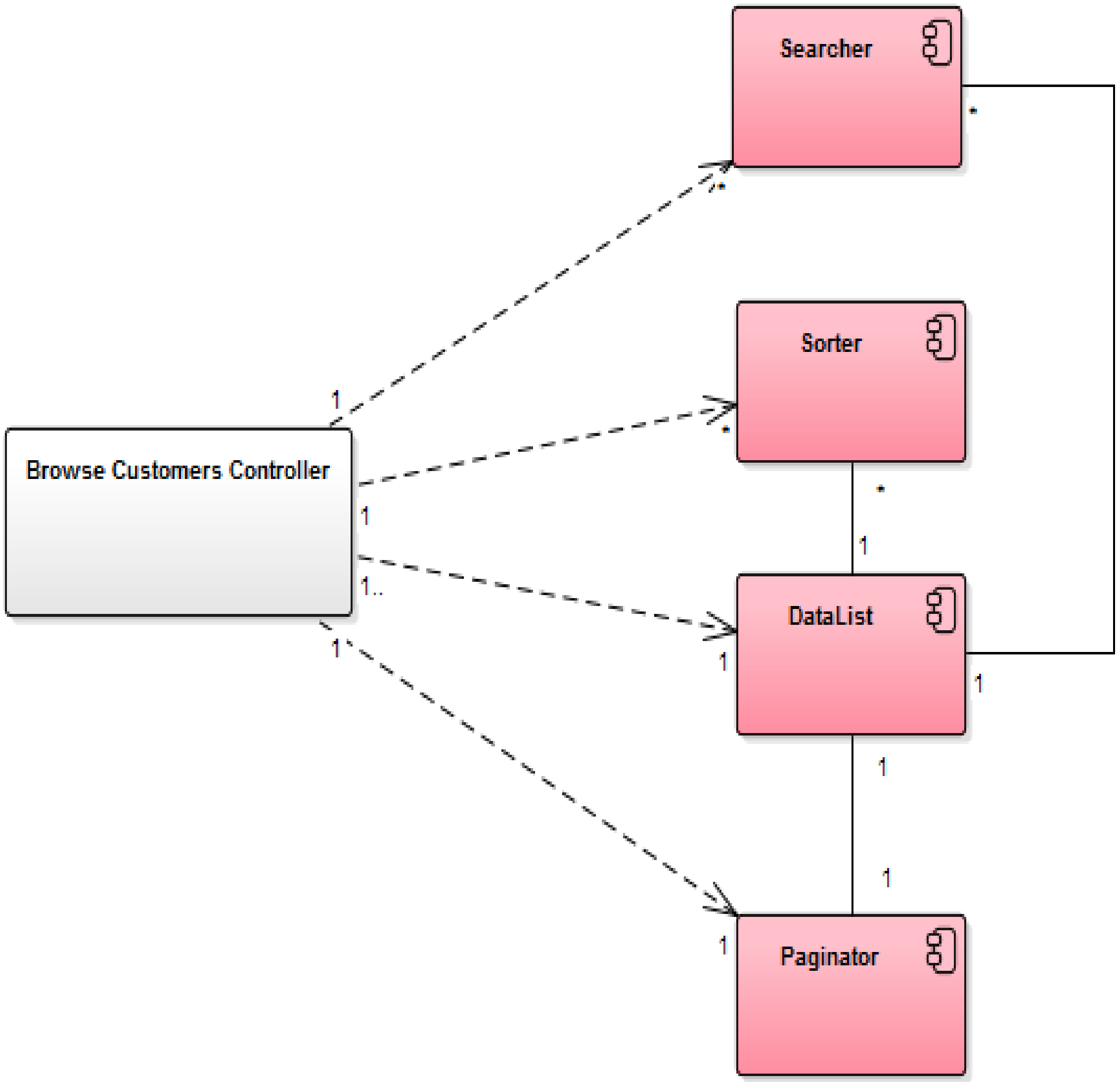


Category [ ] The Acl Type [ ] Search Show all

1 2 3 4 5 6 7 8 9 10 ⏭ ⏭⏭ Totals records 413

Id	Category	The Acl Type
##	Accounts	module
##	Accounts	module
##	Accounts	module
##	Accounts	module
##	Accounts	module
##	Accounts	module
##	Address_Cache	module
##	AOD_Index	module
##	AOD_Index	module
##	AOD_Index	module

Component Based Development and Decomposition



Observation of content changing

A common limitation for web application is its intrinsic **HTTP state less communication between server and client**.

One consequence of a state less application is the inability to update automatically a content when it change and the change is acted outside the application session (for example by another user that access to a shared database table).

Although the MVC design pattern well describe the updating on content change among its layers, this aspect is hard to implement into a state less application like the web applications.

Desktop applications are not affected by this difficulty.

Fortunately, for web application, **AJAX can provides a valid polling solution** to verify the content changing and more, the modern **HTML Web Socket** capabilities surely resolve this HTTP limitation.

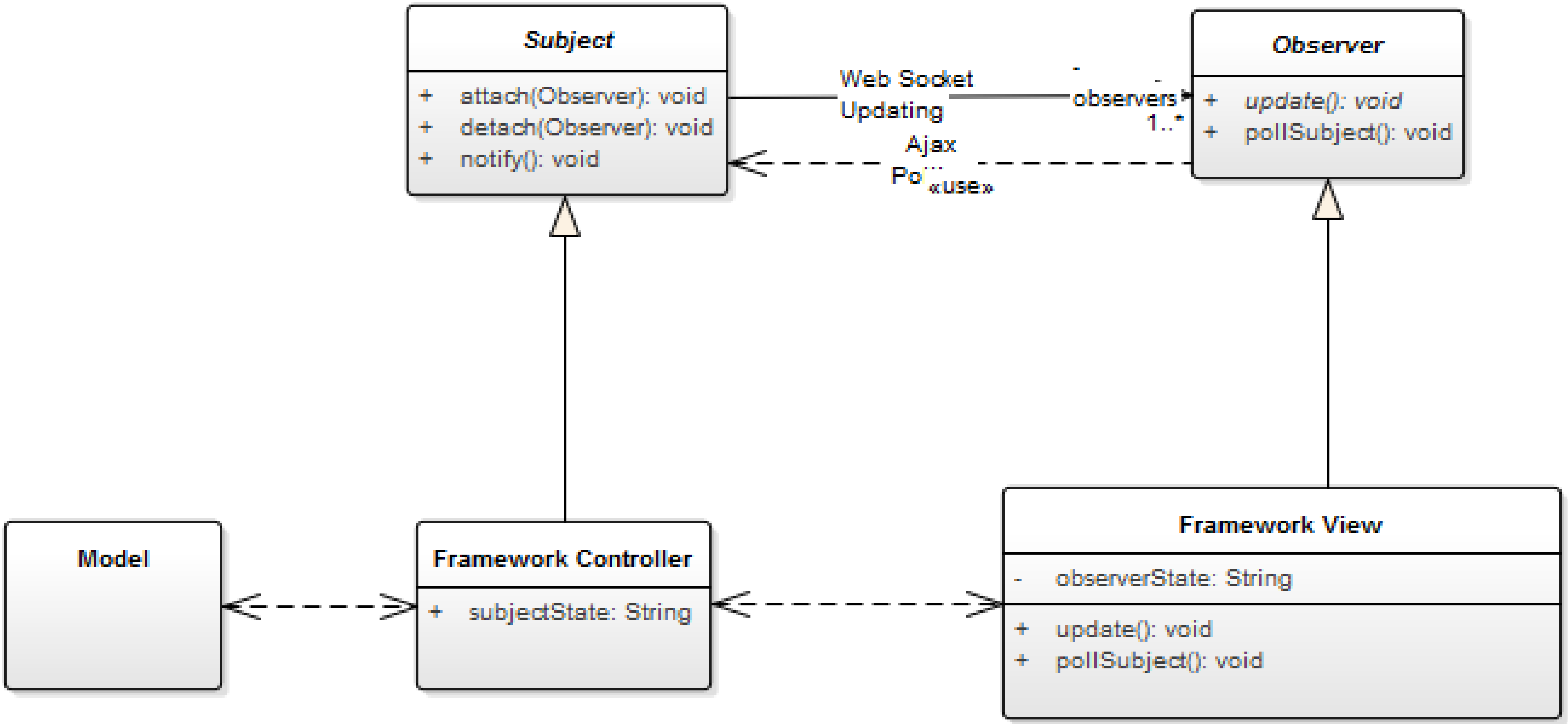
Of course, developers are able to implements their own AJAX or Web Socket hand coded solutions.

Alternately, they can quickly use **a special Framework** feature that can do the job automatically without the need of writing any type of custom code.

Controller base class was built by keeping in mind this protocol limitation.

A special Controller method, **setAsObserver**, realizes the **Observation of Content Changing** by injecting, automatically, all necessary AJAX code into the template file of the View to do the job.

Observing Changes



Thanks for your attention - Grazie per l’attenzione

Rosario Carvello

rosario.carvello@gmail.com