

Università degli studi di Modena e Reggio Emilia

DIPARTIMENTO DI INGEGNERIA "ENZO FERRARI"

Laurea Triennale in Ingegneria Informatica

Confronto tra generazioni di tecnologie Web mediante benchmarking

Relatore:

Prof. Riccardo Lancellotti

Candidato:

Marco Cipriano

Anno accademico 2016/2017

Sommario

1. Motivazioni dietro il benchmarking	1
2. RUBiS	1
3. Software	2
La struttura di RUBiS	2
Architettura dell'emulatore di RUBiS.....	5
Le versioni utilizzate in questo studio	6
Il framework Laravel	10
Altro software utilizzato.....	15
4. Hardware.....	16
5. Svolgimento dell'esperimento	16
Installazione	16
Client	16
Server	17
Considerazioni preliminari sull'esperimento	18
Le tre fasi: RumpUp, RunTime, RumpDown	19
Esperimento su piccolo range: 1-40 Clients.....	19
Analisi del throughput.....	22
Esperimento su grosso range: 1-800 Clients.....	24
Analisi del throughput.....	29
Analisi degli errori del client	30
Il caching di Laravel	32
6. Oltre le prestazioni: analisi del codice sorgente	34
Output delle pagine	34
Accessi al DMBS	41
Gestione degli errori	43
7. Conclusioni	45
Bibliografia	46

1. Motivazioni dietro il benchmarking

Negli ultimi decenni si è assistito alla crescita esponenziale di internet e con esso anche a quella delle tecnologie legate alla rete. In tale contesto il benchmarking legato al web ha assunto un ruolo sempre più fondamentale per definire quali metodologie siano preferibili rispetto ad altre.

Grazie a questa continua crescita, nel corso degli anni si è assistito ad un progressivo abbattimento dei costi delle infrastrutture hardware che ha dato la possibilità agli sviluppatori di focalizzarsi su obiettivi via via sempre meno legati unicamente all'efficienza del codice e più vicini anche alla sua leggibilità e manutenibilità.

Risulta dunque interessante analizzare empiricamente, attraverso benchmark prestazionali ed analisi del codice sorgente, i vari metodi di sviluppo web che si sono succeduti negli anni, contestualizzandoli e dandone un giudizio qualitativo: questo rappresenta l'obiettivo del nostro studio.

2. RUBiS

RUBiS, acronimo per Rice University Bidding System, è un prototipo di un sito d'aste costruito sul modello di Ebay nel lontano 2001. Il sito ha diverse funzioni quali ad esempio la ricerca di oggetti per categoria o regione, il piazzamento di puntate, la messa in vendita di articoli, la registrazione di nuovi utenti e la possibilità di lasciare commenti.

RUBiS è stato creato proprio per effettuare delle analisi prestazionali su questo genere di operazioni. Del sito si sono dunque accumulate nel tempo diverse implementazioni con differenti linguaggi e design pattern.

RUBiS è dotato di un tool, scritto in Java, che permette di effettuare analisi prestazionali del sito emulando il comportamento di un numero definito di utenti, distinguendo ciascuno di essi tramite una sessione. L'arco di tempo di una sessione si divide in 3 fasi: rumpUp, Runtime, RumpDown. Dell'architettura dell'emulatore e del sito ci occuperemo più nel dettaglio nel prossimo capitolo, per adesso ci basti dire che se il sito è costruito in maniera tale da rispettare gli standard dell'emulatore, RUBiS fornirà per ogni pagina del sito una serie di statistiche prestazionali quali: la percentuale di accessi degli utenti, il numero di accessi, il numero di errori, il tempo di risposta massimo, minimo e medio del server. Tutti questi valori sono valutati per le tre fasi e per la media totale tra le tre.

Per la varietà dei dati statistici che fornisce e per la sua storica importanza a livello di benchmarking web abbiamo scelto RUBiS per gli scopi del nostro studio.

3. Software

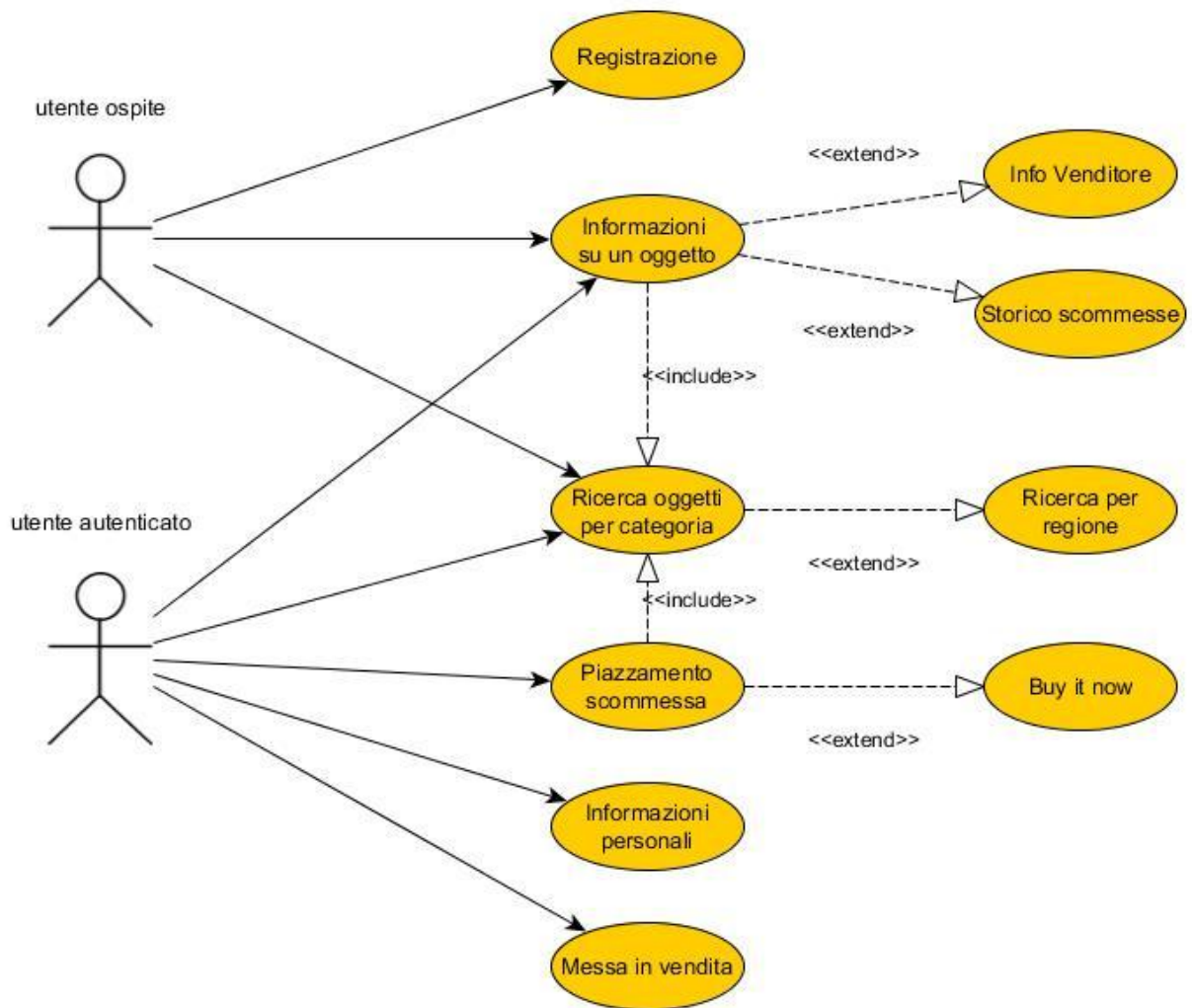
La struttura di RUBiS

Originariamente RUBiS è stato scritto con differenti design pattern prevalentemente legati al mondo java: JBoss, JOnAS, Servlets [1]. In questo studio abbiamo invece preso spunto da una vecchia versione di RUBiS scritta in PHP4 e, pur mantenendo intatto l'output delle pagine, siamo andati a riscrivere quasi completamente il codice che vi sta dietro: mantenendo chiaramente le caratteristiche funzionali del sito.

Il sito RUBiS prevede fino a 26 tipi di operazioni che possono essere eseguite tramite chiamate HTTP a determinati URL sia per mezzo del metodo POST (tipicamente quello che succede quando si visita il sito tramite browser) che di quello GET (il metodo che usa l'emulatore per le sue richieste).

per provare a dare un'idea generica di quelle che sono le azioni alle quali un utente (autenticato e non) può accedere, abbiamo creato uno Use Case Diagram che è presente nella pagina successiva.

Si nota come le azioni disponibili per un utente ospite siano limitate alla creazione di un profilo e alla ricerca di oggetti ed informazioni su di essi. Non è presente un blocco che evidenzia la transizione tra le due tipologie di utenti poiché il login in RUBiS deve essere eseguito ogni volta che si prova ad accedere ad operazioni riservate: Il sito web non fa uso di sessioni o cookies per tenere traccia delle autenticazioni.



Andiamo adesso più nel dettaglio.

Quanto segue è una tabella che riporta tutte le pagine PHP presenti in RUBiS. Per ciascuna pagina è presente la descrizione dei suoi obiettivi. A seguire vi è poi un ulteriore campo che evidenzia i requisiti di accesso alla pagina; ogni requisito nel concreto costituisce un parametro che deve essere specificato tramite metodo GET all'interno dell'URL o tramite metodo POST. I nomi dei parametri effettivi all'interno del sito possono essere leggermente diversi da quelli specificati nella tabella, questo per garantire una maggiore espressività. Se uno dei parametri inseriti risulta mancante RUBiS restituirà un errore.

In realtà è presente anche una categoria di parametri opzionali: nella tabella sono stati indicati in verde. Se uno di tali parametri risulta assente, RUBiS è in grado comunque di garantire un servizio; ad esempio se non è specificato alcun numero di pagina nelle sezioni

che mostrano gli oggetti dopo una ricerca, RUBiS assumerà in automatico che la pagina corrente sia la prima.

Nome pagina	Descrizione	Requisiti (parametri)
AboutMe.php	Mostra tutte le informazioni personali dell'utente autenticato, compresi oggetti venduti, in vendita o acquistati	Nickname, password
BrowseCategories.php	Mostra le categorie in cui cercare	Nickname, password, regione
BrowseRegions.php	Elenca la lista delle regioni presenti sul sito	Nessuno
BuyNowAuth.php	Login page per accedere al form di acquisto	Item ID
BuyNow.php	Form di acquisto	Nickname, password, item ID
StoreBuyNow.php	Processa un nuovo acquisto	Quantità, massima quantità consentita, item ID, user ID
PutBidAuth.php	Login page per accedere al form di piazzamento per una nuova puntata	Item ID
PutBid.php	Form per il piazzamento di una puntata	Nickname, password, item ID
StoreBid.php	Processa una nuova puntata	Nuova puntata, massima puntata ammessa, minima puntata ammessa, quantità acquisto, user ID, item ID
PutCommentAuth.php	Login page per accedere al form di scrittura di un commento	Item ID, destinatario commento (ID)
PutComment.php	Form di scrittura di un nuovo commento	Nickname, password, item ID, destinatario commento (ID)
StoreComment.php	Processa un nuovo commento	Item ID, mittente commento (ID), destinatario commento (ID), rating, commento
SellItemForm.php	Form per la messa in vendita di un oggetto	ID categoria oggetto da vendere, user ID
RegisterItem.php	Inserisce un nuovo oggetto in vendita	User ID, ID categoria oggetto da vendere, nome oggetto, prezzo iniziale, prezzo di riserva, durata asta, prezzo per "compralo subito", quantità,

		descrizione oggetto
RegisterUser.php	Inserisce un nuovo utente nel DB	Nome, cognome, nickname, password, email, regione di provenienza
SearchItemsByRegion.php	Ricerca oggetti per regione	Nome categoria, ID categoria, ID regione, numero di pagina , numero oggetti per pagina
SearchItemsByCategory.php	Ricerca oggetti per categoria	Nome categoria, ID categoria, numero di pagina , numero oggetti per pagina
ViewBidHistory.php	Mostra lo storico delle puntate su un oggetto	Item ID
ViewItem.php	Mostra generiche informazioni su un oggetto	Item ID
ViewUserInfo.php	Mostra generiche informazioni su un utente	User ID

Architettura dell'emulatore di RUBiS

Come detto in precedenza il tool per benchmark integrato in RUBiS emula durante l'esperimento il comportamento di un numero definito di utenti (tale numero è settato all'interno di un file di configurazione insieme ad altri parametri). Per ogni utente viene definita una sessione ed è aperta una connessione http al server nella quale vengono svolte varie operazioni; è previsto inoltre un think-time tra un'operazione ed un'altra (sempre personalizzabile).

Gli spostamenti sul sito dei vari utenti non sono casuali, l'emulatore implementa infatti un automa a stati finiti che decide come muoversi all'interno del sito grazie ad una tabella che gli fornisce la probabilità di passare da uno stato ad un altro (nella pratica da una pagina ad un'altra del sito).

La transition table che definisce queste probabilità può essere personalizzata ma RUBiS viene fornito con all'interno una serie di tabelle già testate. Principalmente si distinguono in due gruppi: quelle che fanno convergere gli utenti virtuali sulle pagine di ricerca (browsing mix) e quelle che li portano anche sulle pagine di acquisto degli oggetti e piazzamento di puntate (bidding mix). Per esperienza abbiamo notato che le tabelle di tipo bidding mix fanno transitare gli utenti con una buona probabilità anche nelle pagine di ricerca. Volendo fare un'analisi che copra la totalità delle operazioni presenti sul sito, abbiamo scelto di usare una tabella di tipo bidding mix per i nostri esperimenti. Una parte

della tabella usata è disponibile nella prossima pagina e dovrebbe rendere l'idea di come ogni azione sia mappata con una sua probabilità.

Per ogni richiesta da parte di un utente durante la propria sessione dovrebbe arrivare in risposta una pagina HTML; di tale pagina viene eseguito il parsing per determinarne i tempi di workload o eventuali errori.

Transition table

Transition set: 'Default transition 1.5 set'

State name	Home	Register	RegisterUser	Browse	BrowseCategories	SearchItemsInCategory	BrowseRegions	BrowseCategoriesInRegion	SearchItemsInRegion	ViewItem
Home	0.0	1.0E-4	0.01	1.0E-4	1.0E-4	1.0E-4	1.0E-4	1.0E-4	1.0E-4	1.0E-4
Register	0.07	0.0	0.0	0.01	0.01	0.01	0.01	0.01	0.01	0.01
RegisterUser	0.0	0.99	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Browse	0.7	0.0058	0.58	0.0	0.0058	0.0058	0.0058	0.0058	0.0058	0.0058
BrowseCategories	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0
SearchItemsInCategory	0.0	0.0	0.0	0.0	0.99	0.4	0.0	0.0	0.0	0.0
BrowseRegions	0.0	0.0	0.0	0.29	0.0	0.0	0.0	0.0	0.0	0.0
BrowseCategoriesInRegion	0.0	0.0	0.0	0.0	0.0	0.0	0.99	0.0	0.0	0.0
SearchItemsInRegion	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.99	0.4	0.0
ViewItem	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0
ViewItemInfo	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
ViewItemHistory	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05
BuyNowAuth	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
BuyNow	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StoreBuyBow	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
PutBidAuth	0.0	0.0	0.0	0.0	0.0	0.09	0.0	0.0	0.09	0.22
PutBid	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StoreBid	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
PutCommentAuth	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
PutComment	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.02
StoreComment	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Sell	0.05	5.0E-4	0.05	5.0E-4	5.0E-4	5.0E-4	5.0E-4	5.0E-4	5.0E-4	5.0E-4
SelectCategoryToSellItem	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SellItemForm	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RegisterItem	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Le versioni utilizzate in questo studio

La prima versione riscritta di RUBiS usa il paradigma procedurale come quella di partenza, ma volendo utilizzare in tutte le nuove versioni l'interprete di PHP nella versione 7.1 tutte le funzioni deprecate legate a PHP4 presenti nel codice sono state sostituite (per la maggior parte erano funzioni legate alle interrogazioni del database).

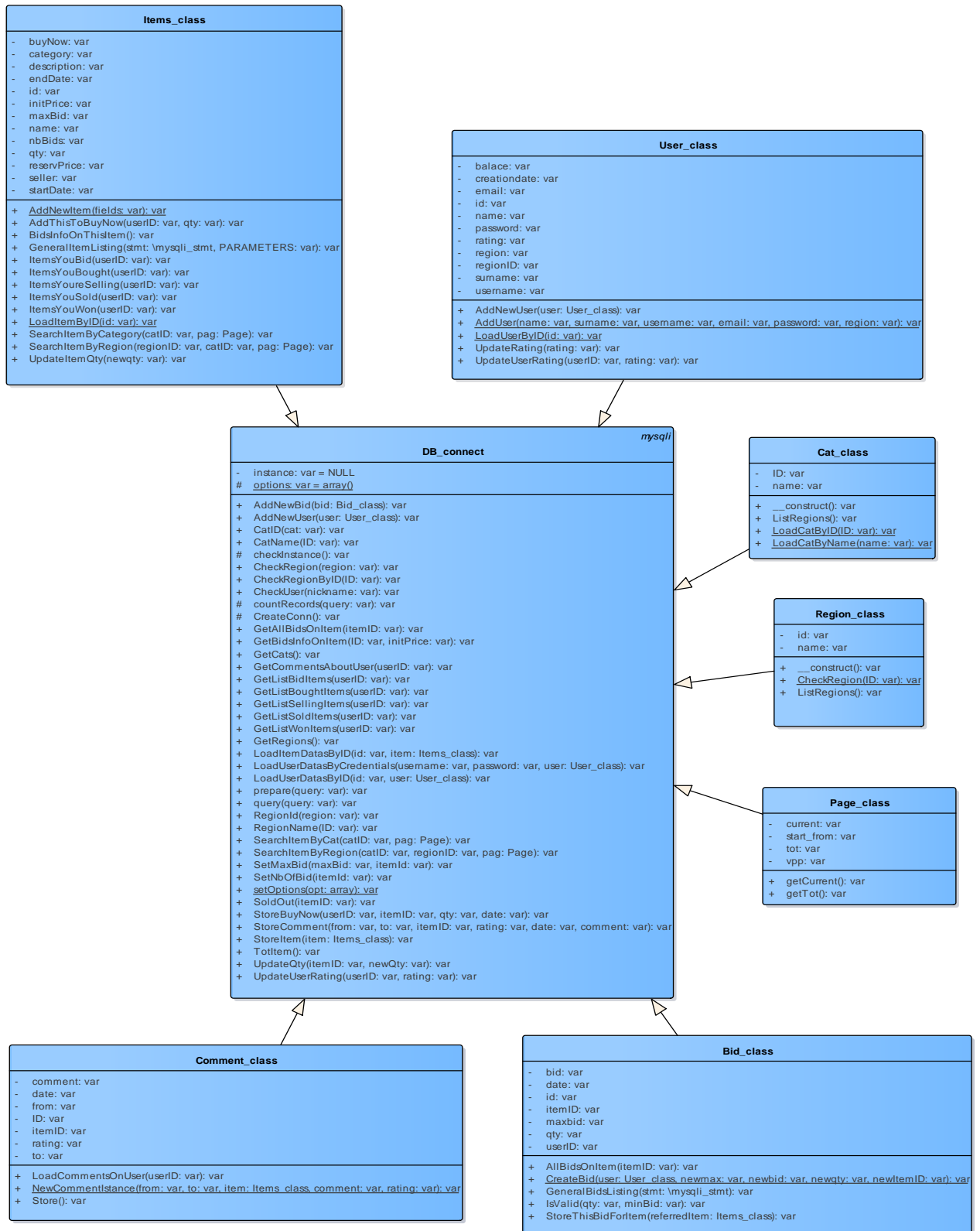
La seconda versione è invece legata al paradigma ad oggetti: sono state scritte 10 classi; tra queste ne spiccano due particolarmente onerose: quella che si interfaccia col DBMS e quella che si occupa di gestire l'output delle varie pagine.

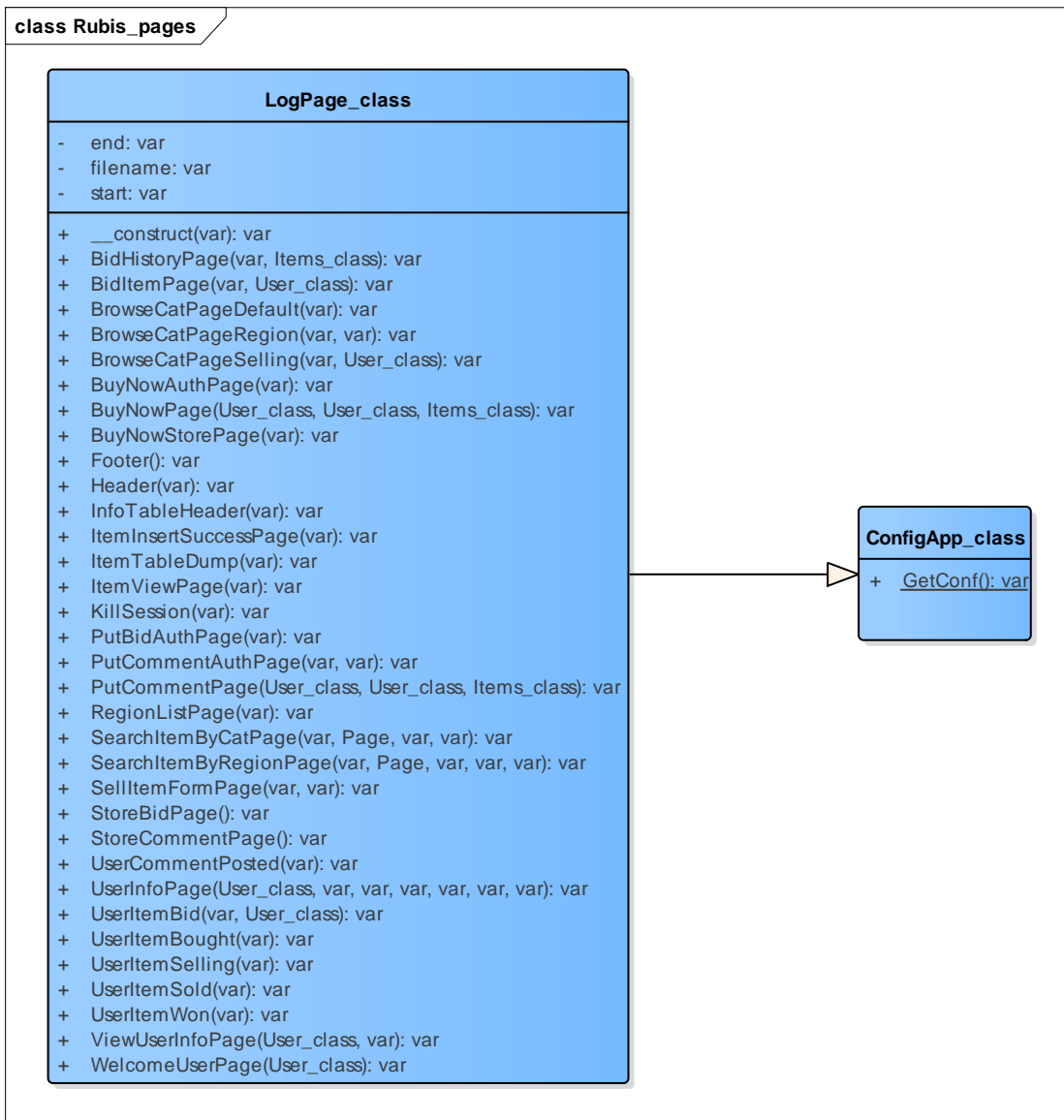
Grazie a quest'ultima classe però si è di gran lunga semplificata la leggibilità del codice e la possibilità di intervenire nell'HTML di una pagina qualora questa generi qualche errore di parsing da parte dell'emulatore. Per avere un'idea del lavoro svolto abbiamo creato un

class diagram, disponibile nelle pagine successive. Come si evince il codice è piuttosto articolato e le classi hanno parecchi metodi. Per cercare di semplificare il diagramma sono stati nascosti i costruttori ed i metodi getter e setter; tuttavia ci teniamo a ricordare che tali metodi sono implementati in ogni classe per quasi tutti gli attributi presenti. Non ci è sembrato lecito rimuovere altri metodi dalle classi poiché riteniamo che questi abbiano tutti un'importanza fondamentale per il corretto funzionamento del sito.

Tipicamente la richiesta ad una pagina di questa versione di RUBiS inizializza degli oggetti di classi intermedie come possono essere ad esempio la item class o la user class a seconda delle funzioni che la pagina deve svolgere; tali classi fanno poi generalmente uso di accessi al database e questi accessi avvengono tramite la classe DB class che implementa tutti i metodi necessari. Infine la risposta alla pagina viene generata dalla classe LogPage che si affida ad un metodo statico della classe ConfigApp per ottenere informazioni quali URL delle immagini o dei link da generare.

class Rubis





La terza versione è stata sviluppata su un framework: i framework (CodeIgniter, Zend, Symfony, per citarne alcuni) stanno avendo una grande crescita nel panorama del web development. Nel nostro caso abbiamo scelto il framework Laravel nella sua versione 5.4, l'ultima al momento in cui viene scritto questo documento.

Il framework Laravel

Laravel è un framework open source di tipo MVC (Model View Controller), secondo tale pattern il processo di risposta alla richiesta di un utente avviene generalmente attraverso tre blocchi: Un Model che si occupa dei dati e fornisce quindi i metodi per accedere al database, una View che si occupa dell'output delle pagine ed un Controller che fa da intermediario tra i primi due.

Generalmente il model non comunica direttamente con un view ma non è impossibile che accada.

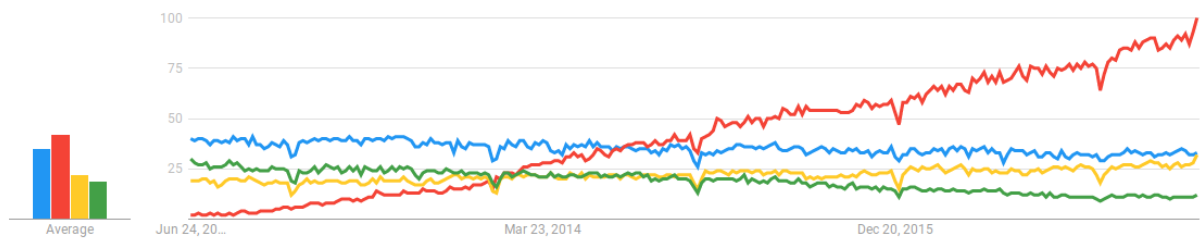
Un model comunica invece con altri model, andando a costruire una serie di relazioni che rendono più semplice e veloce il recupero dei dati da parte del controller.

Laravel allo stesso modo di altri framework utilizza un proprio ORM come Model, chiamato Eloquent. Un ORM permette l'integrazione dei database relazionali con il tipo di programmazione orientata agli oggetti che utilizziamo in Laravel.

In Eloquent una classe corrisponde a una tabella del database e per ogni modifica viene creato un oggetto. Attraverso l'utilizzo di semplici metodi possiamo effettuare operazioni complesse come la creazione, prelievo, aggiornamento ed eliminazione dei dati.

Le View rappresentano il componente di Laravel che si occupa di definire la struttura delle pagine HTML e dei dati che verranno serviti agli utenti. Laravel supporta viste definite in semplici file PHP e viste Blade, un template system diffuso nel mondo della programmazione Web. Il riconoscimento dell'engine è automatico e basato su una convenzione a livello di filename (se la vista ha estensione .php verrà utilizzato l'engine classico, altrimenti se la vista ha estensione .blade.php verrà utilizzato l'engine di blade). Attraverso alcune convenzioni si possono ottenere i valori delle variabili all'interno della pagina o effettuare iterazioni senza necessariamente dover scrivere dei blocchi di codice PHP.

Con la versione 4, come si evince dal grafico di Google trends, Laravel ha avuto una fortissima crescita ritagliandosi una grossa fetta del mercato dei framework. La versione 5.1 è stata inoltre la prima considerata LTS e questo ha contribuito non poco all'appetibilità di Laravel dal punto di vista imprenditoriale e commerciale. Tutto ciò, unito alla sua sintassi molto espressiva e alla documentazione online chiara e precisa, lo ha reso il framework ideale per il nostro studio.



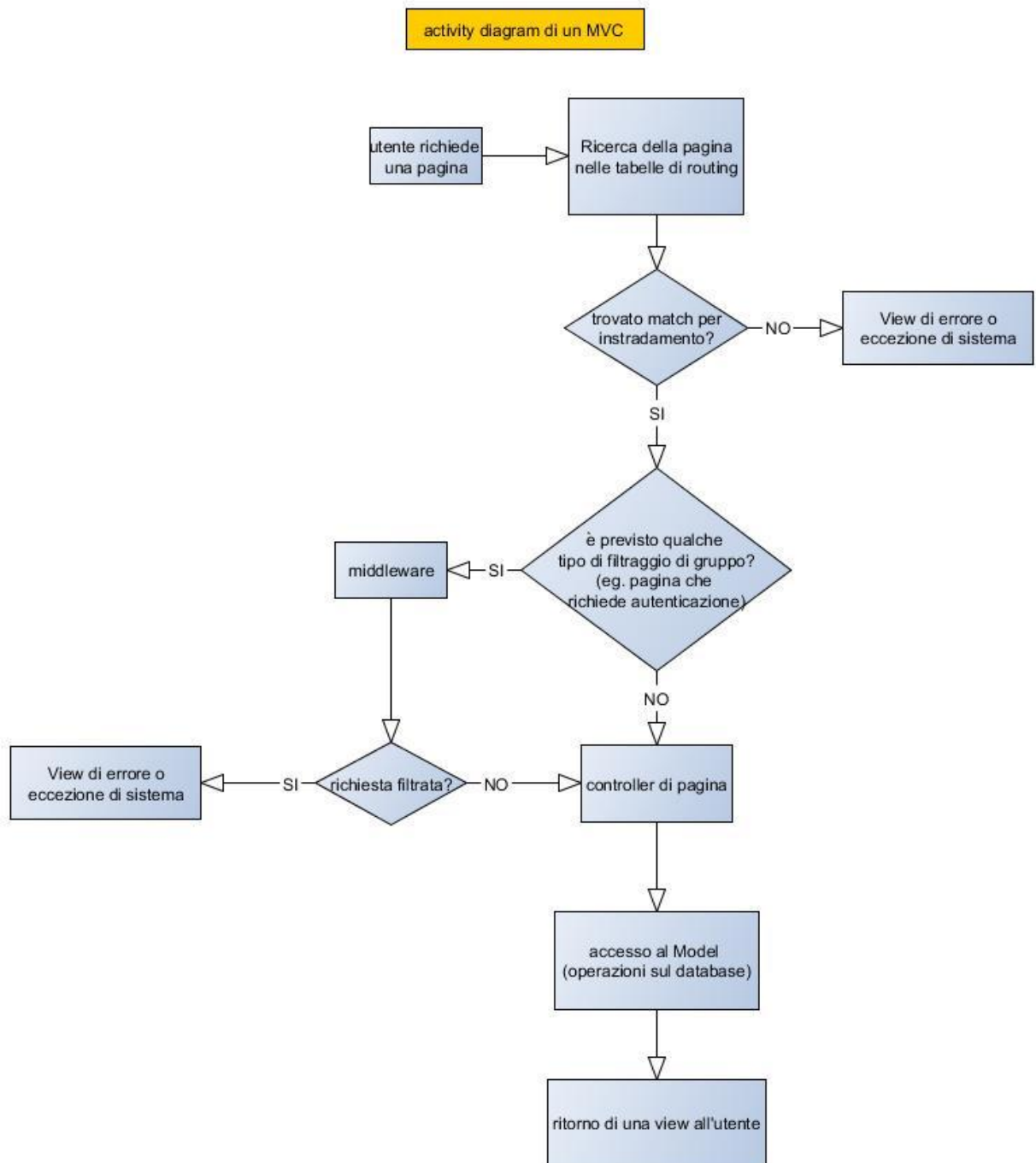
In rosso la crescita nei trend di Google di Laravel, in giallo di Symfony, in blu CodeIgniter, in verde CakePHP.

Risulterebbe a dir poco confusionario cercare di generare un class diagram per il codice scritto su Laravel; questo poiché le classi scritte spesso estendono librerie del framework con logiche particolari e non danno l'idea di quanto è stato realmente fatto. Del resto una delle caratteristiche dei framework è che vincolano il programmatore al rispetto di varie regole e pratiche che alla lunga gli semplificano il lavoro, senza che lui si debba chiedere necessariamente cosa vi è dietro e come è fatta la struttura.

Per dare allora un'idea di come funziona il framework ed, in particolare, l'uso che ne abbiamo fatto noi, abbiamo creato un activity diagram che mostra come viene gestita una generica richiesta al sito nella nostra implementazione. Generalmente una richiesta viene instradata tramite i meccanismi di routing del framework ed eventualmente viene applicato un filtro tramite un middleware. La richiesta viene poi presa in gestione da un controller che svolge tutte le operazioni che richiedono l'uso del model e restituisce una view. Quanto detto dovrebbe apparire chiaro con la prossima figura.

Per dare poi un'idea precisa di come sono state gestite le varie pagine di RUBiS con Laravel abbiamo inserito una tabella che mostra le regole di routing per ognuna delle pagine del sito (anche quelle statiche). Viene specificato dunque per ogni pagina se vi è associato un middleware e quale esso sia (nella pratica abbiamo creato solo un middleware di autenticazione con alias "login" che si occupa di verificare la presenza e la validità dei parametri username e password per le pagine che lo richiedono), viene poi inserito, qualora esista, il nome del controller ed il metodo richiamato al suo interno (un controller può essere assegnato a più pagine: ad esempio ve n'è uno per tutte le operazioni

riguardanti gli acquisti ed uno per le puntate), infine viene segnalata il nome della view alla quale l'utente viene indirizzato se tutte le operazioni vanno a buon fine.



ricordiamo che per ognuna di queste regole di routing, nel caso non siano specificati i requisiti precedentemente segnalati, verrà restituita una specifica view di errore. Nel caso di un errore non gestito Laravel solleva un'eccezione (abbiamo riportato dopo la tabella di routing un esempio di entrambi questi casi)

Per completezza aggiungiamo che se una view si trova all'interno di una sottocartella, la path che porta ad essa, in Laravel, si specifica tramite punti (del tipo: sottocartella.view). Tale convenzione non vale per altri aspetti del framework.

Aggiungiamo anche che sono state omesse alcune regole di routing che si occupano di instradare gli URL delle immagini: queste regole sono state create per evitare di generare qualche possibile errore da parte del client nel parsing, ma a livello di esperienza utente non influiscono in alcun modo.

MATCHING	MIDDLE WARE	CONTROLLER	METODO CONTROLLER	VIEW
index.html,	NO	NO	NO	home
register.html	NO	NO	NO	register
browse.html,	NO	NO	NO	browse
sell.html,	NO	NO	NO	welcome.sell
about_me.html	NO	NO	NO	welcome.aboutme
RegisterUser.php	NO	addNewUserController	AddNewUser	welcome.newuser
BrowseRegions.php	NO	getRegionsController	all	regions
BrowseCategories.php	NO	browseCategoriesController	SwitchUsersTo	browsecatpage
SearchItemsByRegion.php	NO	searchItemsController	ByRegion	searches.byregion
SearchItemsByCategory.php	NO	searchItemsController	ByCategory	searches.bycategory
ViewItem.php	NO	itemInfoController	itemInfoPage	item_information
ViewBidHistory.php	NO	itemInfoController	itemBidHistory	bids_information
ViewUserInfo.php	NO	userInfoController	AboutSomeone	other_user_information
PutCommentAuth.php	NO	commentsController	CommentAuth	comments.auth
StoreComment.php	NO	commentsController	CommentStore	comments.store

SellItemForm.php	NO	sellController	FormSeller	sell.form
RegisterItem.php	NO	sellController	StoreSell	sell.store
PutBidAuth.php	NO	bidsController	BidAuth	bid.auth
StoreBid.php	NO	bidsController	BidStore	bid.store
BuyNowAuth.php	NO	buyNowController	BuyAuth	buynow.auth
StoreBuyNow.php	NO	buyNowController	BuyStore	buynow.store
AboutMe.php	login	userInfoController	AboutMe	user_information
PutComment.php	login	commentsController	CommentForm	comments.form
PutBid.php	login	bidsController	BidForm	bid.form
BuyNow.php	login	buyNowController	BuyForm	buynow.form



[Home](#)

[Register](#)

We cannot process your request due to the following error :

required parameters missed

RUBiS (C) Rice University/INRIA

Page generated by SearchItemsByRegion.php in 0.012204885482788 seconds

Sorry, the page you are looking for could not be found.



(1/1) NotFoundHttpException

```

In RouteCollection.php (line 179)

at RouteCollection->match(object(Request))
In Router.php (line 548)

at Router->findRoute(object(Request))
In Router.php (line 527)

at Router->dispatchToRoute(object(Request))
In Router.php (line 513)

at Router->dispatch(object(Request))
In Kernel.php (line 174)

at Kernel->IlluminateFoundationHttp(closure)(object(Request))
In Pipeline.php (line 30)

at Pipeline->IlluminateRouting(closure)(object(Request))
In TransformsRequest.php (line 30)

```

Altro software utilizzato

Il webserver utilizzato durante gli esperimenti è stato sempre Apache, nella versione 2.4. Non vi è stata necessità di modificare alcuna configurazione di default per la versione procedurale ed ad oggetti; Per Laravel è stato necessario abilitare il `mod_rewrite` ed utilizzare un file di configurazione personalizzato.

Anche per quanto riguarda il DBMS si è scelto di usare unicamente MySQL (14.14). Per la versione ad oggetti tutte le connessioni sono state gestite attraverso una classe che estende la libreria `mysqli`; la versione procedurale ha utilizzato chiamate dirette alla stessa libreria ovviamente in forma procedurale; la versione Laravel ha utilizzato invece il suo ORM (Object-Relational Mapping) Eloquent.

È stato utilizzato il software `composer` per scaricare ed installare Laravel e le sue dipendenze.

L'emulatore client è quello originale ed è stato ricompilato ed eseguito senza particolari problemi con le recenti `openjdk 1.8.0`.

È stato utilizzato il software MATLAB per la generazione i grafici illustrativi dell'andamento del throughput, del tempo di risposta, e del numero di errori in funzione del numero crescente di utenti.

Sono stati scritti alcuni script aggiuntivi: uno script in `bash` per automatizzare al massimo il processo di emulazione in modo da effettuare esperimenti 24 ore al giorno con un numero via via crescente di utenti ed uno in `PHP` per effettuare il parsing dei risultati dell'emulatore e restituirne dati utili in un formato compatibile con Matlab. In questo modo è stato relativamente rapido generare le varie curve di carico del server con le varie versioni di RUBiS.

4. Hardware

Per gli esperimenti sono stati messi a disposizione dall'università di Modena due server sui quali peraltro erano già stati effettuati studi su RUBiS anni prima.

Entrambe le macchine hanno processori Intel(R) Xeon(TM) a 2.40GHz con cache size 512 KB. La RAM totale a disposizione per ciascuna macchina è 1Gb disposta in due slot da 500Mb di tipo DDR1.

Su ciascuna macchina è montato un Hard Disk Seagate ST380011A UDMA/100 7200RPM da 80GB.

Le due macchine sono connesse in rete e collegate tra loro attraverso uno switch 3300 della 3com.

5. Svolgimento dell'esperimento

Installazione

Su entrambe le macchine sono state installate tramite NetInstall le versioni server di Ubuntu 16.10.

Client

Sulla macchina client è stato scaricato il pacchetto RUBiS dal sito <http://RUBiS.ow2.org/> [2]

Per lanciare l'emulatore è bastato installare le JDK 1.8 e crearne dei link simbolici in /bin tramite:

- `sudo apt-get install openjdk-8-jdk`
- `sudo ln -s /usr/bin/javac /bin/javac`
- `sudo ln /usr/bin/java /bin/java`
- `sudo ln /usr/bin/jar /bin/jar`

Inoltre l'emulatore di RUBiS utilizza il software Sar per raccogliere le informazioni sulle prestazioni.

Tale software fa parte del pacchetto Sysstat il quale tuttavia è supportato da RUBiS fino alla versione

8.0. Non è possibile dunque installarla tramite apt dai repository di Ubuntu e si è reso necessario scaricare il pacchetto dall'archivio

<http://sebastien.godard.pagesperso-orange.fr/download.html> ed installarlo tramite comando make dopo aver eseguito:

- `./configure --disable-documentation --disable-stripping --disable-sensors --disable-nls`

Infine è necessario installare il server ssh su entrambe le macchine e condividere le chiavi pubbliche generate tramite

- `ssh-keygen -t rsa`

in questo modo il client RUBiS potrà connettersi alla macchina server senza procedure di autenticazione. [3]

Server

Sul server è anzitutto necessaria l'installazione di PHP7.x, MySQL e Apache2. Ciò si può fare attraverso i repository di Ubuntu o direttamente in fase di installazione del sistema operativo spuntando "server LAMP" nella scelta dei pacchetti da installare.

Una volta configurato LAMP, avendo già a disposizione un dump SQL del database RUBiS è stato semplice ricreare il Database per l'esperimento.

Questi passaggi sono già sufficienti per l'esecuzione delle versioni procedurali ed ad oggetti. Come accennato in precedenza tuttavia, per Laravel si è reso necessario l'uso di un file di configurazione di apache personalizzato come riportato nella pagina successiva

```
potpov@ubuntu:/etc/apache2/sites-available$ cat laravel.conf
<VirtualHost *:80>
    ServerAdmin admin@myrubis.com
    DocumentRoot /var/www/html/PHP/public/
    ServerName myrubis.com
    ServerAlias www.myrubis.com
    <Directory /var/www/html/PHP/>
        Options FollowSymLinks
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>
    ErrorLog /var/log/apache2/myrubis.com-error_log
    CustomLog /var/log/apache2/myrubis.com-access_log common
</VirtualHost>
```

una volta scritto, è necessario disabilitare la configurazione di default ed attivare la nuova

- a2dissite 000-default.conf
- a2ensite Laravel.conf
- service apache2 reload

infine, per utilizzare Laravel è necessario abilitare il mod rewrite di apache

- sudo a2enmod rewrite

ed installare le seguenti estensioni di PHP tramite apt

- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Considerazioni preliminari sull'esperimento

È fondamentale tenere presente che la macchina server non ha un hardware particolarmente aggiornato; i framework così come anche il paradigma ad oggetti, hanno trovato spazio per svilupparsi ed essere adottati proprio grazie ad hardware sempre più prestante. Per tale ragione si è scelto di mostrare due andamenti: il primo, su piccola scala, mostra la curva di carico del server per simulazioni con range di clients da 1 a 40. Il secondo mostra tale curva su un range di utenti molto più ampio 1-800. La seconda

simulazione mira a saturare le risorse del server e mostra effettivamente quale versione, alla lunga, eccella dal punto di vista prestazionale.

Le tre fasi: RumpUp, RunTime, RumpDown

Come accennato in precedenza, l'esperimento si sviluppa su tre fasi. Una prima fase detta di RumpUp nella quale il sistema inizializza le sessioni e inizia le prime richieste al server fino a quando non raggiunge uno stadio stabile. Questo stadio dà inizio alla seconda fase, di RunTime, la più lunga tra le tre, nella quale le misurazioni assumono notevole importanza. Infine vi è una fase di RumpDown dove le richieste diminuiscono e l'esperimento si avvia al termine.

Sono state scelte le seguenti durate per le tre fasi:

RumpUp: 2 minuti, RunTime: 10 minuti, RumpDown: 2 minuti. Tali durate sono abbastanza in linea con gli esperimenti sulle versioni originali di RUBiS.

Esperimento su piccolo range: 1-40 Clients

Proponiamo di seguito il tempo di risposta in millisecondi del server alle richieste del client durante la fase di RunTime. Vi sono due risultati notevoli nel grafico sottostante.

Il primo è che nonostante ci si aspettasse una resistenza davvero bassa per quanto riguarda le versioni ad oggetti (si supponeva un'impennata dei tempi di risposta già intorno alle emulazioni con 20 clients), tutte e tre le versioni hanno mantenuto tempi di risposta accettabili durante questo tipo di simulazione.

Il secondo risultato notevole è la vicinanza di Laravel all'andamento della versione procedurale su piccolo range; essendo comunque Laravel un framework costruito sul paradigma ad oggetti ci si sarebbe aspettato di vedere un andamento più vicino alla versione ad oggetti semplice.

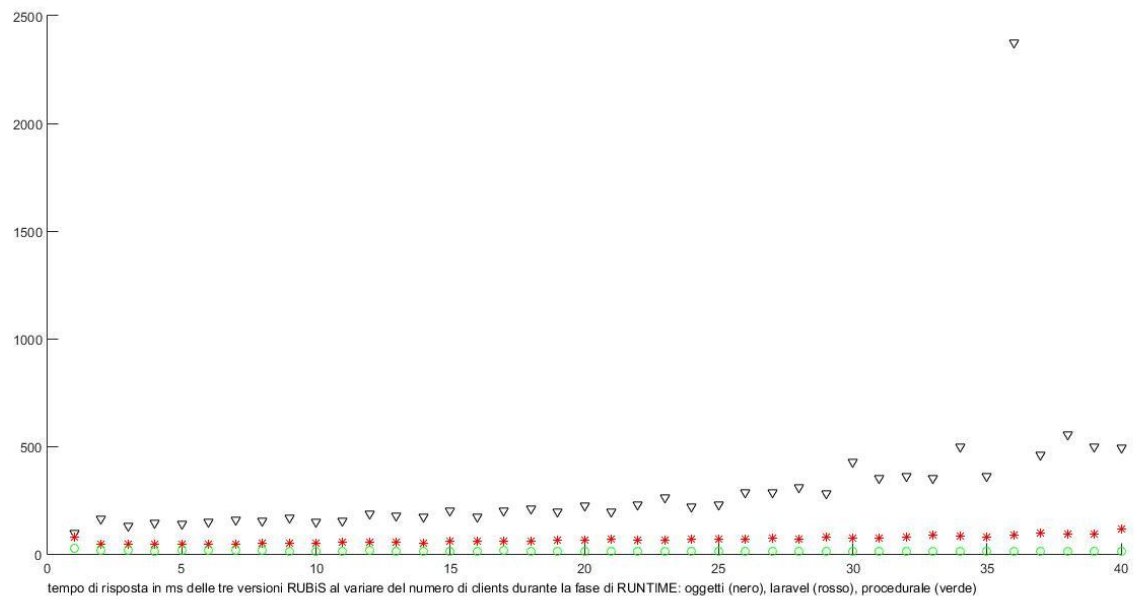
In questo caso le motivazioni delle ottime prestazioni del framework possono trovare risposta dietro ai meccanismi di caching di Laravel, soprattutto relativamente alle view di cui si è parlato prima. Quest'ultime rappresentano un grosso collo di bottiglia per la versione ad oggetti priva di framework che è costretta a rigenerare ogni volta l'output delle pagine tramite un'apposita classe.

Laravel prevede la possibilità di pulire la cache tramite Artisan, una propria linea di comando che può essere considerata una sorta di “coltellino svizzero” del framework. Molti comandi sono rapidamente svolti tramite Artisan, come la creazione di controller o di model.

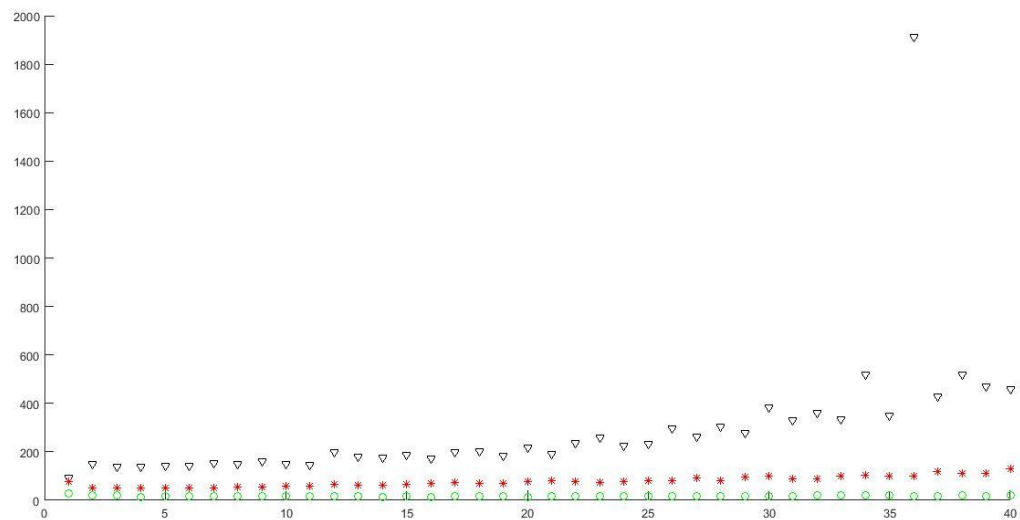
Si può pulire manualmente l’intera cache o quella relativa alle view tramite i comandi:

- `php artisan view:clear`
- `php artisan cache:clear`

sfortunatamente non è prevista attualmente nella documentazione di Laravel un’opzione ufficiale per disabilitare la cache delle view senza evitare l’uso di blade; per tale ragione non è stato possibile effettuare dei test ad hoc sull’apporto del caching nelle prestazioni di Laravel.

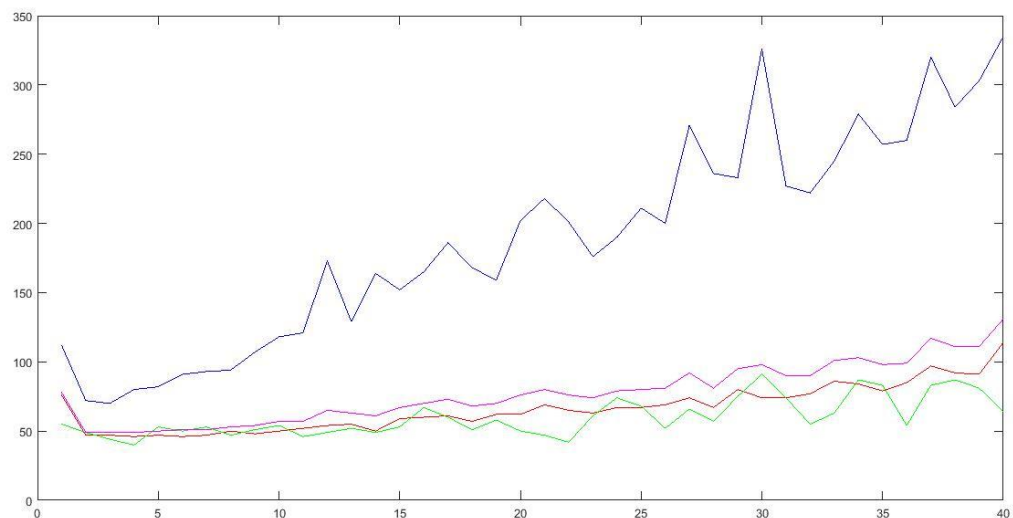


Nei grafici presenti in questa pagina i triangoli neri rappresentano l’andamento della versione ad oggetti senza framework, gli asterischi rossi l’andamento di Laravel e i cerchi verdi della versione procedurale



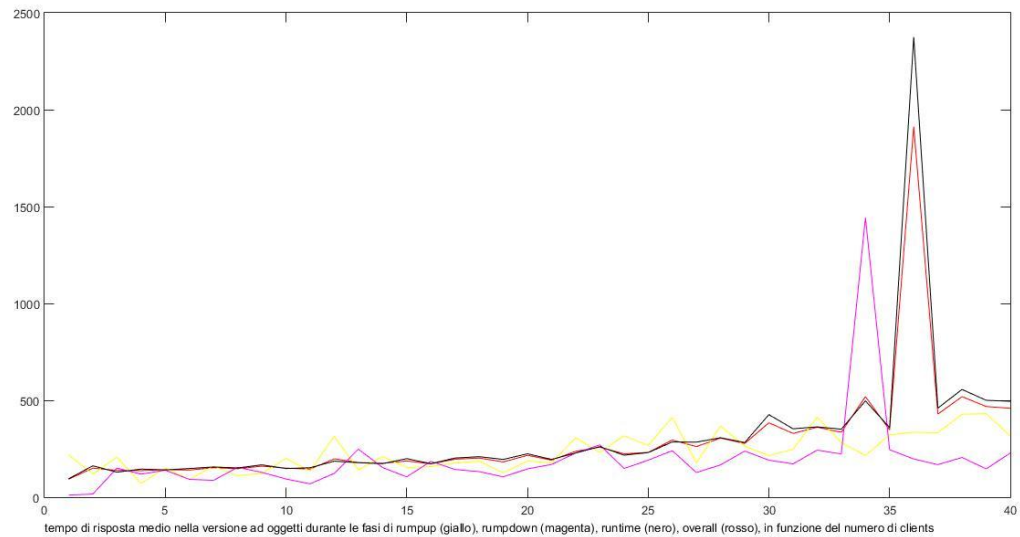
Il grafico soprastante rappresenta l'andamento medio registrato dall'emulatore tra le tre fasi: si vede bene che esso si attiene a quello della fase di RunTime.

Una curiosità è data invece dal grafico sottostante che rappresenta la comparazione in Laravel del tempo di risposta per la fase di rumpUp in blu, quella di runTime in rosso, quella di rumpDown in verde e quella di overAll (media) in giallo.



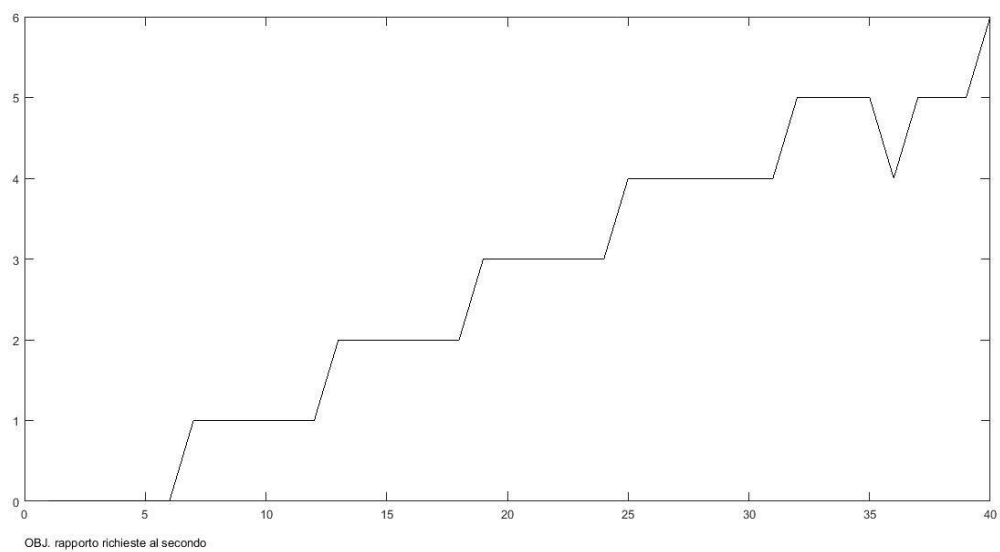
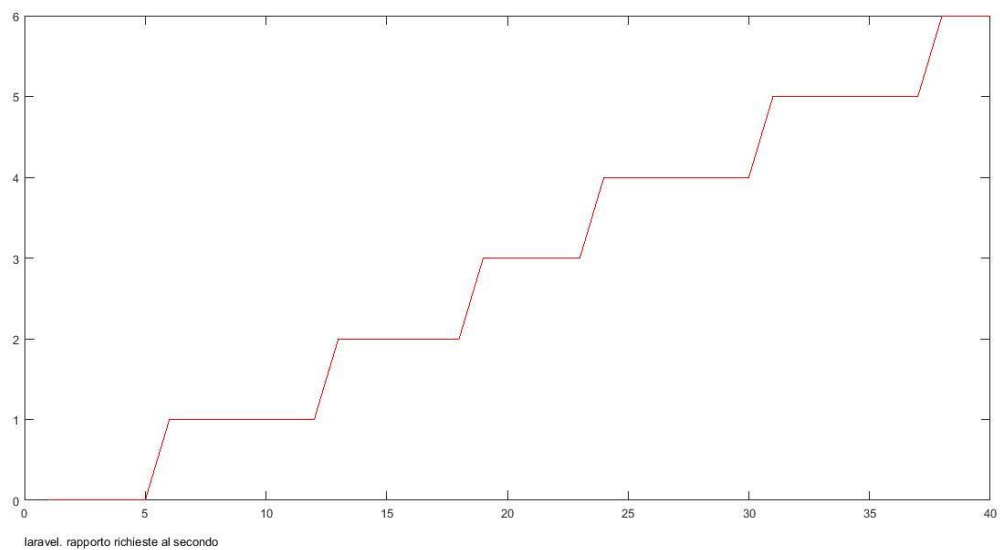
Appare abbastanza evidente che Laravel soffra molto durante la fase iniziale. Potrebbe essere anche qui indicativo del fatto che durante le prime richieste il server non può accedere ad alcun tipo di cache. Del resto tracciando lo stesso tipo di grafico per la versione ad oggetti senza framework l'andamento della fase di rumpUp è molto meno

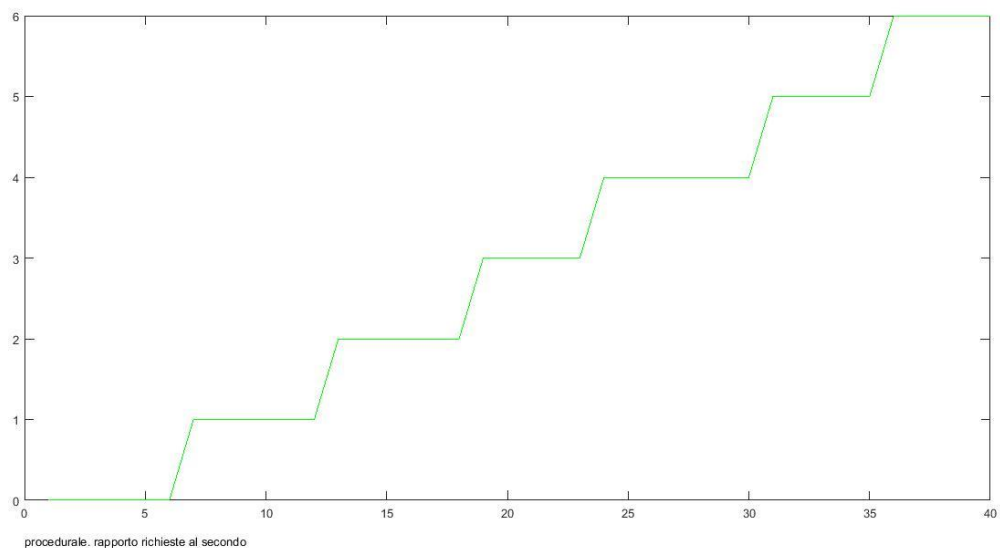
discostato dalle altre fasi ma, come si è visto, in generale le prestazioni sono le peggiori.



Analisi del throughput

Un discorso a parte va fatto per quanto concerne il throughput che in questa fase delle simulazioni non ha fornito particolari scostamenti tra le tre versioni. Vi è infatti un incremento abbastanza graduale delle richieste al secondo servite dal server con poche differenze tra le tre versioni. Riportiamo per completezza i grafici nelle pagine sottostanti.





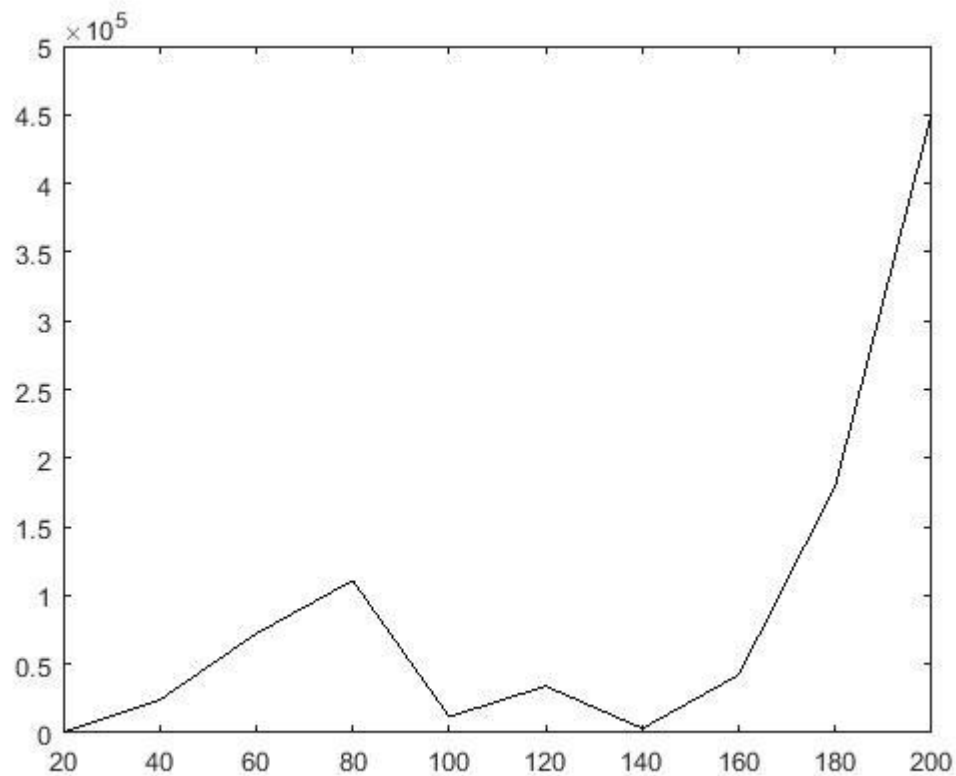
Esperimento su grosso range: 1-800 Clients

Questa seconda parte di esperimenti ha coinvolto un numero di utenti molto più elevato. L'obiettivo della simulazione era saturare almeno due versioni su tre di RUBiS per verificare quale fosse la più prestante e di quanto. Ovviamente non è stato fatto un blocco di simulazioni con rapporto di crescita degli utenti di 1:1 come nel caso precedente ma si è scelto di procedere con un rapporto di crescita di 1:20 così da effettuare comunque cicli da 40 simulazioni.

La versione ad oggetti è stata la prima a cadere, come già si intuiva negli esperimenti su range minore.

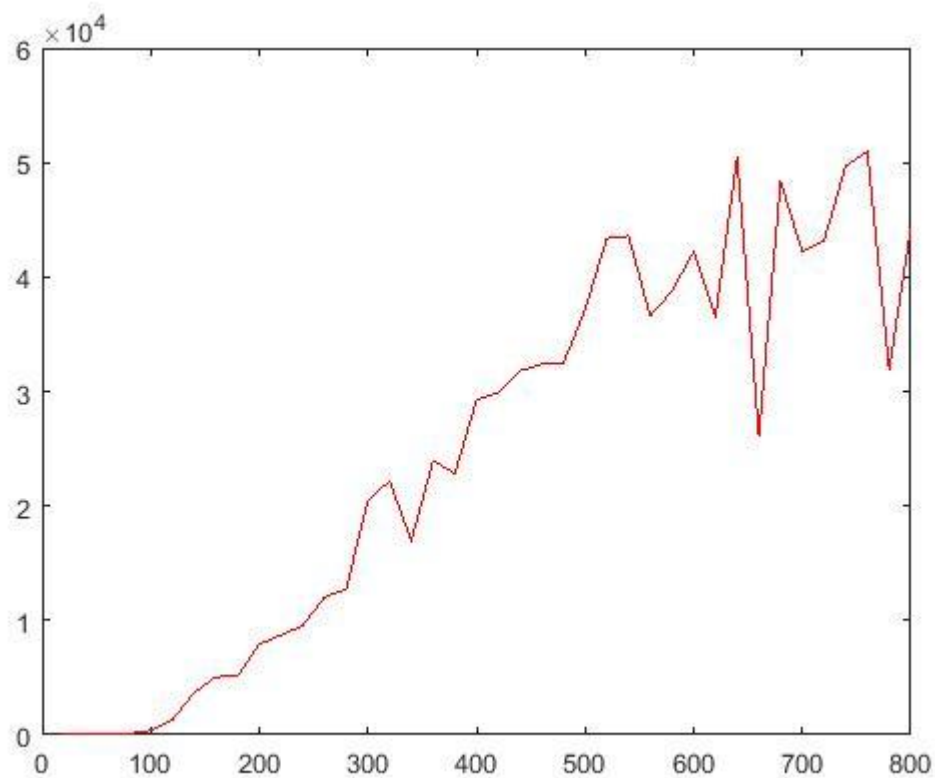
intorno ai 120 client infatti il server ad oggetti ha avuto un'impennata nel tempo di risposta. È bene notare che la maggior parte delle risposte del server durante la fase di RunTime già dai 130 Clients erano errori di tipo 5xx; per tale motivo si è scelto di riportare di seguito il grafico con la curva di carico del server fino a 200 Clients. La ragione dietro ciò sta nel fatto che oltre una certa soglia le statistiche di benchmark sono poco obiettive poiché si basano su un numero di risposte molto basso rispetto al numero effettivo richiesto dalle sessioni attive (banalmente perché la maggior parte delle richieste restituisce un errore o viene rifiutata). C'è inoltre da dire che sopra le 200 richieste il server ad oggetti finiva in crash necessitando un riavvio. Era dunque praticamente impossibile lanciare simulazioni sopra tale soglia.

I grafici successivi mostreranno gli andamenti al variare del numero di client durante la fase di RunTime.



La versione con Laravel (il cui andamento è rappresentato nel grafico della prossima pagina) si è comportata molto meglio di quella ad oggetti anche su larga scala. Il punto di saturazione del server in questo caso è risultato intorno ai 650 Clients.

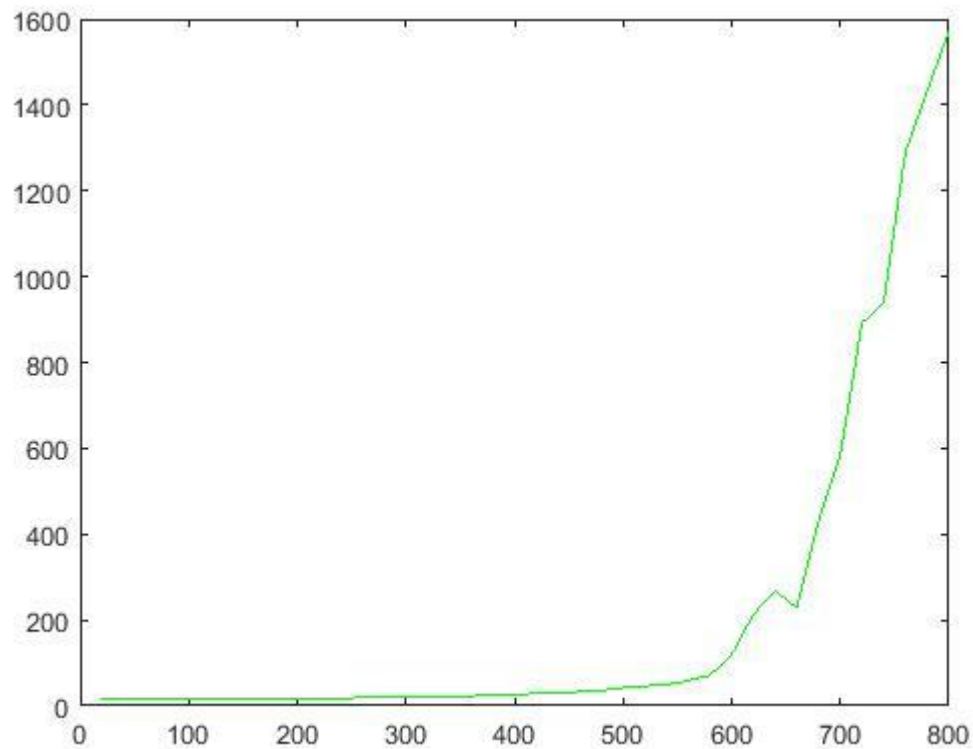
Si è scelto di riportare il grafico con l'intero range di simulazioni (fino ad 800) ma è bene tenere presente il dato fornito nella precedente affermazione: sopra i 650 clients il numero di risposte di tipo 5xx da parte del server è aumentato drasticamente; l'andamento molto instabile delle prestazioni oltre la cifra limite di 650 è sintomatico proprio di tutti questi errori ed è una dimostrazione pratica di quello che già si ipotizzava prima in merito alla versione ad oggetti senza framework per valori oltre la soglia di 200 Clients.



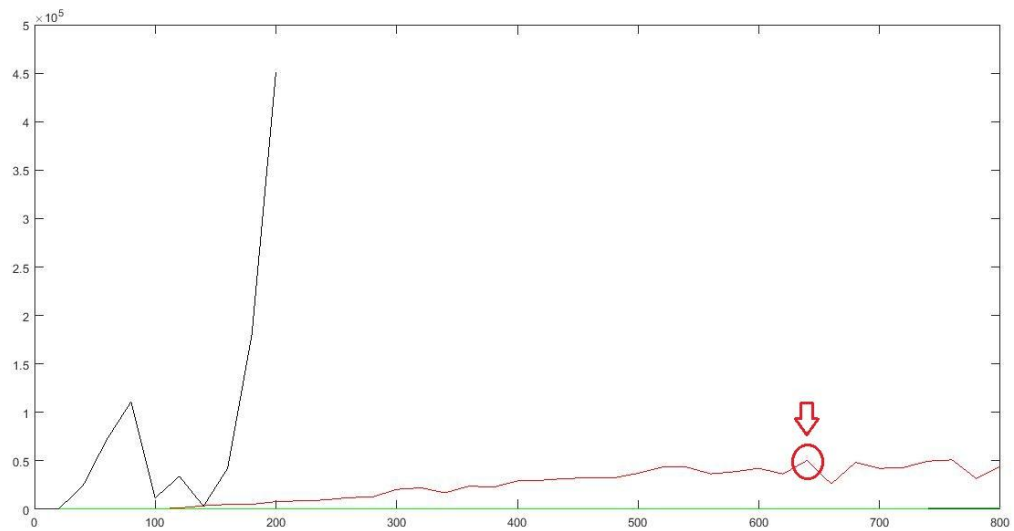
Per ultimo ma non per importanza riportiamo l'andamento della versione procedurale. Le prestazioni su larga scala rispetto alle precedenti versioni, come era facile intuire, sono migliori.

Il paradigma procedurale si conferma dal solo punto di vista prestazionale il metodo più efficiente.

Anche qui è evidente l'aumento del tempo di risposta intorno ai 650 utenti ma è opportuno sottolineare che è un aumento relativo perché i tempi della versione procedurale, pur aumentando, si mantengono ben al di sotto di un ordine di grandezza rispetto alla versione di Laravel e di due ordini di grandezza rispetto a quella ad oggetti senza framework. Riportiamo di seguito l'andamento della versione procedurale



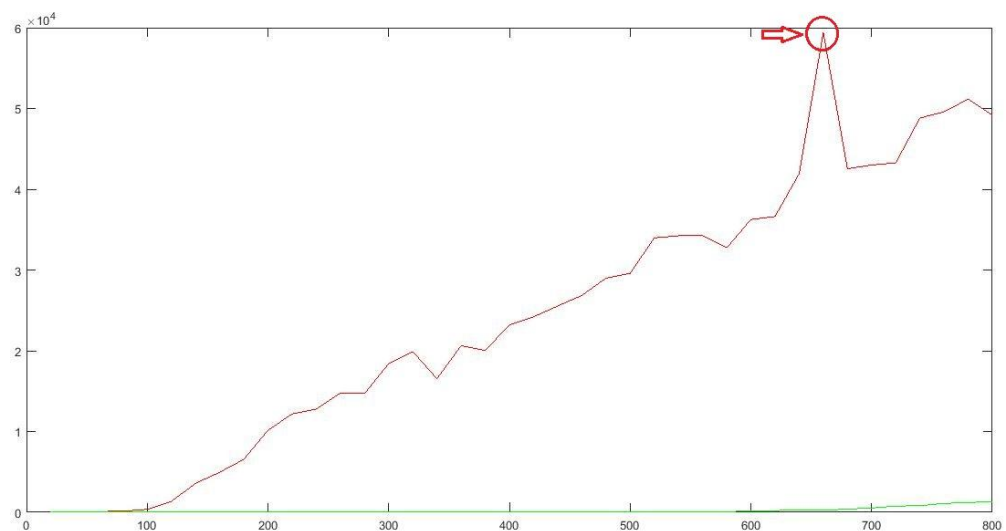
Nel prossimo grafico abbiamo confrontato le tre versioni durante la fase di RunTime: appare molto evidente ciò che si è detto riguardo gli ordini di grandezza. La versione ad oggetti senza framework infatti copre totalmente i picchi delle altre versioni. Se non si fossero analizzati i grafici singolarmente, nel prossimo grafico la versione con Laravel apparirebbe costante e la versione ad oggetti semplicemente piatta, ma sappiamo bene che non è così. Per questioni di chiarezza nel grafico è stato evidenziato il punto di saturazione della versione in Laravel (rosso) rispetto alla versione procedurale (verde) e a quella ad oggetti senza framework (in nero).



Per rendere più evidente la differenza tra la versione con Laravel rispetto a quella procedurale riportiamo il grafico di quest'ultime a confronto senza la versione ad oggetti. Abbiamo qui scelto di usare l'andamento medio tra tre fasi piuttosto che la sola fase di runTime poiché, come si è visto, in questo modo con Laravel si ottengono risultati più oggettivi.

Eliminando la versione ad oggetti senza framework si abbassa drasticamente l'ordine di grandezza del grafico e le differenze appaiono più chiare.

Anche in questo caso il punto di saturazione della versione con Laravel è stato evidenziato con un cerchio rosso.

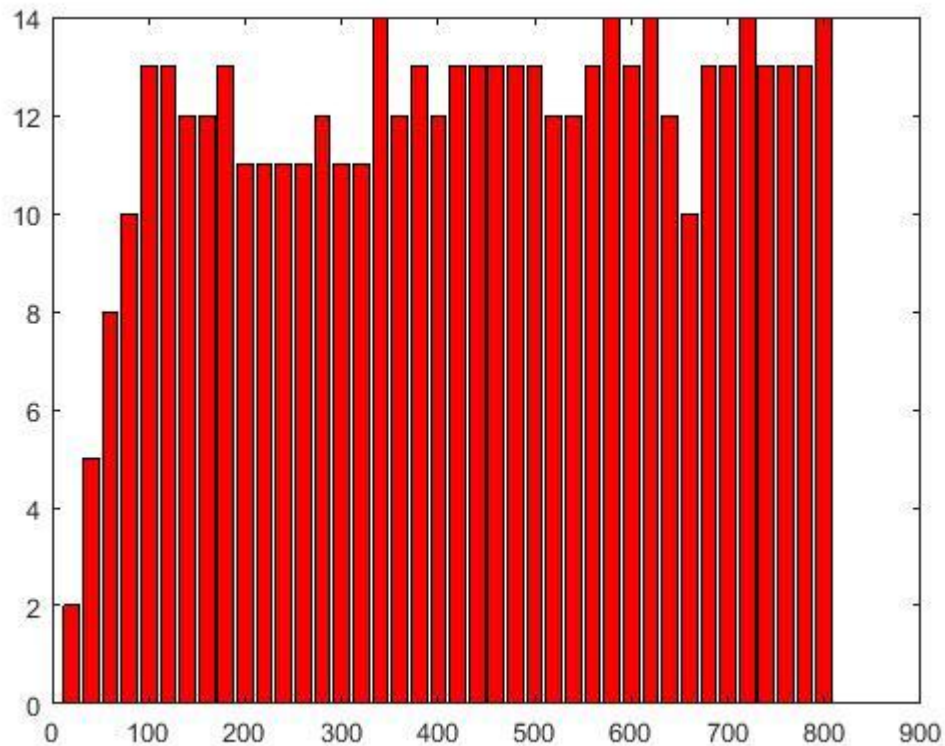


Analisi del throughput

Riportiamo infine l'andamento del throughput per questo secondo gruppo di esperimenti. L'unità di misura è Richieste/Secondo.

Anche qui la versione ad oggetti senza framework ha avuto un comportamento piuttosto fallimentare se comparato con le altre due versioni. Talmente fallimentare che il grafico di throughput è risultato irregolare e di poco interesse per la nostra analisi. Per tale ragione si è scelto di fare una comparazione tra l'andamento del throughput per le sole versioni procedurale e con framework.

Nel seguito riportiamo separatamente i grafici del throughput relativi alla media tra le tre fasi con la versione Laravel (in rosso) e con la versione procedurale (in verde).

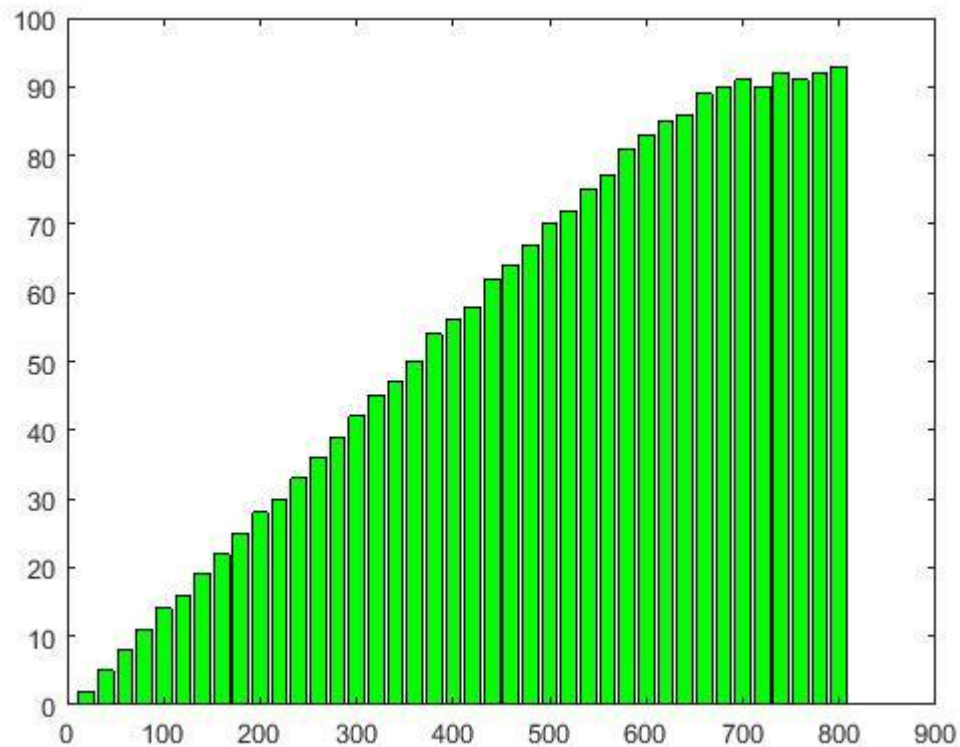


Bisogna sottolineare un paio di aspetti riguardo questi due grafici.

Il primo è che la versione oggetti si colloca anche qui al primo posto tra le due con differenze piuttosto evidenti: mentre Laravel si assesta intorno alle 10 richieste servite al secondo, la versione procedurale arriva fino a più di 90.

Il secondo aspetto riguarda l'andamento della curva. Entrambi i grafici hanno ricalcato abbastanza fedelmente la tipica curva del throughput di un server ma risulta molto chiaro che la versione procedurale ha avuto una crescita più lineare rispetto a quella di Laravel

nella quale si distinguono molte oscillazioni irregolari. Questa considerazione non vale unicamente per il throughput ma è saltata all'occhio in tutti gli altri grafici evidenziati in precedenza. Questa forte regolarità della versione procedurale può essere intesa come un ulteriore punto di forza dal punto di vista prestazionale e della stabilità del codice.



Analisi degli errori del client

Abbiamo scelto di dedicare un paragrafo a parte per gli errori restituiti dalle simulazioni. RUBiS mette a disposizione tre livelli di debug durante un esperimento che permettono di farsi un'idea di cosa funziona e di cosa va storto durante l'emulazione. Per l'esperienza fatta in laboratorio abbiamo identificato tre diversi tipi di errore.

La prima tipologia è legata ad un errore del parser. Significa che RUBiS ha richiesto una pagina, la risposta del server è arrivata, ma il client non è riuscito ad interpretarla. Sono presenti molti errori di questo tipo, i quali hanno suscitato la nostra attenzione ed hanno portato ad un'analisi dettagliata della fonte.

Avendo riscritto quasi totalmente per tre volte il sito web si è prestata massima attenzione al tipo di richieste che potevano arrivare dal client e all'output ottimale per il parsing dell'emulatore; infatti gli errori ottenuti in questo caso appaiono privi di senso perché sono legati a richieste non lecite (ad esempio tentativi di piazzare un rilancio su un asta per un oggetto con ID non valido, oppure la ricerca di informazioni di un utente che non esiste), le spiegazioni plausibili dietro questi comportamenti possono essere legate semplicemente a bug dell'emulatore oppure al tentativo dei programmatori originali di RUBiS di creare anche del "rumore" durante l'emulazione, tramite appunto richieste illecite.

```
Thread UserSession9: Cannot found name=maxQty value= in last HTML reply
Thread UserSession9: Cannot found name=minBid value= in last HTML reply
Thread UserSession9: Cannot found item id in last HTML reply
Thread UserSession9: Error returned from access to http://192.168.1.101:80/PHP/StoreBid.php?itemId=-1&userId=487767&minBid=-1.0&maxQty=1&bid=1.0&maxBid=3.0&qty=1
Thread UserSession15: Unable to open URL http://192.168.1.101:80/PHP/SearchItemsByCategory.php?category=0&categoryName=Antiques+%26+Art&page=2&nbOfItems=20 (Server returned HTTP response code: 500 for URL: http://192.168.1.101:80/PHP/SearchItemsByCategory.php?category=0&categoryName=Antiques+%26+Art&page=2&nbOfItems=20)
Thread UserSession11: Cannot found item id in last HTML reply
```

La seconda tipologia di errore è quella che ci ha permesso di capire quale fosse il punto critico delle varie versioni. Da un certo punto in poi infatti, nella crescita del numero di clients, il server non sarà più in grado di rispondere alle richieste dell'emulatore. Tipicamente in questa situazione un server risponde con un codice di stato HTTP di tipo 5xx.

```
Thread UserSession26: Unable to open URL http://192.168.1.101:80/PHP/BrowseCategories.php (Server returned HTTP response code: 500 for URL: http://192.168.1.101:80/PHP/BrowseCategories.php)
Thread UserSession119: Unable to open URL http://192.168.1.101:80/PHP/BrowseCategories.php (Server returned HTTP response code: 500 for URL: http://192.168.1.101:80/PHP/BrowseCategories.php)
Thread UserSession137: Unable to open URL http://192.168.1.101:80/PHP/BrowseCategories.php (Server returned HTTP response code: 500 for URL: http://192.168.1.101:80/PHP/BrowseCategories.php)
```

In Laravel questo tipo di errore trovava una forma un po' più chiara nei log, come si vede nell'immagine sottostante. C'è da aggiungere per completezza che Laravel, anche quando si trovava in circostanze ormai critiche (sopra la soglia dei 650 Clients) riusciva comunque a gestire alcune richieste.

```
Thread UserSession111: Unable to open URL http://192.168.1.101:80/PHP/index.html (Connessione scaduta (Connection timed out))
Thread UserSession698: Unable to open URL http://192.168.1.101:80/PHP/index.html (Connessione scaduta (Connection timed out))
Thread UserSession12: Error while downloading image http://192.168.1.101:80http://192.168.1.101/storage/RUBiS_logo.jpg (Connessione scaduta (Connection timed out))
Thread UserSession699: Unable to open URL http://192.168.1.101:80/PHP/index.html (Connessione scaduta (Connection timed out))
Thread UserSession106: Unable to open URL http://192.168.1.101:80/PHP/index.html (Connessione scaduta (Connection timed out))
```

Il terzo ed ultimo tipo di errore riscontrato dai log è a nostro avviso quello più interessante. In questo caso apparentemente la richiesta del client sembra lecita, l'URL è corretto, gli ID evidenziati esistenti; Questo inizialmente può lasciare spiazzati. Tuttavia dopo qualche analisi si evince come questo genere di errore fosse concentrato su richieste di bidding e buynow, da un'analisi più approfondita sul DBMS è risultato che l'item richiesto fosse esaurito. Le cause di questo errore possono essere due e andrebbero rintracciate all'interno del codice sorgente JAVA dell'emulatore: può infatti essere che vi siano più richieste concorrenti per l'acquisto di uno stesso item e questo generi un errore di consistenza nel database per cui uno degli utenti riceva un messaggio di errore (questo sarebbe risolvibile mediante uso di lockTables, commit e rollback che tuttavia non sono

stati implementati nelle nostre versioni), oppure può dipendere dal fatto che l'emulatore si limiti a ricavare gli id degli item in vendita tramite le pagine di browsing ma non faccia alcuna verifica sulla quantità dell'oggetto in vendita (nell'implementazione del sito risultano nell'elenco vendite anche oggetti con quantità nulla perché esauriti ma l'acquisto o il piazzamento di scommesse su di essi viene bloccato poi in fase di registrazione restituendo appunto il tipo di errore evidenziato dal client). Non essendoci calati nell'analisi dettagliata del codice in Java dell'emulatore, la soluzione, sebbene abbastanza grezza, per questo genere di errori è stata quella di aumentare notevolmente le quantità dei 35mila item disponibili sulla piattaforma RUBiS prima di ogni ciclo di emulazioni.

Questo approccio ha ridotto notevolmente gli errori di questo tipo che sono risultati solo nelle parti finali della simulazione e, considerandone il numero limitato, sono stati ritenuti parte lecita di uno scenario di simulazione realistico di un sito d'aste molto frequentato.

Un esempio di questo tipo di errore è disponibile nell'immagine sottostante.

Thread UserSession54: Error returned from access to <http://192.168.1.101:80/PHP/StoreBid.php?itemId=533685&userId=736436&minBid=1188.0&maxQty=1&bid=1189.0&maxBid=1190.0&qty=1>

Il caching di Laravel

Dato che ha influenzato non poco questo studio, è stato scelto di dedicare un breve paragrafo per spiegare come è strutturato il meccanismo di caching in Laravel. Essendo un framework personalizzabile sotto molti punti di vista, anche per quanto riguarda il caching Laravel offre molte scelte.

All'interno della cartella di progetto esiste un file `cache.php` (nello specifico locato nella cartella `config`) che permette la scelta di un driver che svolga tutto il lavoro di cache come potrebbe essere Memcached o Redis.

Nel nostro esperimento abbiamo lasciato il driver predefinito di Laravel: `file`.

Benché il caching delle view sia fatto automaticamente dal framework ed è possibile osservare le view cachate all'interno della directory `storage/framework/views` molti altri meccanismi di caching possono essere gestiti manualmente. Ad esempio si può definire una tabella di caching per alcuni dati del database

Si può creare una tabella di caching del DBMS inserendo nel file di configurazione qualcosa di simile a ciò che è riportato nell'immagine sottostante.

```
Schema::create('cache', function ($table) {
    $table->string('key')->unique();
    $table->text('value');
    $table->integer('expiration');
});
```

Per la maggior parte dei bisogni si può usare la façade Cache che permette di accedere in maniera molto versatile ai servizi di cache di Laravel. Alcuni usi della cache sono riportati nelle immagini sottostanti.

Si può accedere alla cache mediante il metodo GET specificando la chiave

```
$value = Cache::get('key');

$value = Cache::get('key', 'default');
```

Si possono memorizzare dati sempre specificando la chiave.

```
$value = Cache::store('file')->get('foo');

Cache::store('redis')->put('bar', 'baz', 10);
```

Si può verificare se un determinato oggetto contrassegnato da una chiave è già presente nella cache

```
if (Cache::has('key')) {
    //
}
```

Si possono modificare direttamente dei dati memorizzati nella cache accedendo direttamente ad essi tramite metodi appositi

```
Cache::increment('key');  
Cache::increment('key', $amount);  
Cache::decrement('key');  
Cache::decrement('key', $amount);
```

Così come è possibile cancellare un dato dalla cache o pulirla del tutto

```
Cache::forget('key');
```

```
Cache::flush();
```

Vi sono molti altri metodi e funzionalità in merito al meccanismo di caching in Laravel, per approfondire vi rimandiamo alla pagina della documentazione del framework: <https://Laravel.com/docs/5.4/cache> [4]

6. Oltre le prestazioni: analisi del codice sorgente

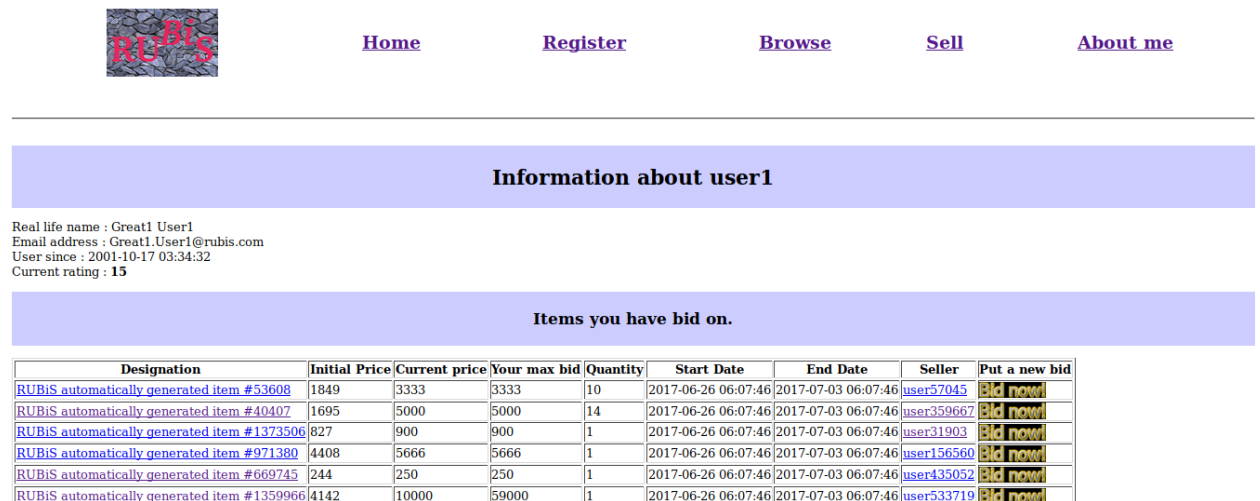
Abbiamo visto che in termini prestazionali il paradigma procedurale non ha eguali ed eccelle sulle altre versioni implementate in questo studio. Ci domandiamo adesso quali siano i vantaggi pratici nell'uso del paradigma ad oggetti e proviamo ad evidenziare tali vantaggi tramite esempi presi nel codice sorgente delle tre versioni di RUBiS.

Riteniamo che, in base all'esperienza fatta riprogrammando RUBiS, vi siano tre aspetti fondamentali da tenere in considerazione per avere un'idea generale del perché convenga adottare l'uso del paradigma ad oggetti e nello specifico l'uso di framework: anzitutto la gestione dell'output delle pagine, a seguire l'accesso ai dati nel DBMS (in particolare l'update dei record) ed infine la gestione degli errori.

Output delle pagine

La ragione forse più evidente della convenienza del paradigma ad oggetti risiede nella gestione dell'output delle pagine. Analizziamo dunque il codice sorgente che sta dietro una delle pagine con maggiore output in RUBiS, ossia quella che mostra le informazioni personali dell'utente una volta che esso si è loggato nella propria area personale.

Una tipica pagina di questo tipo, se aperta nel browser avrà l'aspetto mostrato nell'immagine sottostante



Information about user1

Real life name : Great1 User1
 Email address : Great1.User1@rubis.com
 User since : 2001-10-17 03:34:32
 Current rating : **15**

Items you have bid on.

Designation	Initial Price	Current price	Your max bid	Quantity	Start Date	End Date	Seller	Put a new bid
RUBIS automatically generated item #53608	1849	3333	3333	10	2017-06-26 06:07:46	2017-07-03 06:07:46	user57045	Bid now!
RUBIS automatically generated item #40407	1695	5000	5000	14	2017-06-26 06:07:46	2017-07-03 06:07:46	user359667	Bid now!
RUBIS automatically generated item #1373506	827	900	900	1	2017-06-26 06:07:46	2017-07-03 06:07:46	user31903	Bid now!
RUBIS automatically generated item #971380	4408	5666	5666	1	2017-06-26 06:07:46	2017-07-03 06:07:46	user156560	Bid now!
RUBIS automatically generated item #669745	244	250	250	1	2017-06-26 06:07:46	2017-07-03 06:07:46	user435052	Bid now!
RUBIS automatically generated item #1359966	4142	10000	59000	1	2017-06-26 06:07:46	2017-07-03 06:07:46	user533719	Bid now!

Tralasciando l'aspetto grafico del sito, di cui purtroppo non si è potuto alterare nulla onde evitare errori di parsing, risulta evidente come sia presente una grossa varietà di informazioni differenti da stampare, con differente formattazione e come vi sia la necessità di accedere ad una variegata mole di dati.

Nel paradigma procedurale le parti di codice di accesso ai dati, di controllo degli errori e di stampa dell'output si intrecciano dando origine ad un codice, sì efficiente, ma fortemente confusionario; come è possibile notare nell'immagine sottostante dove abbiamo riportato un estratto del codice procedurale per la pagina in esame.

```
// Get the items the user sold the last 30 days
$spastSellsResult = mysqli_query($link, "SELECT * FROM items WHERE items.seller=$userId AND TO_DAYS(NOW()) - TO_DAYS(items.end_date) < 30");
if (!$spastSellsResult)
{
    error_log("[".__FILE__."] Query 'SELECT * FROM items WHERE items.seller=$userId AND TO_DAYS(NOW()) - TO_DAYS(items.end_date) < 30' failed for getting sold items list: " .
    die("ERROR: Query failed for getting sold items list: " . mysqli_error($link));
}
if (mysqli_num_rows($spastSellsResult) == 0)
    printHTMLhighlighted(msg: "<h3>You didn't sell any item in the last 30 days.</h3>\n");
else
{
    printHTMLhighlighted(msg: "<h3>Items you sold in the last 30 days.</h3>\n");
    print("<p><TABLE border='1' summary='List of items'\n".
    "<thead>\n".
    "<tr><th>Designation</th><th>Initial Price</th><th>Current price</th><th>Quantity</th><th>ReservePrice</th><th>Buy Now</th>".
    "<th>Start Date</th><th>End Date</th>\n".
    "</thead>\n");
    while ($spastSellsRow = mysqli_fetch_array($spastSellsResult))
    {
        $itemName = $spastSellsRow["name"];
        $itemInitialPrice = $spastSellsRow["initial_price"];
        $quantity = $spastSellsRow["quantity"];
        $itemReservePrice = $spastSellsRow["reserve_price"];
        $buyNow = $spastSellsRow["buy_now"];
        $endDate = $spastSellsRow["end_date"];
        $startDate = $spastSellsRow["start_date"];
        $itemId = $spastSellsRow["id"];
        $currentPriceResult = mysqli_query($link, "SELECT MAX(bid) AS max_bid FROM bids WHERE item_id=$itemId") or die("ERROR: Query failed for getting the item current price (sold item)");
        if (mysqli_num_rows($currentPriceResult) == 0)
            die("ERROR: Cannot get the current price (sold item).");
        $currentPriceRow = mysqli_fetch_array($currentPriceResult);
        $currentPrice = $currentPriceRow["max_bid"];
        if ($currentPrice == null)
            $currentPrice = "none";
        print("<tr><td><a href='/PHP/ViewItem.php?itemId=".$itemId.">".$itemName,
        "<td>".$itemInitialPrice."<td>".$currentPrice."<td>".$quantity,
        "<td>".$itemReservePrice."<td>".$buyNow,
        "<td>".$startDate."<td>".$endDate."</td>\n");
    }
    //mysqli_free_result($currentPriceResult);
    print("</TABLE>\n");
}
```

In questo estratto è evidente come vengano effettuate due query, fatto il fetch dei risultati e stampato l'output tramite la funzione print; tutto in sequenza. Se si volesse andare a modificare un parametro di queste query si dovrebbe cercare in mezzo a tutto il codice il punto in cui la query viene eseguita e sarebbe già un processo difficoltoso. Se immaginiamo allora di dover cambiare ad esempio la formattazione di una delle tabelle presenti in output, è facile immaginare quanto possa essere complicato rintracciare i punti di interesse e non sbagliare nella formattazione del codice HTML (non a caso nella versione originale scritta in procedurale vi erano diversi errori di chiusura/apertura dei tag).

Analizziamo ora la stessa pagina nella versione ad oggetti senza framework. Ne abbiamo riportato un estratto nella pagina seguente.

La pagina in sé risulta molto più compatta ed è facile osservare i vari blocchi dedicati alla gestione degli errori, all'acquisizione dei dati e alla stampa.

Grazie ai nomi dei metodi espressivi è semplice intuire cosa una riga di codice faccia rispetto ad un'altra.

L'output della pagina viene gestito da una classe denominata `LogPage_class` che una volta inizializzata inizia a tenere conto dei tempi di caricamento e delle informazioni della pagina che dovranno essere presenti nel footer affinché il client sia in grado di fare il suo dovere. Le stesse informazioni nel codice procedurale vengono inserite manualmente per ogni pagina e, qualora fosse necessario apportare delle modifiche a tali dati, costringono il programmatore ad onerose modifiche in termini di tempo.


```

$log = new LogPage_class(basename( path: __FILE__, suffix: ".php"));
try{
    switch($_SERVER['REQUEST_METHOD']) {
        case "POST":
            //do post
            $user = User_class::LoadUserByCredential($_POST['nickname'], $_POST['password']);
            break;
        case "GET":
            //do get;
            $user = User_class::LoadUserByCredential($_GET['nickname'], $_GET['password']);
            break;
        default:
            //error
            throw new Exception( message: "please load this page from register form or via API with dataa", code: 5);
    }
    //login completed
    $item = new Items_class();
    //generating list of item-object (BID) / 0 if no item found
    $bidlist = $item->ItemsYouBid($user->getId());
    if($bidlist != 0) {
        $i = 0;
        while ($bidlist[$i]['item_id'] != NULL) {
            //loading seller user settings for every object
            $seller = User_class::LoadUserByID($bidlist[$i]['item_seller']);
            //finalizing list with this last informations
            $bidlist[$i]['seller_id'] = $seller->getId();
            $bidlist[$i]['seller_user'] = $seller->getUserName();
            $i++;
        }
    }
    //generating list of item-object (WON/SOLD/SELLING ETC) / 0 if no items found
    $wonlist = $item->ItemsYouWon($user->getId());
    $boughtlist = $item->ItemsYouBought($user->getId());
    $sellinglist = $item->ItemsYoureSelling($user->getId());
    $soldlist = $item->ItemsYouSold($user->getId());
    //comments for this user on list
    $comment = new Comment_class();
    $commentlist = $comment->LoadCommentsOnUser($user->getId());
    //printing page
    $log->UserInfoPage($user, $bidlist, $wonlist, $boughtlist, $sellinglist, $soldlist, $commentlist);
} catch (Exception $e) {
    $log->KillSession($e->getMessage());
}
?>

```

L'oggetto \$log della classe che si occupa dell'output della pagina viene utilizzato quando ormai tutti i dati ed i controlli sono stati effettuati e in una singola riga ne viene invocato il metodo (con tutti i parametri necessari ricavati precedentemente) che si occupa della generazione dell'output per quella pagina. Mostriamo il metodo chiamato sull'oggetto \$log in questione nella pagina sottostante.

Appare evidente che tale codice risulta molto più chiaro e si aggancia ad altri metodi della stessa classe che stampano le relative sezioni della pagina. Se adesso avessimo la necessità di operare delle modifiche sull'output html sapremmo in pochi istanti dove andare a trovare il codice necessario.

Per completezza riportiamo anche uno dei metodi invocati nella pagina per ricavare informazioni sugli oggetti venduti dall'utente mostrando come anche in questo caso la classe che gestisce gli oggetti si limiti a richiamare la classe che si interfaccia col DBMS (che in ultima istanza esegue le query necessarie).

Nessun tipo di output viene generato in queste classi, neppure in caso di errore. Tutto è ben distinto ed assume un ruolo specifico.

```
public function UserInfoPage(User_class $user, $itemList, $wonItemList, $boughtItemList, $sellingItemList, $soldItemList, $commentList) {
    $this->Header(Title: "RUBIS: About me");
    //general user information printing
    $tag = "<h2>Information about " . $user->getUserName() . "<br></h2>";
    $this->InfoTableHeader($tag);
    //user activity informations
    //general info
    print("Real life name : " . $user->getName() . " " . $user->getSurname() . "<br>");
    print("Email address : " . $user->getEmail() . "<br>");
    print("User since : " . $user->getCreationdate() . "<br>");
    print("Current rating : <b>" . $user->getRating() . "</b><br><p>");
    //end of general info
    $this->UserItemBid($itemList, $user);
    $this->UserItemWon($wonItemList);
    $this->UserItemBought($boughtItemList);
    $this->UserItemSelling($sellingItemList);
    $this->UserItemSold($soldItemList);
    if ($commentList == 0) {
        $tag = "<h2>There is no comment for this user.</h2>\n";
        $this->InfoTableHeader($tag);
    }
    else {
        $tag = "<h3>Comments about you.</h3>\n";
        $this->InfoTableHeader($tag);
        $this->UserCommentPosted($commentList);
    }
    $this->Footer();
}
```

Di seguito il metodo della classe che gestisce gli oggetti

```
public function ItemsYouWon($userID) {
    if(!is_int($userID))
        throw new Exception( message: "ERROR: user id for item list not valid", code: 11);
    $db = new DB_connect();
    $stmt = $db->GetListWonItems($userID);
    if($stmt == NULL)
        return NULL;
    else if ($stmt == 0)
        return 0;
    else
        return $this->GeneralItemListing($stmt);
}
```

E quello della classe che si interfaccia con il database: le query sono state mantenute identiche.

```
/**
 * @param $userID
 * @return int|mysqli_stmt
 */
public function GetListWonItems($userID){
    $this->checkInstance();
    $query = "SELECT items.* FROM bids, items
              WHERE bids.user_id=$userID
              AND bids.item_id=items.id
              AND TO_DAYS(NOW()) - TO_DAYS(items.end_date) < 30
              GROUP BY item_id";
    if($this->countRecords($query) == 0) {
        return 0;
    }
    return $this->prepare($query);
}
```

Sul framework è necessario spendere due parole in più. Laravel come già spiegato in precedenza è basato sul MVC, ogni richiesta di pagina deve essere instradata tramite Route all'interno dei file presenti nell'omonima cartella del framework. Ad ogni route può essere associata una view, un controller o un middleware (che in genere si usa su gruppi di route per effettuare controlli di vario tipo).

È stata dunque definita una route per la pagina in esame, al quale è stato agganciato un middleware per verificare l'autenticazione dell'utente e, una volta verificate le credenziali, l'accesso ad un controller per ricavare i dati tramite Eloquent; infine l'output viene gestito da una view di cui mostreremo un estratto.

È evidente che in questo caso il codice risulta ancora più strutturato e semplice da usare, obbligando allo stesso tempo il programmatore all'uso delle best practice.

Di seguito riportiamo un estratto della route

```
//all the route which wants credential
Route::group(['middleware' => 'login'], function() {
    Route::match(array('GET', 'POST'), url: 'AboutMe.php', action: 'userInfoController@AboutMe');
```

Si nota che al suo interno viene comunicato al framework che possiamo aspettarci richieste di tipo GET o POST: da qui in poi non dovremo più occuparci di tale aspetto. La

route inoltre è legata ad un gruppo di routes associate al middleware di autenticazione che ci garantirà l'autenticità delle credenziali immesse.

```
public function AboutMe(Request $request){
    $user = $request->get( key: 'nickname');
    $password = $request->get( key: 'password');
    //general user informations
    $userInfo = user::select( columns: 'id', 'firstname', 'lastname', 'password',
                                'email', 'nickname', 'creation_date', 'rating')
        ->where( column: 'nickname', $user)
        ->where( column: 'password', $password)
        ->first();
}
```

Nell'immagine soprastante è stato inserito un estratto del controller che si occupa di ricavare i dati dal database. Esso ottiene i dati di input tramite l'oggetto \$request e li usa per ricavare le informazioni dell'utente tramite il model User legato alla tabella users nel Database con metodi particolarmente espressivi: il programmatore può lavorare ignorando persino la sintassi SQL grazie all'ORM.

Alla fine di tutte le operazioni per ricavare i dati, il controller richiama la relativa view passando le informazioni necessarie alla generazione della pagina. Nella pagina successiva si vede anche un estratto della view ed è facile notare come Blade abbia una sintassi molto semplice ed intuitiva: potenzialmente si può evitare di scrivere codice PHP. Alle variabili di pagina si può accedere tramite le doppie graffe {{ \$var }} innestandole nel codice HTML. Per le iterazioni sono previsti i comuni cicli della programmazione richiamati antepponendo loro il simbolo di chiocciola

@foreach (\$var as \$value) {{ \$value->var }} @endforeach

```
return view( view: 'user_information', ['userInfo' => $userInfo,
    'title' => $title,
    'filename' => $filename,
    'itemYouBid' => $itemYouBid,
    'itemYouWon' => $itemYouWon,
    'itemYouBought' => $itemYouBought,
    'itemYouSelling' => $itemYouSelling,
    'itemYouSold' => $itemYouSold,
    'comment' => $comment
]);
```

```
@include('header')
<TABLE width="100%" bgcolor="#CCCCFF">
<TR><TD align="center" width="100%"><FONT size="4" color="#000000"><B><h2>Information about {{ $userInfo->nickname }}<br></h2></B></FONT></TD></TR>
</TABLE><p>
Real life name : {{ $userInfo->firstname }} {{ $userInfo->lastname }}<br>
Email address : {{ $userInfo->email }}<br>
User since : {{ $userInfo->creation_date }}<br>
Current rating : <b>{{ $userInfo->rating }}</b><br><p>
```

Purtroppo nella versione in Laravel abbiamo avuto poche volte occasione di scrivere l'output delle pagine in pura sintassi Blade; per evitare errori di parsing nella maggior parte dei casi sono stati inseriti blocchi PHP con le stesse funzioni print della versione ad oggetti. Sarebbe infatti bastato un solo spazio omissso per mandare in crisi il parser ed era una circostanza che si voleva a tutti i costi evitare.

Accessi al DBMS

Un'altra grande differenza tra le tre versioni è legata all'accesso al DBMS. Come precedentemente ricordato abbiamo usato MySQL per la gestione dei dati. Omettendo la parte di configurazione di Eloquent in Laravel che peraltro è molto semplice, ci concentreremo subito sulle differenze a livello di codice, citando una delle parti in cui queste sono più evidenti: l'acquisto di un oggetto.

Per ogni acquisto RUBiS svolge una serie di operazioni sul database: verifica la presenza dell'oggetto, controlla le quantità, le decrementa e qualora l'item sia esaurito lo sposta in un'altra tabella, associa infine l'item acquistato al rispettivo proprietario.

Nella versione procedurale, come è facile aspettarsi, tutte queste operazioni sono svolte in sequenza.

Nella versione ad oggetti invece, viene inizializzato un oggetto di tipo "oggetto" tramite l'id e su di esso si eseguono tutte le operazioni di controllo ed aggiornamento dei dati nel database; ovviamente, similmente a quanto fatto in precedenza, nulla di ciò che riguarda il DBMS viene invocato dentro la classe dell'oggetto ma questa richiama la relativa classe coi metodi di accesso al database.

Abbiamo riportato nella pagina seguente un estratto della versione procedurale, uno della versione ad oggetti

```

$result = mysqli_query($link, "LOCK TABLES buy_now WRITE, items WRITE", $link);
if (!$result)
{
    error_log(["__FILE__."] Failed to acquire locks on items and buy_now tables: " . mysqli_error($link));
    die("ERROR: Failed to acquire locks on items and buy_now tables: " . mysqli_error($link));
}
$result = mysqli_query($link, query: "SELECT * FROM items WHERE items.id=$itemId");
if (!$result)
{
    error_log( message: ["__FILE__."] Query 'SELECT * FROM items WHERE items.id=$itemId' failed: " . mysqli_error($link));
    die("ERROR: Query failed for item '$itemId': " . mysqli_error($link));
}
if (mysqli_num_rows($result) == 0)
{
    printError($scriptName, $startTime, title: "BuyNow", error: "<h3>Sorry, but this item does not exist.</h3><br>");
    commit($link);
    exit();
}
$row = mysqli_fetch_array($result);
$newQty = $row["quantity"]-$qty;
if ($newQty == 0)
{
    $result2 = mysqli_query($link, query: "UPDATE items SET end_date=NOW(),quantity=$newQty WHERE id=$itemId");
    if (!$result2)
    {
        error_log( message: ["__FILE__."] Failed to update item 'UPDATE items SET end_date=NOW(),quantity=$newQty WHERE id=$itemId': " . mysqli_error($link));
        die("ERROR: Failed to update item '$itemId': " . mysqli_error($link));
    }
}
else
{
    $result2 = mysqli_query($link, query: "UPDATE items SET quantity=$newQty WHERE id=$itemId");
    if (!$result2)
    {
        error_log( message: ["__FILE__."] Failed to update item 'UPDATE items SET quantity=$newQty WHERE id=$itemId': " . mysqli_error($link));
        die("ERROR: Failed to update item '$itemId': " . mysqli_error($link));
    }
}
}

```

```

    if($qty > $maxQty)
        throw new Exception( message: "ERROR: you want more than we can sell you", code: 34);
    $newQty = $item->getQty()-$qty;
    $item->UpdateItemQty($newQty);
    $item->AddThisToBuyNow($user->getId(), $qty);
    $log->BuyNowStorePage($qty);
}

```

```

public function UpdateItemQty($newqty){
    if(!is_int($newqty))
        throw new Exception( message: "ERROR: no valid new qty", code: 27);
    if($newqty<0)
        throw new Exception( message: "ERROR: you want to buy more than we can sell you.", code: 33);
    $db = new DB_connect();
    if($newqty == 0){
        //sold out now
        $db->SoldOut($this->getId());
    }
    else {
        //update
        $db->UpdateQty($this->getId(), $newqty);
    }
}
}

```

```

public function UpdateQty($itemID, $newQty){
    $this->checkInstance();
    $this->begin_transaction();
    $query = "UPDATE items SET quantity=$newQty WHERE id=$itemID";
    $this->query($query);
    $this->commit();
}

```

Per quanto concerne il framework basta un singolo estratto per rendere l'idea di come sia semplice effettuare le operazioni descritte in precedenza. Si carica la riga (o le righe) della tabella sotto forma di oggetto tramite la chiave primaria (o altre clausole) e le modifiche a tale riga avvengono con la stessa semplicità della modifica di un attributo dell'oggetto. Per completare le modifiche è necessario invocare il metodo save.

```
$item = items::find($request->itemId);  
$buyer = user::find($request->userId);  
//okay...updating item  
$item->quantity = $item->quantity - $request->qty;  
if($item->quantity == 0)  
    $item->end_date = date( format: 'Y-m-d H:i:s');  
//creating buynow row  
$buynow = new buy_now();  
$buynow->date = date( format: "Y:m:d H:i:s");  
$buynow->buyer_id = $buyer->id;  
$buynow->item_id = $item->id;  
$buynow->qty = $request->qty;  
  
$buynow->save();  
$item->save();
```

Gestione degli errori

Ultimo punto di analisi è la gestione degli errori. RUBiS è un sito pieno di form e parametri da validare; non solo per quanto riguarda l'autenticazione dell'utente ma anche per quanto concerne ID degli oggetti, quantità, ecc. Molti di questi dati vengono passati al server tramite richieste HTTP di tipo GET (ovviamente tale metodo si rende necessario per il corretto funzionamento dell'emulatore), omettendo le profonde falle di sicurezza che un utente malevolo potrebbe sfruttare ma che non sono scopo di questo studio, è previsto quantomeno un minimo controllo per quelli che sono i campi che, se non filtrati, manderebbero in crisi il DBMS anche durante il normale uso del sito.

Nella versione procedurale di RUBiS tali controlli si inseriscono direttamente all'interno del flusso di codice come evidenziato nella figura sottostante.

```

print("<p><TABLE border='1' summary='List of items'>\n".
      "<thead>\n".
      "<tr><th>Designation<th>Quantity<th>Price you bought it<th>Seller".
      "<tbody>\n");
while ($buyNowRow = mysqli_fetch_array($buyNowResult))
{
    $itemId = $buyNowRow["item_id"];
    $itemResult = mysqli_query($link, "SELECT * FROM items WHERE id=$itemId");
    if (!$itemResult)
    {
        error_log("[".__FILE__."] Query 'SELECT * FROM items WHERE id=$itemId' failed for getting the item the user bought: " . mysqli_error($link));
        die("ERROR: Query failed for getting the item '$itemId' the user bought: " . mysqli_error($link));
    }
    if (mysqli_num_rows($itemResult) == 0)
    {
        rollback($link);
        die("<h3>ERROR: This item '$itemId' does not exist.</h3><br>\n");
    }
}

```

Nella versione ad oggetti invece si è fatto ampio uso delle eccezioni. Con PHP5 infatti è stato introdotto l'uso dell' Exception Handling che ci permette di gestire gli errori in una sezione a parte rispetto al flusso del codice.

Nella figura sottostante se ne vede un esempio: l'oggetto di tipo “oggetto” viene caricato tramite ID, ma tale operazione viene inserita all'interno di un blocco try-catch.

```

try{
    switch($_SERVER['REQUEST_METHOD']) {
        case "POST":
            //do post
            $item = Items_class::LoadItemByID($_POST['itemId']);
            break;
        case "GET":
            //do get;
            $item = Items_class::LoadItemByID($_GET['itemId']);
            break;
        default:
            //error
            throw new Exception("please load this page from register form or via API with datas", code: 5);
    }
    $list = $item->BidsInfoOnThisItem();
    $log->ItemViewPage($list);
} catch (Exception $e) {
    $log->KillSession($e->getMessage());
}

```

Osservando dall'interno il metodo statico richiamato nella classe Item si nota subito che in caso di errore viene sollevata un'eccezione. Tale eccezione viene catturata nella pagina principale ed il messaggio di errore viene stampato tramite la classe che si occupa dell'output. In tal modo, se è presente un errore questo muterà l'output della pagina senza che altri dati possano essere stati stampati prima.

```

public static function LoadItemByID($id){
    $db = new DB_connect();
    if( !isset($id) || !is_numeric($id) || empty(trim($id))) {
        throw new Exception("Invalid item ID: $id<br></h2>", code: 14);
    }
    $item = new self();
    // if next method not found itemID on item then search for itemID on old items
    $db->LoadItemDatasByID($id, $item);
    return $item;
}

```


Per quanto riguarda Laravel, esso prevede una sua infrastruttura interna per la gestione degli errori che si basa sulla libreria Monolog. Sfortunatamente non vi è stato il tempo di implementare tale libreria in RUBiS; per questa ragione la gestione degli errori all'interno dei controller è rimasta abbastanza in linea con la versione procedurale (pur utilizzando le view per ritornare pagine di errore evitando altri tipi di output indesiderati).

La documentazione di Laravel comunque si dilunga in merito alla sua struttura di error e log handling. Tali informazioni sono presenti al seguente link: <https://Laravel.com/docs/5.4/errors> [4]

7. Conclusioni

Abbiamo visto in dettaglio i punti di forza e di debolezza delle tre versioni di RUBiS; possiamo concludere che il paradigma procedurale resta indubbiamente il migliore dal punto di vista prestazionale anche se il framework, grazie al grande lavoro di ottimizzazione, riesce ad essere una valida alternativa su un buon range di carico del server. Tuttavia il paradigma procedurale è ormai alla stregua dell'inutilizzabilità per quanto riguarda l'innovazione nella scrittura del codice, specie in un periodo storico in cui le prestazioni dei calcolatori in rapporto al costo tendono ad abbassarsi sempre di più e si hanno a disposizione macchine più prestanti. La versione ad oggetti infine è stata quella con il rapporto prestazioni - manutenibilità del codice peggiore ma ci riserbiamo di dire che con un grande lavoro di ottimizzazione probabilmente avrebbe delle performance di gran lunga migliori di quelle ottenute in questo studio.

L'uso dei framework sta prendendo piede però proprio per evitare che il programmatore sia costretto a costruirsi da sé un'intera infrastruttura e gestirne tutte le falle e le ottimizzazioni; possiamo dunque concludere che, ad oggi, il framework resta l'alternativa migliore per il web development mentre il paradigma procedurale può ancora rilevarsi utile in tutte quelle applicazioni commerciali a basso budget in cui le prestazioni dei calcolatori sono minime o, viceversa, in quei grossi progetti nei quali nonostante si abbiano calcolatori molto prestanti è necessaria una spinta in più per raggiungere gli obiettivi prefissati [6]

Bibliografia

- [1] E. Cecchet, «Performance and Scalability of EJB Applications».
- [2] R. University, «sito ufficiale RUBiS,» [Online]. Available:
<http://RUBiS.ow2.org/index.html>.
- [3] G. K. Babbling, «RUBiS Workload: Simple Installation Guide,» [Online]. Available:
<https://sanifool.wordpress.com/2012/09/03/RUBiS-workload-simple-installation-guide/>.
- [4] Laravel, «documentazione online,» [Online]. Available: <https://Laravel.com/docs/5.4/>.
- [5] html.it, «guida all'uso di Laravel,» [Online]. Available:
<http://www.html.it/guide/Laravel-la-guida/>.
- [6] matlab, «documentazione online,» [Online]. Available:
<https://it.mathworks.com/help/matlab/>.
- [7] S. S. J. a. D. Sunil Dutt, «Object Oriented Vs Procedural Programming in Embedded Systems».