Kasdi Merbah University - Ouargla
Faculty of New Technologies of Information and Communication
Department of Computer Science and Information Technology

Dissertation
ACADEMIC LICENCE

Domain: Mathematic and Computing Science
Field: Computing Science
Specially: Computer Systems

Theme

Enterprise Web Application Architectures

Submitted by: DEBBAH Youcef
KHENNOUR Mohammed Elmahdi

Supervise by: ZGA Adel

Academic Year: 2016/2017
We would love to thank

- Our parents for all the support that they gave us.
- Professors of our college.
- Our supervisor Adel ZGA for all the guideline and tips.
- Cervex manager Mr Nabil, who introduced us to software business world and overlooked all those over-delay deliveries and glitches we caused.
Summary

With the growth of web applications, companies have started using web applications to manage their business in form of an Enterprise Application Software (EAS) across different domains like eCommerce, healthcare, eLearning and many others.

Different software development companies use different architectures while developing such web applications and the question that we are trying to answer is: what is the best architecture for an enterprise web application?

In fact to know what architecture you should use do not think about the architecture itself! Instead, you must separate your application to different layers then deploy (distribute) these layers on the available physical tiers in your environment, taking into consideration the requirements of each layer the characteristics of each tier. By the time that we finish this step the general architecture should be clear, the rest is choosing the right patterns to use within each layer, the final design that we get from these steps is what we call an architecture.

To illustrate this we have developed an enterprise web application named “Cervex” which provide services to a software development company named “Cervex”, these services include messaging, clients applies management and an embedded blog. Using the above-mentioned approach, we end up using a 3-tier architecture to implement this application, after that we used this project as a use case in our study.

Therefore, there is no such thing as the perfect enterprise web site architecture because such architecture depends on numerous factors like project budget, developer’s perspective, etc. However, by following a good methodology we can find the relatively best architecture to suit a given application; although implementing such an architecture can be complex and expensive, this approach is more cost effective on the long run.

Key words: Web application, Single page application, N-Tier Deployment, Pattern, Layers, Tiers, Scaling.
الملخص

في السنوات الأخيرة ومع تطور تطبيقات الويب بدأت المؤسسات والشركات باعتمادها لتسير أعمالها في شكل تطبيقات ويب خاصة بها تعني: (EAS) في مختلف المجالات مثل التجارة الإلكترونية والرعاية الصحية والتعليم عن بعد وغيرها، وتستخدم هذه الشركات المطورين لتطبيقات الويب تصاميم مختلفة لتطبيقات الويب الخاصة بها، وبقي السؤال مطروح، ما هو أحسن تصميم لهذه التطبيقات؟

في الحقيقة لمعرفة أحسن مخطط لاستعماله يجب عدم التفكير في المخطط بحد ذاته! عوضا عن ذلك يجب تقسيم البرمجية إلى عدة طبقات منطقية ومن ثم توزيعها على المستويات المادية المتوفرة أخذا الاعتبار مطابقة كل طبقة (Patterns) وخصائص كل مستوى، عند انتهاء هذه الخطوة سيصبح المخطط العام واضحا، يبقى فقط اختيار الأطراط المناسبة لكل طبقة، بابتعاد كل هذه الخطوات سنحصل على تصميم عام يدعى مخطط تطبيق الويب للمؤسسات.

لتصحيح ذلك أكثر قمتنا بتطوير تطبيق ويب للمؤسسات يدعى "سرفسكس" أو "Cervex" كعينة لدراسة، يوفر هذا التطبيق "Cervex" خدمات لشركة تطوير برامج اسمها "Cervex"، من بين هذه الخدمات: خدمة المراسلة وإدارة طلبات البرمجيات إضافة إلى احتوائه على مدونة خاصة بالشركة، مستغلين المقاربة المذكورة أعلاه توصلنا إلى مخطط ثلاث طبقات استعملنا في تطوير هذا التطبيق.

يوضح لنا أنه لا يوجد مخطط مثالي إذ أن مثل هذه المخططات تختلف باختلاف عوامل عدة منها: ميزانية المشروع، وجهة نظر المطورين وغيره من العوامل، ولكن باتباع منهجية صحيحة تستطيع إيجاد الحل الأحسن نسبيا لتطبيق معين. ومع أن تنفيذ مثل هذه المهنية يمكن أن يكون معقدا ومكلفا لكنه يبقى الأكثر فعالية على المدى الطويل.
Introduction

In the last few years, Ecommerce is growing constantly, and even small business owners start investing in the web according to Americommerce [1]: Ecommerce is a $220 Billion industry in the U.S. alone growing by nearly 17% a year.

Enterprises required a cross platform, scalable and easy to ship Enterprise application software (EAS) for Ecommerce, and this was the birth of enterprise web application, later this new type of software has been adopted to many other use cases like: social media, healthcare, ELearning and many others.

Moving this huge amount of data and processing to the web was not without a price, it became clearly that the old and general client server architecture that websites and small web applications used was not practical for enterprise web applications that needed to be reliable, secured and scalable.

Many software and hardware architectures has been adopted to solve this dependencies but this leave to web developers a wide range decisions to take concerning their system design.

This report contain three major sections: the first one will cover some patterns and architectures that can be used by developers working on an enterprise web application, and try to answer the main question: what is the best architecture of an enterprise web application? The second section cover the conception of an enterprise web application that we have developed and the third section contains the implementation details of this web application.
Section 1: Designing an enterprise web application

1 Enterprise software

Enterprise application software (EAS), also known as enterprise software or enterprise application, is a computer software used to satisfy the needs of an organization rather than individual users. Such organizations would include businesses, schools, interest-based user groups, clubs, charities, or governments. [2]

Enterprise applications are about the display, manipulation, and storage of large amounts of often-complex data and the support or automation of business processes with that data. Examples include reservation systems, financial systems, supply chain systems, and many others that run modern business. Enterprise applications have their own particular challenges and solutions, and they are different from embedded systems, control systems, telecoms, or desktop productivity software.

2 Web applications

A web-based application is any application that uses a website as a user interface.

Users can easily access the application from any computer connected to the Internet using a standard browser. These applications offer a range of business advantages over traditional desktop applications including:

2.1 Accessible anywhere

Unlike traditional applications, web systems are accessible anytime, anywhere and via any PC with an Internet connection. This puts the user firmly in charge of where and when they access the application.

2.2 Easily customizable

The user interface of web-based applications is easier to customise than is the case with desktop applications. This makes it easier to update the look and feel of the application or to customise the presentation of information to different user groups.

2.3 Accessible for a range of devices

In addition to being customisable for user groups, content can also be customized for use on any device connected to the internet. This includes the likes of PDAs, mobile phones and tablets.

2.4 Improved interoperability

It is possible to achieve a far greater level of interoperability between web applications than it is with isolated desktop systems. For example, it is much easier to integrate a web-based shopping cart system with a web-based accounting package than it is to get two proprietary systems to talk to each other.

Because of this, web-based architecture makes it possible to integrate enterprise systems rapidly, improving workflow and other business processes. By taking advantage of internet technologies, we get a flexible and adaptable business model that can be changed according to shifting market demands.
2.5 Easier installation and maintenance

With the web-based approach installation and maintenance becomes less complicated too. Once a new version or upgrade is installed, on a host server, all users can access it straight away and there is no need to upgrade the PC of each potential user.

Rolling out new software can be accomplished more easily, requiring only that users have up-to-date browsers and plugins. As the upgrades are only performed by an experienced professional to a single server, the results are also more predictable and reliable.

2.6 Adaptable to increased workload

Increasing processor capacity also becomes a far simpler operation with web-based applications. If an application requires more power to perform tasks, only the server hardware needs to be upgraded.

The capacity of web-based software can be increased by “clustering” or running the software on several servers simultaneously. As workload increases, new servers can be added to the system easily.

For example, Google runs on thousands of inexpensive Linux servers. If a server fails, it can be replaced without affecting the overall performance of the application.

2.7 Increased Security

Web-based applications are typically deployed on dedicated servers, which are monitored and maintained by experienced server administrators. This is far more effective than monitoring hundreds or even thousands of client computers as is the case with desktop applications.

2.8 Flexible core technologies

Many core technologies can be used for building web-based applications, depending on the requirements of the application. The Java-based solutions (Java EE) from Sun Microsystems involve technologies such as JSF, EJB and Servlets.

Microsoft .NET platform uses Active Server Pages, SQL Server and .NET scripting languages.

Platforms powered by PHP and MySQL, are best suited for smaller websites and lower budget applications.

Other frameworks like Ruby on Rails (ROR) and Django (a Python Web framework) are also available options. With that being said, it looks like there is at least one suitable web platform for all developers.

3 Basic concepts

3.1 Pattern

Christopher Alexander says, “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that
problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”. Even though Alexander was talking about patterns in buildings and towns, what he says is true about object-oriented design patterns. Our solutions are expressed in terms of objects and interfaces instead of walls and doors, but at the core of both kinds of patterns. [3]

Although there is also patterns for non-object-oriented designs, we will focus our attention on object-oriented design patterns because modern web frameworks are based on object-oriented programming languages.

3.2 Module

Modules are reusable software components that form the building blocks of applications. Modularity satisfies some very important design goals, perhaps the most important of which is simplicity. [4]

When we design an application with many interdependencies between different parts, it becomes more difficult to understand the impact of our changes across the whole system.

If we design parts of a system to a modular interface contract instead, we can safely make changes without having a deep understanding of all of the related modules.

Another important goal of modularity is the ability to reuse our module in other applications.

Well-designed modules built on similar frameworks should be easy to transplant into new applications with few (if any) changes. By defining a standard interface we will go a long way toward building an application that is easy to extend and maintain and easy to reassemble into different forms in the future.

3.3 Separation of Concerns

Separation of concerns is the idea that each module or layer in an application should only be responsible for one thing and should not contain code that deals with other things. Separating concerns reduces code complexity by breaking a large application down into many smaller units of encapsulated functionality. [4]

It is easy to confuse separation of concerns with employing modules for the construction of our application, but separation of concerns also implies the layering of functionality in our application. For example, n-tier architecture and MVC architectures are the result of separating concerns across the entire application, rather than at the individual module level. The goal of MVC and related patterns is to separate data management from presentation.

Separation of concerns can be expressed as functions, modules, controls, widgets, layers, tiers, services, and so on. The various units of concern vary from one app to the next, and each different app may use a different combination.
4 Logical layers of an enterprise web application

Layering is a technique commonly used to break apart a complicated software system; this is done by dividing the principal subsystems in the software to some form of layer cake, where each layer rests on a lower layer. In this scheme, the higher layer uses various services defined by the lower layer, but the lower layer is unaware of the higher layer. Furthermore, each layer usually hides its lower layers from the layers above, so layer 4 uses the services of layer 3, which uses the services of layer 2, but layer 4 is unaware of layer 2.

Layering is an important technique, but there are downsides:

- Layers encapsulate some, but not all, things well. As a result, sometimes we get cascading changes. The classic example of this in a layered enterprise application is adding a field that needs to display on the UI, must be in the database, and thus must be added to every layer in between.
- Extra layers can harm performance. At every layer things typically need to be transformed from one representation to another. However, the encapsulation of an underlying function often gives us efficiency gains that more than compensate. For example: a layer that control transactions can be optimized, and will then make everything faster.

However, the hardest part of a layered architecture is deciding what layers to have and what the responsibility of each layer should be.

Most common layers separation is the one imposed by enterprise frameworks like: ASP.net and Java EE, which define three principal layers in addition to a data store layer for persisting data, of course we can make our own separation using these frameworks but using the recommended separation is more effective since it is supported out of box by most frameworks.

![Figure 1 Different layers of an enterprise web application](image)

4.1 Presentation layer

The presentation layer provides the application's user interface (UI). In our case an HTML page with CSS style, JavaScript code, images, fonts, etc.

However, this is how our view looks from a low-level perspective; in reality, we would write the UI as components, to reduce its complexity and reduce repeated code; when it comes to components designing there is many patterns to choose from.
4.1.1 MVC Design Pattern

MVC is a software design pattern, introduced in the 1970s. It impels a bifurcation of concerns, which means domain model and controller logic are separated from the user interface (view). As a result, maintenance and testing of the application becomes simple and easy. MVC design pattern divides an application into three major aspects: model, view, and controller [9]

**Model**: The model represents a collection of classes that explains the business logic i.e. business model and the data model (data access operations). It also defines the business rules for data means as how the data can be altered and manipulated.

**View**: The view represents the user interface components such as CSS, jQuery, HTML, etc. View displays the data that is received from the controller as the outcome. This also changes the model(s) into the user interface.

**Controller**: The responsibility of the controller is to process incoming requests. It gets the input from users via the view, and then processes the user's data through the model, passing back the results to view. It normally acts as a mediator between the view and the model.

4.1.2 MVP Design Pattern

The MVP pattern is similar to the MVC pattern, wherein the presenter replaces the controller. This pattern divides an application into three major aspects: model, view, and presenter. [9]

**Model**: The model represents a collection of classes that explains the business model and the data model. It also defines the business rules for data means as how the data can be altered and manipulated.

**View**: The view represents the user interface components such as CSS, jQuery, HTML, etc. View displays the data that is received from the presenter as the outcome. This also changes the model(s) into the user interface.

**Presenter**: The presenter is responsible for addressing all user interface events on behalf of the view. It receives input from users via the view, and then process the user's data through the model that passes the results back to the view. The view and the presenter are completely separated and unlike the view and the controller, they communicate to each other by an interface. The presenter also does not handle the incoming request traffic like controller.
4.1.3 MVVM Design Pattern

MVVM pattern supports two-way data binding between view and view model. This allows automatic propagation of changes, inside the state of view model to the View. Generally, the view model utilizes the observer pattern to inform changes in the View Model to the Model.\[9\]

Model: The model represents a collection of classes that explains the business model and the data model. It also defines the business rules for data means as how the data can be altered and manipulated.

View: The view represents the user interface components such as CSS, jQuery, HTML, etc. View displays the data that is received from the view model as the outcome. This also changes the model(s) into the user interface.

View Model: The view model is responsible for displaying methods, commands, and other functions that assist in maintaining the state of the view, manipulating the model as the result of actions on the view, and triggering the events in the view itself.

4.2 Business layer

The business layer (also called domain layer) is where problem-solving code reside. To manage this layer we need an application server that provide functionality like security, transaction management, interceptors ...etc. and without such a server, we would have to add an additional layer (services layer) to implement these functionalities.

4.2.1 Domain Model

Business layer contains all the complex logic of course, complex logic is where objects come in, and the object-oriented way to handle this problem is with a Domain Model. With a Domain Model, we build a model of our domain, which, at least on a first approximation, is organized primarily around the nouns in the domain. Thus, a leasing system would have classes for lease, asset, and so forth. The logic for handling validations and calculations would be placed into this domain model, so shipment object might contain the logic to calculate the shipping charge for a delivery. There might still be routines for calculating a bill, but such a procedure would quickly delegate to a Domain Model method.

4.2.2 Service layer

An old approach in handling business logic is to split the business layer in two. A Service Layer that is placed over an underlying Domain Model (or other business logic model), the Presentation logic interacts with the Domain Model purely through the Service Layer, which acts as an API for the application.\[5\]
The Service Layer is also responsible for functionality like transaction managements, security, etc.

Most modern frameworks (like Java EE) encourage the use of an application server that work as a replacement for the Service layer, and save us from writing a lot of functionality like transaction management and security checks, so technically the layer is still there but application developers don’t have to create it, instead a minor configuration is needed.

4.3 Data access layer

The data store layer provides access to external systems such as databases or LDAP servers that can persist data. What software will manage this layer depends on Data store type, in case of a relational database; we would use an Object-relational mapping tool (O/RM) to map our entities cases and object to database tables and rows.

4.3.1 Pooling pattern

The Pooling pattern describes how expensive acquisition and release of resources can be avoided by recycling the resources no longer needed. [6]

![Sequence diagram that describe Pooling pattern](image)

Figure 5 Sequence diagram that describe Pooling pattern [6]
There is many different pooling patterns, for example: Thread pooling, Memory pooling and Object pooling, but these patterns tend to be used by the underlying systems of our web application like a database management system (DBMS) or operating system (OS), connection pooling is a different, and generally it used once per application (same implementation for the whole application).

**Connection Pooling:** Opening a connection to a database is a time-consuming process. For short queries, it can take much longer to open the connection than to perform the actual database retrieval. Consequently, it makes sense to reuse Connection objects in applications that connect repeatedly to the same database. This section presents a class for connection pooling: pre-allocating database connections and recycling them as clients connect.

## 5 Physical tiers of an enterprise web application

To make the discussion bellow less abstract and more realistic the data store layer is explicitly referred as a database since it is the usual implementation for this layer, and we are including an application server to the environment thus there is no need for the extra services layer.

### 5.1 Distributed and Non-distributed Deployment

When creating a deployment strategy for our application, we have first to determine if we will use a distributed or a non-distributed deployment model. Non-distributed are more suited for a simple intranet web application that run on a local server, or any other web application that can be only accessed by finite set of users. While distributed deployment are better to be used with more complex applications that must be optimize for scalability and maintainability. \[7\]

### 5.2 Non-distributed Deployment

In a non-distributed deployment, all of the functionality and layers reside on a single server except for data storage functionality\[^7\] as shown in Figure 6.

#### 5.2.1 Advantages

- Easy to setup and configure
- Simplify and minimize the number of physical servers required
- Minimizes the performance impact of communication between layers

#### 5.2.2 Disadvantage

- one layer can negatively affect all of the other layers when it is under heavy utilization
- The servers must be generically configured and designed around the strictest of operational requirements
- The servers must support the peak usage of the largest consumers of system resources

![Figure 6 Non-distributed server architecture](image-url)
• Reduces the overall scalability and maintainability because all the layers share the same physical hardware

5.3 Distributed Deployment

In a distributed deployment, the layers of the application reside on separate physical tiers. Tiered distribution organizes the system infrastructure into a set of physical tiers to provide specific server environments optimized for specific operational requirements and system resource usage. It allows us to separate the layers of an application on different physical tiers [7], as shown in the example in Figure 7.

![Figure 7 Separation of presentation layer in an independent physical tier](image)

A distributed approach allows us to configure the application servers that host the various layers in order to best meet the requirements of each layer. However, because the primary driver for optimizing component deployment is to match a component's resource consumption profile to an appropriate server, this implies that a direct mapping of layers to tiers is often not the ideal distribution strategy.

Multiple tiers support multiple environments. We can optimize each environment for a specific set of operational requirements and system resource usage. We can then deploy components onto the tier that most closely matches their resource needs to maximize operational performance and behaviour. The more tiers we use, the more deployment options we have for each component. Distributed deployment provides a more flexible environment where we can more easily scale out or scale up each physical tier as performance limitations arise, and when processing demands increase.

However, adding more tiers adds complexity, deployment effort, and cost.

Another reason for adding tiers is to apply specific security policies. Distributed deployment allows us to apply more stringent security to the application servers; for example, by adding a firewall between the Web server and the application servers, and by using different authentication and authorization options.

Several common patterns represent application deployment structures found in most solutions. When it comes to determining the best deployment solution for an application, we need first to identify the common patterns. Once we have a good understanding of the different patterns, we can then consider scenarios, requirements, and security constraints to choose the most appropriate pattern.
5.3.1 Client-Server Deployment
This pattern represents a basic structure with two main components: a client and a server. In this scenario, the client and server will usually be located on two separate tiers. Figure 8 represents a common Web application scenario where the client interacts with a Web server. \[7\]

![Figure 8 Simple client/server deployment](image)

This client/server pattern is suitable when developing a client that will access an application server, or a stand-alone client that will access a separate database server.

5.3.2 N-Tier Deployment
The n-tier pattern represents a general distribution pattern where components of the application are separated across one or more servers. Commonly, we will choose a 2-tier, 3-tier, or 4-tier pattern as described in the following sections. While we will often locate all of the components of a layer on the same tier, this is not always the case. Layers do not have to be confined to a single tier—we can partition workloads across multiple servers if required. For example, one may decide to have side-by-side tiers that contain different aspects of our business logic. \[7\]

5.3.3 2-Tier Deployment
Effectively this is the same physical layout as the client/server pattern. It differs mainly on the ways that the components on the tiers communicate. In some cases, as shown in Figure 9, all of the application code is located on the client, and the database is located on a separate server. The client makes use of stored procedures or minimal data access functionality on the database server. \[7\]

![Figure 9 2-tier deployment with all the application code located on the client](image)

The 2-tier pattern is a good option if we are developing a client that will access an application server, or a stand-alone client that will access a separate database server.
5.3.4 3-Tier Deployment
In a 3-tier design, the client interacts with application software deployed on a separate server, and the application server interacts with a database that is located on another server, as shown in Figure 10. This is a very common pattern for most Web applications and Web services, and sufficient for most general scenarios. We might implement a firewall between the client and the Web/App tier, and another firewall between the Web/App tier and the database tier.\textsuperscript{[7]}

![Figure 10 3-tier deployment with the application code on a separate tier](image1)

We should consider 3-tier pattern if we developing an Intranet-based application where all servers are located within the private network or an Internet based application where security requirements do not prevent us from implementing business logic on a public Web or application server.

5.3.5 4-Tier Deployment
In this scenario, shown in Figure 11, the Web server is physically separated from the application server. This is often done for security reasons, where the Web server is deployed into a perimeter network and accesses the application server located on a different subnet. In this scenario, we might implement a firewall between the client and the Web tier, and another firewall between the Web tier and the application or business logic tier.\textsuperscript{[7]}

![Figure 11 4-tier deployment (Presentation and Business Logic on separated tiers)](image2)

We should consider 4-tier pattern if security requirements dictate that business logic cannot be deployed to the perimeter network, or we have an application code that makes heavy use of resources on the server and we want to offload that functionality to another server.

5.3.6 Scale Up and Scale Out
Scaling is a critical design consideration. Whether we are planning to scale out our solution through a load-balanced cluster or a partitioned database, we must ensure that our design supports the option we choose. In general cases, when we scale an
application, we can choose from and combine two basic choices: scale up (get a better tier) and scale out (get more tiers). \[7\]

5.3.6.1 Scale up
With the scale up approach, we add hardware such as processors, RAM, and network interface cards (NICs) to our existing servers to support increased capacity. This is a simple option and can be cost-effective up to a certain level because it does not introduce additional maintenance and support costs. However, beyond a certain threshold, adding more hardware to the existing servers may not produce the desired results, and getting the last 10% of theoretical performance from a single machine though upgrades can be very expensive. \[7\]

For an application to scale up effectively, the underlying framework, runtime, and computer architecture must scale up as well. When scaling up, we have to consider which resources are limiting application performance. For example, if it is memory bound or network bound, adding CPU resources will not help. \[7\]

5.3.6.2 Scale out
With the scale out approach, we add more servers, use load balancing, and clustering solutions. In addition to handling additional load, the scale out scenario also mitigates hardware failures. If one server fails, there are additional servers in the cluster that can take over the load. For example, we might have multiple load-balanced Web servers in a Web farm that host the presentation and business layers. Alternatively, we can split our business logic physically, and use a separate load-balanced middle tier for that logic while hosting the presentation layer on a load-balanced front tier. If our application is I/O constrained and we must support an extremely large database, we might partition our database across multiple database servers. In general, the ability of an application to scale out depends more on its architecture than on the underlying infrastructure. \[7\]

5.3.7 Single page application
The shift came around the year 2000 when Flash and Java applets started to gain a lot of attention. We could (and still can) host an embedded Flash or Java object inside a single web page and it can give us a hosting environment for all the user interaction. Working with Flash, Java Applets or Silverlight resembles working on a native application and helps to create a richer user experience. On the other hand, Flash and Java have major flaws: they are third-party browser plug-ins that need stand-alone installation and security consideration. Moreover, it does not feel natural to build web applications with plug-ins. \[8\]

Another way of creating an application within a single web page was to create a web page with an embedded iframe HTML element. This solution imposed security problems (cross-domain issues, for example), and it wasn’t good enough since we still had the same refresh problem but this time only in one section of our web page (the embedded iframe). Moreover, as a developer you had to maintain both the hosting application and the hosted application and had to find ways to communicate between the hosting and hosted applications. That and other problems made the iframe solution very problematic and developers preferred to avoid it if they could. \[8\]
JavaScript, as opposed to the plug-ins discussed previously, can be executed by the browser, no need for additional runtime like Flash, Java, and .NET. JavaScript does not introduce extra security concerns like browser plug-ins. However, JavaScript was not mature enough, and the browser render and JavaScript engines were not fast enough until recently. Witch helped plug-ins like Flash to be widely adopted and created a bad name for JavaScript.

Now with JavaScript engines being way faster and capable, even in mobile devices a new type of web application adopted these web applications are known by Single page applications.

A single page application (SPA) is a web application that uses only one HTML web page as a shell for all the application’s web pages and whose end-user interactions are implemented by using JavaScript, HTML, and CSS. Most of the SPA development is done on the front end as opposed to traditional web applications that rely heavily on web server interactions and that reload new web pages whenever navigation occurs. SPAs resemble native applications in their behaviour and development but they run inside a browser process as opposed to native applications, which run in their own process.

![Figure 12 Single page application](image)

So we’ve moved most of the business layer to the client, and embedded the rest with Data layer, now the server is not responsible for generating UI, instead it contain critical business logic, it also work as a gateway to Data store layer.

**Section 2: Modelling**

In this section we are going to discuss the conception of our Enterprise Web Application "Cervex" which is a website for a software development company that allows company customers to contact, apply for software and observe its state, it also have an embedded blog.

Our modelling is based on a four-layer model; we have in total three principal layers and a relational database for persisting data

---

1 Presentation layer

1.1 System overview

In this diagram, most functionality requires user authentication or administrator authentication, users can communicate with administrators, apply for a software and observe its state or comments in the blog, all this under administrators observation.
Figure 13 System overview
1.2 Use case diagram

1.2.1 Messaging system

In this Messaging system, a user can send a message to an administration and an administrator can send a message to a user, they also can manage their messages by showing it, send it, or delete it, all these functionality requires authentication.

Figure 14 Messaging System
1.2.2 Article Delivery System

In this system, a user can apply for an article by sending a demand, after a new article is created by the admin for this demand, the user can observe its status.

An administrator could also delete, update or show more detail of a specific article, all these functionality requires authentication.

Figure 15 Article delivery system
1.2.3 Authentication System

In this system, we have two types of authentications User authentication and Administration authentication, both can logout. In User authentication, new user sign-up and old user sign-in and both are user who could manage their profile and comment on blogs. In Administration authentication, Administrator manages user's account and delete them if necessary or create articles in Blog, while users and visitors can comment on blog articles.
2 Business layer

2.1 Business model

The Business model handle all operations in the system, like sending a message, posting a blog article, etc. the presentation layer use the interfaces declared in this package as an API to access to these functionality.
2.2 Sequence Diagrams:
We are going to represent the life cycle of three major functionality of Cervex system:

- **Login system**: users in both roles (admin or user) need login to access protected pages.
- **Articles system**: it shows how a user can apply for an article and how an admin accept it and create new article of it.
- **Messaging system**: it shows how users (in any role) can communicate with each other by sending messages.

![Figure 18 Authentication](image18.png)

![Figure 19 Sending demand and creating new article out of a demand](image19.png)
3 Data access layer

3.1 Object-Relation Mapping

The technique of bridging the gap between the object model and the relational model is known as object-relational mapping often referred to as O-R mapping or simply ORM. The term comes from the idea that we are -in some way- mapping the concepts from one model onto another, with the goal of introducing a mediator to manage the automatic transformation of one to the other. [12]

Therefore, ORM is the process of mapping data held in business (domain) model objects to database tables automatically. It relieves us from the task of writing low-level and complex JDBC code to persist objects into a database. [13]

3.1.1 ORM Advantages [10]

- Facilitates implementing the Domain Model pattern. In short using this pattern means that we model entities based on real business concepts rather than based on our database structure. ORM tools provide this functionality through mapping between the logical business model and the physical storage model.

- Huge reduction in code. ORM tools provide a host of services thereby allowing developers to focus on the business logic of the application rather than repetitive CRUD (Create Read Update Delete) logic.
• Changes to the object model are made in one place. Once we update our object definitions, the ORM will automatically use the updated structure for retrievals and updates. There are no SQL Update, Delete and Insert statements strewn throughout different layers of the application that need modification.

• Rich query capability. ORM tools provide an object oriented query language. This allows application developers to focus on the object model and not to have to be concerned with the database structure or SQL semantics. The ORM tool itself will translate the query language into the appropriate syntax for the database.

• Navigation. We can navigate object relationships transparently. Related objects are automatically loaded as needed. For example if you load a Demand and you want to access its Article, you can simply access Demand. Article and the ORM will take care of loading the data for you without any effort on your part.

• Data loads are completely configurable allowing you to load the data appropriate for each scenario. For example in one scenario you might want to load a list of Demands without any of its child / related objects(Files), while in other scenarios you can specify to load a Demand, with all its child (Files), etc.

• Concurrency support. Support for multiple users updating the same data simultaneously.

• Cache management. Entities are cached in memory thereby reducing load on the database.

• Transaction management and Isolation. All object changes occur scoped to a transaction. The entire transaction can either be committed or rolled back. Multiple transactions can be active in memory in the same time, and each transactions changes are isolated from one another.

• Key Management. Identifiers and surrogate keys are automatically propagated and managed.

3.1.2 ORM Disadvantages[11]

• Queries cannot be optimized.

• Queries select more data than needed, things get slower, more latency.

• Compiling queries from ORM code is slow.

• SQL is more powerful than ORM query languages.

• Database abstraction forbids vendor specific optimizations.

22
3.2 Entities

This class model represents database structure from an object-oriented view, when we interact with the database using an ORM tool we use these classes as the building blocks. Thus instead of reading Rows from Tables, we read Objects from entities, an entity could represent a single table of many tables, and of course, we can do all kinds of operations on those entities like add, delete and join operations.

Stereotypes:

<<Entity Bean>>: classes with this stereotype are entities that represent at least one table in the database, the main difference between these entities and regular classes is the auto-generated configuration in the source code, these configuration can then be read by the ORM tools, and then being used for example to convert rows returned by DBMS to entities objects.

<<ORM Persistable>>: classes with this stereotype are converted to Entities in the corresponding E/R diagram while other classes are considered helper classed and thus will not be included the generated E/R diagram.
4 Database

This E/R diagram is based on the previous entities class diagram, but from a relational perspective, the SQL queries that ORM tool generate will be based on this model.

![E/R diagram of database entities](image)

**Figure 22** Entities model

5 Thoughts

We believe that O-R mapping tools provides more flexibility in handling data as objects of different classes rather than regular tuples from relational database tables, although such tools are considered quite heavy-weight which affect performance it is still effective in medium to large projects.
Section 3: Implementation

1 Environment
We have used Java EE 7 framework with Red Hat Open JDK, the development was on Windows 7/8.1 64bit and the actual deployment in a Linux 2.6.32 (by Red Hat) environment.

The reason for choosing Windows OS for development is that we have some experience in development on windows, but choosing the server OS was something different, because the main factor here is hosting cost that is why we have choose Linux OS because it is very lightweight and free.

The web app itself is hosted in a Red Hat hosting service called Openshift, we are using two tiers to host the whole application each one has 1GB hard disk space and 512MB of RAM, we have used one as a Database Server and the second as a Web server and an Application server.

2 Modelling tool
The main modelling tool that we have used is Visual Paradigm (v14.0); we have chosen this tool due to the good integration with Java EE framework. For example: the predefined stereotype “Entity Bean” in class diagram can mark a class as an Entity, after generating the source code for this class the any needed metadata will be automatically added to the generated code, this metadata can be used by ORM tool. In addition, Visual Paradigm comes with a free month trailer, which is enough to pass the modelling phase and generate the needed source code.

3 Implementing the layers
So after all that talk about architecture how we are going to implement it? First we need to choose the software that we will use, then write each layer separately, since every layer depend only on the underlying layer and give services only to the higher layer, we can implement our application from Data store layer to Presentation layer

Since we wanted a free Java EE we had two main choices TomEE (by Apache) or Wildfly (by Red Hat), and we end up using Wildfly because has used on real production server with more functionality and stability while TomEE is an easy server for beginners. Also finding a good hosting company was easier since Red Hat itself has a hosting service (Openshift).

A Java EE server is more than just a web server, and it has three main components:

- Web container: which is the actual HTTP server that server web page and manage the presentation layer.
- EJB container: which is an application server (see section 1: 4.2.2) that manage the business logic layer.
- JPA provider: which is basically an ORM tool (see section 2: 3.1) that manage the data access layer.
3.1 Data store layer

In this project, we are using a MySQL (v5.5) relational database as an implementation for this layer, but instead of using the connector to write queries, we have uploaded the connector to the Application Server, and configured the Database. Now we will access the database only through the server, which give us some advantages like connection pooling, security (we do not have to write DB username or password in the source code), loose coupling with the other layers.

3.2 Data access layer

With the help of the JPA provider we have some choices on how to deal with the Database for example we want to execute this query:

```
SELECT * FROM user WHERE user.username = 'aName'
```

We declare a field like this:

```java
@PersistenceContext(unitName = "cervex")
private EntityManager em;
```

Here "cervex" is the name of the underlying Data store layer that we configured earlier, now when the server create an object of this class, it will set (inject) this variable and we can do same thing but using JPQL (Java Persistence Query Language) instead of SQL like this:

```java
List<User> users = em.createQuery("SELECT u FROM User u WHERE u.username=:username ", User.class)
    .setParameter("username", "aName")
    .getResultList();
```

This time we will get objects of the Entity User in a list, instead of a simple result set.

The SQL generated by JPA is:

```
SELECT user0_.userID AS userID1_7_, user0_.creationDate AS creation2_7_, user0_.email AS email3_7_, user0_.enterpriseAddress AS enterpri4_7_, user0_.enterpriseName AS enterpri5_7_, user0_.password AS password6_7_, user0_.phoneNumber AS phoneNum7_7_, user0_.role AS role8_7_, user0_.username AS username9_7_, user0_.website AS website10_7_ FROM user user0_ WHERE user0_.username='aName'
```

3.3 Business layer

This layer is an ensemble of classes that implements at least one interface like this:

```java
@Stateless
public class MessageManagerBean implements MessageManager {
}
```

```java
@Local
public interface MessageManager {
}
```

@statless means that an object of this class do not care about which user is actually invoking the methods, this mean that the server can for example use the object pooling pattern (see section 1: 4.3.1) to handle invocation of methods on this class.
@Local mean this is a business interface that expose business functionality in the same tier (actually the same JVM) as the caller (which mean there is no need for RMI to invoke methods on this interface)

3.4 Presentation layer

After having all layers ready, we can now just call methods of the business model like this:

```java
@Named
@ViewScoped
public class MessagingManager implements Serializable {
    @EJB
    private MessageManager messageManager;
    // here you we can use messageManager
}
```

For the make-up language, we have used JSF, JSF files are .xhtml files that the application can parse and then return an html page, for example:

```html
<h:form id="form">
    <h:inputSecret id="passwordInput" />
</h:form>
```

Would be rendered by JSF Runtime as:

```html
<form id="form" name="form" method="post" action="/cervex/login.xhtml" enctype="application/x-www-form-urlencoded">
    <input id="form:passwordInput" type="password" name="form:passwordInput" />
</form>
```

This give us the ability to write a simple make-up at the server and get a rich web page at user side.

4 Screenshots

![Cervex Home Page](image.png)

Figure 23 Cervex home page
Figure 24 Cervex side menu

Figure 25 A user applying for a software

Figure 26 Log in panel
Conclusion

Web applications are getting more important to enterprises business more than ever, which is why enterprise web applications market is always hungry for a high performance, reliable, scalable and secured applications.

Using good architectures for large-scale enterprise applications is critical to match such a high characteristics, because it adds more performance without including any new Hardware and web site architecture because such architecture depends on numerous factors like project budget, developer’s perspective, etc. However, by following a good methodology we can find the relatively best architecture to suit a given application making teamwork more effective, also significantly reduces maintaining and upgrading cost.

I would have been better if we had developed few web applications using different architectures, so then we can run a test on them all, which would be a better comparison case, however we could not do this due to the limited time and small budget we have.

We hope that software development companies give more importance to their logical architectures and not only to their physical servers because it is actually harder to get a better performance by upgrading the Hardware while the Software is not fully profiting from this Hardware.
Figures table

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Different layers of an enterprise web application</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>MCV models</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>MVP models</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>MVVM models</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Sequence diagram that describe Pooling pattern</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Non-distributed server architecture</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>Separation of presentation layer in an independent physical tier</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Simple client/server deployment</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>2-tier deployment with all the application code located on the client</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>3-tier deployment with the application code on a separate tier</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>4-tier deployment (Presentation and Business Logic on separated tiers)</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>Single page application</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>System overview</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Messaging System</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Article delivery system</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>Authentication System</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>Business model</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>Authentication</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>Sending demand and creating new article out of a demand</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>Sending and receiving messages</td>
<td>21</td>
</tr>
<tr>
<td>21</td>
<td>Entities</td>
<td>23</td>
</tr>
<tr>
<td>22</td>
<td>Entities model</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>Cervex home page</td>
<td>27</td>
</tr>
<tr>
<td>24</td>
<td>Cervex side menu</td>
<td>28</td>
</tr>
<tr>
<td>25</td>
<td>A user applying for a software</td>
<td>28</td>
</tr>
<tr>
<td>26</td>
<td>Log in panel</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>Visitors statistics</td>
<td>29</td>
</tr>
</tbody>
</table>
Bibliography

[1] Blog article: https://www.sparkpay.com/blog/Data-Driven-Ecommerce-Infographic, 09/05/2017

[2] Article: https://www.mail-archive.com/kragen-tol@canonical.org/msg00109.html, 03/05/2017


[12] Mike Keith, Merrick Schincariol. pro jpa 2 mastering the java persistence api, Apress, 2009


Links

- Cervex home page: https://cervex-jsoftware95.rhcloud.com
- OpenShift home page: https://www.openshift.com
- Visual Paradigm home page: https://www.visual-paradigm.com
- MySQL home page: https://www.mysql.com
- Eclipse home page: https://www.eclipse.org
- JBoss tools (plugin): http://tools.jboss.org
- Wildfly home page: http://wildfly.org
- Red Hat Open JDK: https://developers.redhat.com/products/openjdk/overview