DigiCamere

DIGICAMERE S.c.a.r.l.

Appendice 4 al Capitolato Tecnico

CIG 5507332A21 - CIG DERIVATO 69285536CE

Modalità sviluppo software

DigiCamere

1 Introduzione

Il presente documento servirà a definire delle linee guida per lo sviluppo delle applicazioni nei diversi ambienti, inizialmente in ambito java e .Net e, successivamente, verrà applicato anche lato php. Le linee guida dovranno contenere indicazioni sulle versioni dei middleware da utilizzare, sulle versioni dei db, sulle librerie, sugli stili di programmazione, sulla documentazione minima da predisporre, sui sistemi applicativi e informativi a corollario dei progetti (ad esempio utilizzo di Planio come SVN, utilizzo di Planio come bug tracker applicativo, utilizzo di google sites come sistema per la condivisione degli elementi di progetto, ecc).

1.1 A chi è rivolto

E' rivolto a diverse figure:
Sviluppatori esterni (intesi come ulteriori membri del team interno)
Società esterne che sviluppano chiavi in mano
Nuovi assunti senior
Nuovi assunti junior
Colleghi.

1.2 Scopo

Sarà utile per diversi scopi.

- Divisione in cluster applicativi: vi sono necessità differenti in funzione degli ambienti, dei linguaggi, delle dimensioni dei progetti, del costo derivante da una eccessiva burocratizzazione delle procedure. Per i progetti più semplici e caratterizzati da molteplici attività di messa in produzione di piccoli elementi, sarà possibile l'eliminazione di alcuni passaggi a vantaggio della economicità degli interventi rispetto al budget di progetto.
Tutto ciò è legato pure agli SLA (Service Level Agreement), ovvero alla percentuale di disponibilità del servizio.
- Vi è l'esigenza di capire se, all'interno dello stesso progetto, vi siano delle parti critiche e altre meno (parti statiche); questo perché il rischio di disservizio, in caso di incident, cambia notevolmente.

1.3 Argomenti

a) Come si documenta (sia codice che DB, vedi più avanti procedure per lo sviluppo del codice)

b) Documenti di Macro-Analisi

c) Analisi funzionale

d) Mock up

e) Analisi tecnica

f) Piano di Test

g) Come si sviluppa

h) Come si sviluppa BENE in generale (best practices)

i) Come si sviluppa sui “Framework” (ad esempio elenco delle modalità di utilizzo di Spring, Elenco best practices, ecc)

j) Come si sviluppa “da noi” in generale (ad esempio naming convention ecc)

k) Come si sviluppa “da noi” su un progetto specifico (ad esempio per sviluppare su un certo applicativo è necessario conoscere ed usare l'architettura progettata)

l) Che librerie si usano

m) In quale ambiente si sviluppa (limiti dell'ambiente, come si configurano)

n) Come si usa SVN

o) Come si usa plan.io

p) Procedure per lo sviluppo e gestione del codice sorgente

1.4 Ciclo di vita sviluppo software

E' importante delineare il ciclo di vita completo di un software.
Quasi tutti i modelli di ciclo di vita del software prevedono una scomposizione del processo di sviluppo in insiemi di attività simili. Le distinzioni fra diversi cicli di vita si evidenziano su altri aspetti, quali:

- l'enfasi relativa che si attribuisce a ciascuna attività;
- l'individuazione degli attori specifici incaricati di ciascuna attività;
- l'ordine in cui le attività si svolgono.

In tutti i cicli di vita del software svolge inoltre un ruolo essenziale la documentazione dei prodotti delle varie sotto-attività; la stesura della documentazione viene quindi regolamentata nello stesso modo delle attività menzionate.

1.4.1 Analisi

L'analisi è l'indagine preliminare del contesto in cui il prodotto software deve inserirsi, sulle caratteristiche o requisiti che deve esibire ed eventualmente su costi e aspetti logistici della sua realizzazione; questa fase può essere scomposta in sotto-attività quali analisi di fattibilità, analisi e modellazione del dominio applicativo, analisi dei requisiti e così via. In senso più ampio si può dire che l'analisi ha lo scopo di definire (il più precisamente possibile) il problema da risolvere. Questa fase è costituita anche da raccolta dei dati tramite colloqui tra cliente/committente e relativi sviluppatori. Al termine della fase verrà creato un documento che descrive le caratteristiche del sistema, tale documento viene definito documento di specifiche funzionali.

1.4.2 Progettazione

Nell'attività di progettazione si definiscono le linee essenziali della struttura del prodotto software in funzione dei requisiti evidenziati dall'analisi, tipicamente appannaggio di un analista programmatore. Anche la progettazione può essere scomposta in sotto-attività, dal progetto architetturale al progetto dettagliato. Si può dire che la progettazione ha lo scopo di definire (a un certo livello di dettaglio) la soluzione del problema. In questa fase sarà sviluppato un documento che permetterà di avere una definizione della struttura di massima (architettura di alto livello) e una definizione delle caratteristiche dei singoli componenti (moduli).

1.4.3 Sviluppo

Lo sviluppo, detto anche implementazione o codifica del prodotto software, è la fase di programmazione vera e propria da parte di uno o più programmatori e consiste nella realizzazione di uno o più programmi in un determinato linguaggio di programmazione, benché possano essere coinvolte anche tecnologie diverse (database, linguaggi di scripting ecc...). L'infrastruttura utilizzata in tale fase è detta ambiente di sviluppo. Si può dire che lo sviluppo è la concreta realizzazione della soluzione. Nella maggior parte dei casi è possibile distinguere

almeno una sotto-attività di implementazione dei singoli moduli che costituiscono il sistema e la sotto-attività dell'integrazione di tali moduli a formare il sistema complessivo.
Tipicamente in questa fase si usano strumenti di sviluppo noti come software development kit (SDK) e sistemi di controllo versione delle varie versioni software rilasciate dai componenti del team di sviluppo (es. Subversion), mentre la documentazione relativa al codice scritto si ottiene in maniera automatica a partire dai commenti scritti all'interno del codice sorgente attraverso tool opportuni. Il prodotto finale di questa fase solitamente è un software in versione alfa seguito da uno in versione beta con feedback da parte dei tester della successiva fase di collaudo, fino al rilascio della versione ultima che soddisfa le specifiche funzionali richieste.
In questa fase è necessario definire:

a) tools da usare:
b) editor
c) maven-like (jfrog ?)
d) librerie
e) quale versione db
f) versione apache/tomcat/IIS
g) Framework (Spring)
h) scrittura codice
i) struttura progetto
j) dipendenze
k) svn
l) test/segnalazione errori
m) issue tracker (Planio)

1.4.4 Testing

Il testing o collaudo, appannaggio di uno o più tester, consiste nella verifica e validazione di quanto (misurabilità) il prodotto software implementato soddisfi i requisiti individuati dall'analisi. Anche per il testing possono essere individuate le due sotto-attività di testing dei singoli moduli e testing del sistema integrato; inoltre possono essere individuate ulteriori sotto-attività per ogni aspetto del prodotto software che interessa testare: testing funzionale, testing di performance, testing di rottura, testing di regressione, testing di sicurezza, testing, testing di accettazione, ecc... . In caso di mancato rispetto delle specifiche il software torna indietro agli sviluppatori con il compito di risolvere i problemi riscontrati attraverso anche il debugging del software. In genere le anomalie di funzionamento sono gestite tramite i cosiddetti software di ticketing (Planio in Digicamere).

1.4.5 Deployment

Il deployment o rilascio consiste nell'installazione del prodotto software che ha superato il testing nell'infrastruttura di esecuzione utilizzabile dagli utenti detta anche ambiente di produzione. A seconda della complessità di tale infrastruttura il rilascio può essere scomposto in varie sotto-attività; esso, infatti, può variare dalla semplice copia di un file, alla copia opportuna di molti file organizzati in una complessa gerarchia di directory e componenti software, eventualmente distribuiti su hardware differenti.

1.4.6 Manutenzione

La manutenzione comprende quelle sotto-attività necessarie a modifiche del prodotto software successive al rilascio, al fine di correggere ulteriori errori attraverso patch, adattarlo a nuovi ambienti operativi (migrazione) o estenderne le funzionalità. La manutenzione incide sui costi per una stima che si aggira intorno al 60% dei costi totali. Ogni modifica al software comporta

necessariamente la necessità di nuovi test, sia relativi alle nuove funzionalità eventualmente introdotte, sia mirati a verificare che le modifiche apportate non abbiano compromesso funzionalità preesistenti (collaudo di regressione).

Va fatta una precisazione, in quanto queste attività non sono sempre lineari, e questo è dovuto al fatto che vi sono i test, i quali non vengono svolti sempre in un momento preciso. Ciò è dovuto a diversi fattori, tra i quali le esigenze del cliente, una nuova implementazione, un bug e così via: ecco che quindi vi sono dei cicli, e questo processo è detto modello incrementale.

1.4.7 Modello incrementale

Per modello incrementale o modello iterativo si intende un modello di sviluppo di un progetto software basato sulla successione dei seguenti passi principali:

1. pianificazione
2. analisi dei requisiti
3. progetto
4. implementazione
5. prove
6. valutazione

Questo ciclo può essere ripetuto diverse volte, denominate "iterazioni", fino a che la valutazione del prodotto diviene soddisfacente rispetto ai requisiti richiesti.

L'utilizzo del modello incrementale è consigliabile quando si ha, fin dall'inizio della progettazione, una visione abbastanza chiara dell'intero progetto, perché occorre fare in modo che la realizzazione della generica versione k risulti utile per la realizzazione della versione k+1.

Un approccio incrementale è particolarmente indicato in tutti quei casi in cui la specifica dei requisiti risulti particolarmente difficoltosa e di difficile stesura (semi)formale. L'uso di questo modello di sviluppo favorisce la creazione di prototipi, ovvero parti di applicazione funzionanti, che a loro volta favoriscono il dialogo con il cliente e la validazione dei requisiti.

1.4.8 Documentazione software

Parallelamente alle fasi di sviluppo di cui sopra è abitudine e buona norma da parte del team di progettazione, sviluppo e testing provvedere alla redazione della corrispondente documentazione che sarà corredata al software in fase finale o durante le singole fasi stesse a supporto della fase successiva.

Di seguito sono riportate le LINEE GUIDA all'interno dei linguaggi di programmazione

2 Linee guida sviluppo Java

2.1 Package

I nomi devono essere lowercase, tipicamente iniziano con il dominio dell'azienda, il nome del progetto, il dominio applicativo e il layer applicativo.

Esempio : it.camcom.mycamera.atti.controller,it.camcom.mycamera.atti.service ecc.

2.2 Classi

I nomi devono essere camelcase

Esempio :

class DirittiPratica

class UtenteRuolo

2.3 Interfacce

i nomi devono essere camelcase

Metodi/Variabili : i nomi devono essere mixed case.

2.4 Costanti

Nomi in uppercase.

static final int DEFAULT_WIDTH

static final int MAX_HEIGHT

Selezionare nomi significativi per le classi, la selezione del nome della classe dovrebbe essere un nome breve e significativo in grado di identificare le responsabilità principale della classe. (I nomi più efficaci sono quelli che fanno riferimento al dominio che il software vuole automatizzare).

- aggiungere al nome delle classe un suffisso relativo allo strato di appartenenza (PraticaController,PraticaService,PraticaManager ecc.)
- selezionare nomi mnemonici e brevi per attributi/variabili
- selezionare nomi significativi per i metodi in modo da riuscire a identificare chiaramente il servizio che forniscono, i metodi rappresentano azioni per tanto i loro nomi dovrebbero includere un verbo. Qualora un metodo svola più azioni aggiungere le "and" o "or" nel nome , saveOrUpdate,findByNameAndCode ecc.
- evitare l'implementazione di metodi troppo lunghi (nel caso si utilizzano framework suddividere il codice in base alle responsabilità dei componenti)
- Non scrive metodi con molti parametri, se la firma del metodo contiene più di 4/5 parametri è necessario valutare se è il caso di incapsulare questi parametri in un apposita classe (DTO/Model)
- Ritornare oggetti strutturati ove necessario
- Utilizzare le parentesi graffe anche quando non è strettamente necessario

2.5 Gestioni eccezioni

1) Dichiarare le specifiche eccezioni che un metodo può lanciare (e non l'Eccezione generica)
2) Utilizzare il blocco finally per rilasciare le eventuali risorse acquisite
3) Evitare blocchi catch vuoti
4) Effettuare correttamente il wrap di un'eccezione, in modo da non perdere il suo stack trace
5) Effettuare il log di un'eccezione o rilanciarla, fare entrambe le cose solo in casi specifici
6) Documentare tutte le eccezioni in javadoc

2.6 Database/ORM

La definizione del database e delle relative tabelle/campi dovrebbe essere fatta solo dopo aver definito (aiutati da una buona analisi) le entità coinvolte e le relazioni tra esse. Dalla definizione degli oggetti (tabelle) e delle proprietà (campi) si può poi partire con la creazione (manuale o con l'uso di tool automatici) del database.

Effettuare l'operazione inversa ovvero partire da database magari datati, senza fk, senza naming convention con relazioni scarsamente definite è della causa principali di problemi che verranno fuori durante la fase di sviluppo di un applicazione sia su piattaforma Java,PHP,Ruby,Grails ecc.

E' quindi molto importante definire e rispettare una naming convention :

- lowercase per tutti i nomi di tabelle,view,campi,fk,indici ecc.
- nomi composti separati da "_" per esempio : residenza_indirizzo,residenza_cap ecc.
- corretto utilizzo delle foreign keys
- corretto utilizzo degli indici
- autoincrement

2.6.1 ORM/Hibernate

- Prestare particolare attenzione alle relazioni tra gli oggetti, valutare a seconda dei casi , della mole dati, degli utilizzo, del numero di accessi, se usare relazioni Fetch o EAGER
- Utilizzare lazy=”extra” per grandi collezioni, non recupererà tutti gli elementi a meno che non venga richiesto, si può anche usare il metodo size(), per esempio, senza recuperare elementi dal DB
- Se possibile, utilizzare il metodo load() poiché non esegue una query select fino a quando non sarà necessario.
- Valutare che tipologia di query utilizzare a seconda dei casi NamedQuery,Native,Criteria
- Valutare se usare OSIV nelle webapp poiché questo caricherà i dati solo se e quando necessario
- Utilizzare la modalità read-only per le sole select: session.setReadOnly(object, true). In questo modo Hibernate non manterrà uno snapshot originale dell’entità selezionata nel contesto persistente per ulteriori dirty check
- Utilizzare cache di secondo livello e Query Cache per dati di tipo read-mostly e read-only
- Utilizzare il lock ottimistimico (campo version) ove necessario
- Usare FlushMode.COMMIT al posto di AUTO, in questo modo Hibernate non effettuerà select prima di update, ma porre attenzione in quanto questo potrebbe portare alla scrittura di dati più vecchi (anche se l’Optimistic Locking potrebbe essere d’aiuto)
- Prendere in considerazione il fetching batch (batch size) per selezionare diverse entità/collection in un’unica volta invece di effettuare query separate per ognuno di esse
- Utilizzare query come ‘'select new Entity(id, someField) from Entity’ al fine di recuperare solo i campi richiesti. Controllare anche i result transformers.
- Utilizzare operazioni batch (come la delete) se necessario
- Prendere in considerazione i materialized path e i nested set per strutture ad albero
- Utilizzare StatelessSession se possibile, in quanto evita i dirty check, il cascading, gli interceptor, etc…
- Limitare i risultati di ricerca e usare le paginazioni

2.7 View

- Mantenere le view più semplici possibili, evitare di inserire nelle view logica di business, codice java o collegamenti diretti al database e/o altri componenti (ejb ecc)
- Usare un template engine (sitemesh/tile)

- Rendere la view DRY ("Don't Repeat Yourself"), separando i contenuti eventualmente ripetuti in template o pagine da includere
- Utilizzare le tag lib JSTL,Expression Language e le altre tag lib messe a disposizione dai framework utilizzati o creane di proprie
- Organizzare le view in sottocartelle a seconda dell'utilizzo layout,pages,includes,template ecc. ed utilzzare nomi coincisi e sensati
- L'uso di librerie javascript dovrebbe essere centralizzato e organizzato anch'esso in ottica MVC (quanto necessario) attraverso l'uso di framework tipo backbonejs ecc.
- L'utilizzo di framework html5/css3/js è indicato per sfruttare soluzioni già pronte ed usare gli asset messi a disposizione per velocizzare e standardizzare la realizzazione del sito (esempio Twitter Bootstrap)

2.8 Maven

- Evitare codice duplicato spostando i tag comuni nel pom parent
- Usare la sezione di gestione delle dipendenze
- Minimizzare il numero di dipendenze SNAPSHOT
- Specificare sempre una versione di dipendenze in un pom parent
- Minimizzare il numero di Profili
- Produrre spesso release funzionanti
- Utilizzare il plugin delle dipendenze per recuperare risorse condivise
- Generare artifacts portabili (non inserire nel codice proprietà, path assoluti o dati di accesso a databse)

2.9 Caching/Serializzazione

- valutare l'utilizzo di cache Ehcache/Spring Cache

2.10 Linee guida sviluppo Spring

Una delle peculiarità di Spring, positivamente e negativamente, è che vi sono differenti modi per risolvere il problema; ecco che una delle più grandi sfide, utilizzando Spring, è scegliere il modo migliore per attuare le soluzioni.

2.10.1 Spring Namespaces

Non ci sono numeri di versione in riferimento allo schema.

Usare la seguente configurazione:

Questo perché spring aggiorna automaticamente la versione più recente disponibile presso le dipendenza del progetto (jars). Durante lo sviluppo del progetto, la versione di Spring verrà aggiornata, e lo sviluppatore non potrà mantenere tutti i file di configurazione XML per vedere le nuove funzionalità.

2.10.2 One Bootstrap XML File

Ci sono molti esempi di utilizzo di più file di configurazione XML. Un'applicazione di solito ha diversi file di configurazione XML, ma dovrebbe avere un solo file di bootstrap. Questo file di bootstrap dovrebbe usare il tag *<import resource=””/>* per includere altri file di configurazione.

2.10.3 Classpath Prefix

Usare sempre *classpath*: prefix

Durante l'importazione di risorse, configurazioni XML, proprietà, ecc, usare sempre *classpath*: oppure *classpath**: prefix.

Questo fornisce coerenza e chiarezza nella posizione della risorsa. Non tutte le funzionalità di Spring si comportano allo stesso modo, classpath: garantisce consistenza.

Il classpath è determinato dal tool di creazione e dall'IDE. Di solito è:

a) *src/main/java* per il codice java
b) *src/main/resources* per le dipendenze non-java e i testi
c) *src/test/resources* per le risorse non-java.

Esempio:

2.10.4 Bean Naming

Il contesto di Spring è un contenitore per i beans dell'applicazione. Ogni bean è identificato univocamente dal suo nome. L'attributo XML”id” è più comunemente usato per definire il nome del bean. L'attributo “id” è ottimo perché, per le leggi XML, è univoco per ogni file.

Tuttavia, se lo sviluppatore vuole usare simboli speciali nel nome o fornire alias nel nome, può usare come attributo "name".

Spring 3.1 aggiunge la funzione del profilo, che fornisce la possibilità di configurare i beans per categoria o regione.

2.10.5 Dependency Injection

Dependency Injection è uno dei principi base del framework Spring. Dependency Injection fornisce agli sviluppatori la capacità di "legare insieme" relazioni tra bean in configurazione invece di codificare le relazioni.
I nuovi metodi per effettuare Dependency Injection sono tramite Constructor Injection oppure Setter Injection.

2.10.6 Constructor Injection

Constructor Injection è eseguito utilizzando il <bean/> nodo <constructor-arg.
La sicurezza dei thread è una forte necessità per l'uso dei costruttori.

2.10.7 Setter Injection

Il setter injection offre la possibilità di iniziettare i beans attraverso un metodo setter. Tradizionalmente, questa è stata la scelta preferita per molti sviluppatori, perché la configurazione è più facile da leggere.

2.10.8 Third Party Beans

Qualsiasi classe Java può essere usata nel framework Spring.

2.10.9 Proprietà esterne

La configurazione di distribuzione richiede l'impostazione dei parametri dell'ambiente, come ad esempio la proprietà di connessione al database.
Dal momento che XML può essere fragile, è meglio esternare le impostazioni in file di proprietà. Questo rende più facile per il team di distribuzione di cambiare la configurazione delle risorse con meno rischi.
Spring fornisce un *PropertyPlaceholderConfigurer* a tale scopo.

2.10.10 Java Util Logging

Per abilitare la gestione di java.util.logging con classi SLF4J, registrare il seguente codice nella configurazione di Spring:

2.10.11 System.out e System.err

Per abilitare la gestione di System.out e i messaggi di System.err, occorre registrare il seguente codice nella configurazione di Spring:

3 Linee guida di sviluppo .Net

L'ufficio Sistemi Gestionali si occupa della realizzazione di servizi Intranet per la Camera di Commercio di Milano e socie. Nel corso degli anni le scelte di sviluppo si sono orientate verso l'ambiente web su piattaforme e con strumenti Microsoft.

3.1 Naming Convention

Per la naming convention utilizzata dai gruppi DigiCamere che sviluppano in .Net, valgono le linee guida utilizzate in Java, con le seguenti eccezioni:

- i nomi dei metodi vengono espressi con anche l'iniziale MAIUSCOLA, come da consuetudini di programmazione in ambienti Microsoft;
- in sostituzione a get e set, si utilizzano le proprietà native del framework .Net.

3.2 Database

Per il DBMS si è optato per l'utilizzo di Microsoft SQL Server; nelle release 2005 e 2008. Per l'amministrazione della base di dati è stato adottato il tool Microsoft SQL Server Management Studio o, in alternativa, Microsoft Access (tramite collegamento con driver ODBC).

Dove richiesto, sull'istanza del DBMS è presente anche il tool SQL Reporting Service, per la pubblicazione e l'esecuzione della reportistica.

3.3 ORM

Per l'accesso ai dati utilizziamo degli ORM, dopo alcune esperienze con il prodotto Opensource NHibernate siamo passati ad utilizzare il prodotto EDM nativo di Microsoft, che risulta totalmente integrato in Visual Studio, permettendo un approccio grafico alla modellazione del progetto.

Per l'interrogazione e la modifica dei dati, allo strato nativo del framework si aggiunge lo strato applicativo LINQ, che permette una gestione completamente ad oggetti del model sottostante. In questo modo viene disaccoppiato lo strato di connessione al database da quello riguardante il dbms in senso stretto, permettendo una sostituzione relativamente rapida dello strato dati, come ad esempio nel caso di un cambiamento di rdbms.

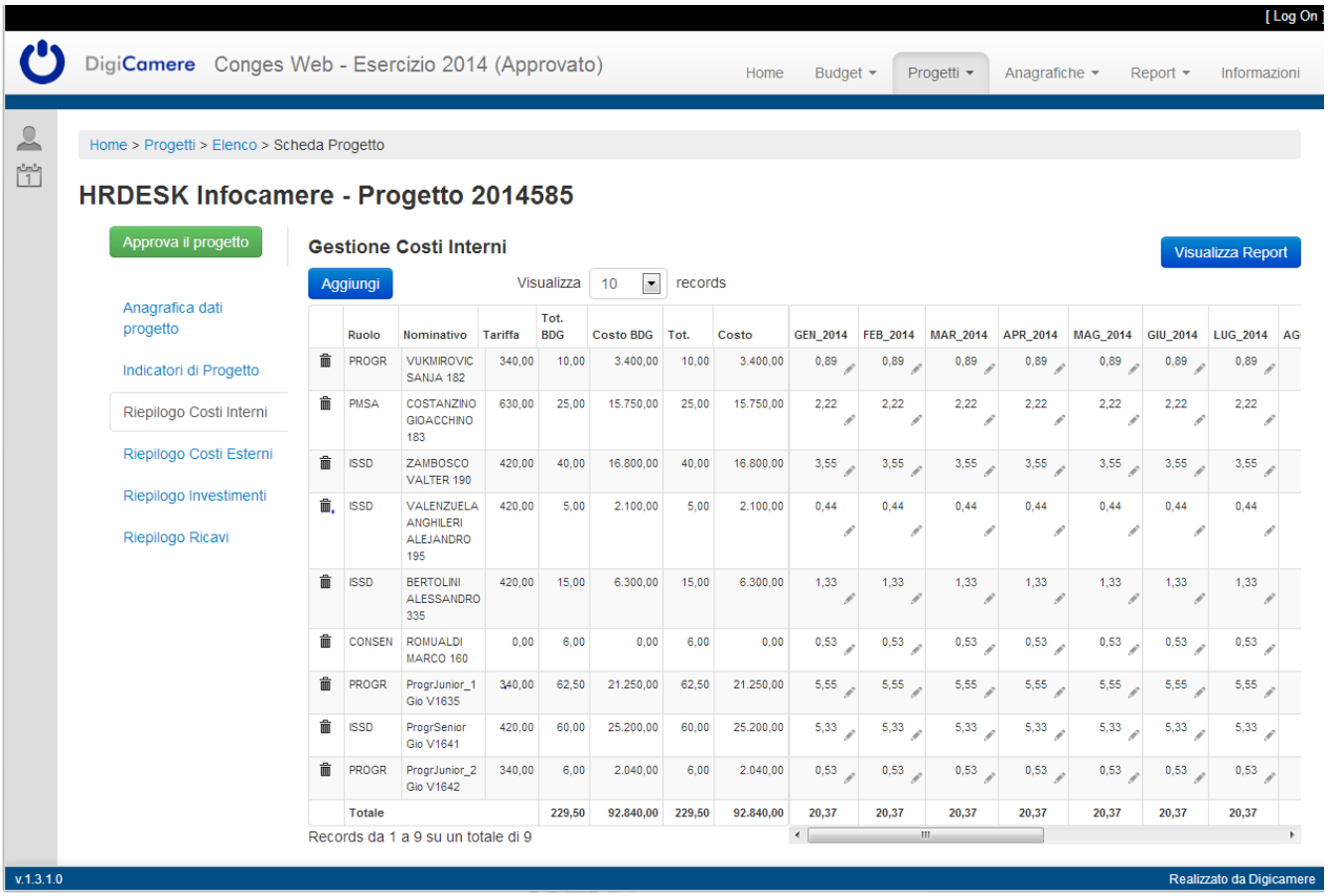
3.4 Framework (.Net Framework, MVC4 e version precedenti)

Lato server le nostre applicazioni sono sviluppate con il linguaggio Microsoft C#; negli ultimi anni è stato adottato il pattern MVC (versione attualmente utilizzata è la 4). Tale pattern permette uno sviluppo strutturato del codice, facilitando il riuso dello stesso, il lavoro in team e il test del software, e permettendo la realizzazione di applicazioni di maggiore qualità. In linea con il pattern abbiamo adottato delle convenzioni interne sulla nomenclatura degli elementi, come indicato nei precedenti paragrafi.

3.5 Front End

Per lo sviluppo del frontend, negli ultimi progetti si sta utilizzando il framework Twitter Bootstrap. Bootstrap offre una serie di tool per creare applicazioni web utilizzando template HTML basati su CSS per la tipografia, i form, i bottoni, il layout, navigazione e altre componenti di interfaccia. Con questo framework abbiamo creato un template comune a tutte le applicazioni sviluppate, con un forte impatto sulla produttività.

Sempre a livello di frontend, utilizziamo gli ultimi standard web: HTML 5, CSS3 e JQuery per la programmazione lato client.



3.6 Prototipazione

Durante la prima fase di definizione delle specifiche funzionali e tecniche di progetto, viene preparato un prototipo (mockup) parzialmente funzionale, che illustra l'interfaccia e la navigazione all'interno dell'applicativo. Tale prototipo viene proposto al committente per la verifica e la validazione; il prototipo approvato costituirà la specifica grafica e funzionale su cui si baserà lo sviluppo dell'applicativo vero e proprio.
Per la realizzazione dei prototipi, ad oggi viene utilizzato il software open source PENCIL; è inoltre in corso di sperimentazione la realizzazione di prototipi con Twitter Bootstrap; dal momento che tale libreria viene utilizzata nello sviluppo delle interfacce di prodotto, la realizzazione dei prototipi con questo sistema costituirebbe, a validazione da parte del cliente ultimata, già il primo nucleo della realizzazione della UI.

3.7 Ambienti di sviluppo

L'ambiente di sviluppo applicativo adottato è Microsoft Visual Studio; attualmente viene utilizzata la versione 2013.

Per il collegamento con il repository subversion, può essere utilizzato il plugin AnkhSvn, che permette la gestione integrata in Visual Studio di checkout e commit dei sorgenti. Parallelamente, resta comunque la possibilità di utilizzare i classici strumenti di commit e check out SVN (es. TortoiseSVN)

Per la gestione dei package e delle dipendenze, viene utilizzato l'add on NuGet, integrato all'interno dell'IDE, che si occupa del download, dell'integrazione e dell'aggiornamento dei package all'interno della soluzione, all'incirca come Maven lavora nel mondo Java.

Per la gestione e lo sviluppo della base dati, viene utilizzato il pacchetto Microsoft SQL Server Management Studio, con eventuale integrazione di un server di database locale, per sviluppo e test.

3.7.1 MIDDLEWARE

Il web server utilizzato è Microsoft Internet Information Services (IIS) 7 installato su macchine Windows, sulle quali è necessaria anche l'installazione e la configurazione delle estensioni MVC (nella versione utilizzata dall'applicativo), nonché eventuali plugin aggiuntivi, come ad esempio il Microsoft Sql Reporting Service Runtime qualora necessario per la visualizzazione della reportistica client, o ancora l'agent SSO per l'autenticazione ai servizi tramite il sistema di autenticazione OpenSSO adottato da DigiCamere.

3.7.2 Ambienti di deploy

Il deploy di database e applicativi può essere eseguito su tre ambienti:

- Sviluppo
- Test
- Produzione

3.7.3 SVILUPPO

È un ambiente riservato agli sviluppatori, contenente dati di prova; non viene garantita la stabilità del servizio in quanto soggetto a continue prove e rilasci. L'ambiente di sviluppo non ha nessun sistema di autenticazione sofisticato se non l'accesso all'applicazione tramite una password conosciuta dai soli sviluppatori

3.7.4 TEST

E' l'ambiente che viene messo a disposizione del cliente per effettuare le prime prove e i primi test sull'applicativo; non contiene dati reali se non quando da testare è proprio la correttezza del dato. L'ambiente è protetto con il sistema Open SSO. In alcuni casi vengono create delle

procedure per permettere ad una persona di simulare più utenti (addetto, responsabile, dirigente). L'ambiente ha una stabilità superiore allo sviluppo, tendenzialmente in momenti di testing importanti viene posta particolare cura alla comunicazione ed esecuzione dei rilasci.

3.7.5 PRODUZIONE

E' l'ambiente di esercizio dell'applicativo, contenente i dati reali, e protetto con i sistemi di autenticazione e profilazione completi (OpenSSO). Le utenze sono nominative, e non è possibile la simulazione di utenze nemmeno ai fini di assistenza. I rilasci in questo ambiente vengono fatti fuori dall'orario di lavoro standard; interventi eccezionali durante l'orario di lavoro che si dovessero rendere necessari vengono comunicati tramite mail.

3.7.6 Compatibilità delle soluzioni realizzate con il contesto tecnologico disponibile

Dovendo realizzare applicativi che risultino perfettamente fruibili sulle diverse piattaforme web browser disponibili, è stata fatta una selezione dei browser per i quali garantire la compatibilità; in particolare, prendendo in considerazione le seguenti precondizioni:

- I browser Internet Explorer 6, 7 e 8 sono sconsigliati dallo stesso produttore, con particolare riferimento alla sicurezza degli stessi;
- La versione 10 di Internet Explorer è la più recente release del browser Microsoft installabile su Windows XP;

Si è stabilito che gli applicativi realizzati debbano essere compatibili con i seguenti browser:

- Internet Explorer, versione 10 e successive, con o senza attivazione della modalità compatibile;
- Mozilla Firefox, tutte le versioni correnti;
- Google Chrome, tutte le versioni correnti.

4 Linee guida sviluppo Php

4.1 Premessa

Nel corso degli anni, a partire dalla fine degli anni 90, sono state realizzate diverse decine di siti informativi e di applicazioni web facendo uso del linguaggio php. Le linee guida qui di seguito riportate fanno riferimento alle modalità che si intendono adottare per i nuovi siti o in fase di revisione dei siti esistenti.
La grande maggioranza dei siti realizzati si basa su un CMF (Content Management Framework) realizzato internamente e che è cresciuto negli anni sulla base delle esigenze di volta in volta emerse. Nonostante il core del CMF sia stabile da diversi anni, ad oggi non ne esiste una versione unica ma molte diverse con singole peculiarità.

4.2 AMBIENTE

Lo stato attuale dell'arte vede l'utilizzo dei seguenti strumenti:

Frontend: HTML, CSS, Javascript, JQuery
Developer Tool: PhpDeveloper, NetBeans, IntelliJ, Eclipse; PhPStorm
Linguaggio lato server: Php 5
Database: MySQL 5
Middleware: Apache 2
Knowledge Management Platform: Planio
Version Control: Subversion
Principali Librerie: Google API, phpLib, libChart, phpexcel, phpMailer
Prototipazione: Pencil

4.3 Situazione attuale

I siti costruiti sul CMF sono realizzati tenendo conto del pattern MVC separando la logica di presentazione dei dati dalla logica di business. Le componenti HTML e CSS statiche sono memorizzate in apposite cartelle (tmpl e css) mentre quelle dinamiche sono memorizzate nel database. I template sono modificati di volta in base alla grafica originale scelta dal cliente e sono sviluppati partendo dalla soluzione esistente più simile.

Per la realizzazione e dinamicizzazione del frontend vengono utilizzati gli standard web HTML, CSS e Javascript, in quest'ultimo caso facendo uso di librerie pubbliche di larga diffuzione come jQuery e prototype. Per i nuovi sviluppi si tende a seguire la tecnica RWD (Responsive Web Design) in modo che le pagine adattino automaticamente il layout per fornire una visualizzazione ottimale in funzione dell'ambiente nei quali vengono visualizzati.

4.4 FRAMEWORK

Per lo sviluppo di nuove applicazioni si adottano dei framework che agevolano e velocizzano lo sviluppo di applicazioni web.
Lato Backend vengono usati due framework basati su symfony php: Drupal (7 e 8) e Laravel 5. Drupal viene utilizzato per la realizzazione di siti informativi e intranet, mentre Laravel viene utilizzato per lo sviluppo di applicazioni custom.
Lato Frontend vengono utilizzato Twitter Bootstrap e Angular JS.

4.5 STRUMENTI DI SVILUPPO

Gli sviluppatori php, per velocizzare gli sviluppi, fanno uso di editor che mettono a disposizione le tipiche funzionalità di sviluppo rapido quali auto completamento, evidenziazione e verifica della sintassi, editing file via ftp, etc.

Gli editor attualmente utilizzati in azienda sono PhpDeveloper, PhpStorm, NetBeans ed Eclipse

Per la gestione e lo sviluppo dei database sono utilizzati diversi tool quali MySQL Workbench, Toad, heidiSQL, phpMyAdmin e MySql Front.

4.6 MIDDLEWARE

Il web server utilizzato per i progetti realizzari in php è Apache 2 in ambiente linux (RedHat o Debian).

Per il corretto funzionamento dei siti realizzati con CMF è necessario impostare alcuni parametri in php.ini, in particolare register_globals ad ON e safe_mode ad OFF. Per garantire un buon livello di sicurezza anche con l'impostazione di questi parametri è stata scritta una libreria php denominata "cleaninput" che analizza tutti i parametri in ingresso, li valida e bonifica eventuali tentativi di sql injection. Sono in corso aggiornamenti al CMF per evitare queste impostazioni che nelle prossime release di php saranno deprecate se non addirittura disattivate.

4.7 KNOWLEDGE MANAGEMENT PLATFORM E VERSION CONTROL

Per la gestione del progetto si sta convergendo all'utilizzo della web application Plan.io con url: digicamere.plan.io.

In particolare, su Plan.io, sono registrate le seguenti attività:

- Segnalazione di anomalie
- Condivisione di informazioni
- Pianificazione e suddivisione di attività
- File Sharing
- SVN Repository
- Git Repository

La suddivisione del repository rispetta le convenzioni standard:

- In Trunk risiede l'attuale versione del progetto
- In Branches mettiamo eventuali personalizzazioni, o sviluppi "pericolosi"
- In Tag pubblichiamo le main release

4.8 PRINCIPALI LIBRERIE

Gli principali librerie utilizzate nella maggior parte dei progetti sono:

- phpLib: librerie open source per la gestione di sessioni, autorizzazioni, connessioni a db, etc.
- libChart: libreria per la generazione di grafici dinamici;
- phpexcel: libreria per la generazione di documenti excel;
- Google API: librerie per l'interazione con Google (Maps, Graph, …);
- phpMailer: libreria per l'invio di mail;
- dompdf e html2pdf: libreria per la generazione di file pdf;

4.9 PROTOTIPAZIONE

Durante la prima fase di definizione delle specifiche funzionali e techiche di progetto, viene preparato un prototipo (mockup) parzialmente funzionale, che illustra l'interfaccia e la navigazione all'interno dell'applicativo. Tale prototipo viene proposto al committente per la verifica e la validazione; il prototipo approvato costituirà la specifica grafica e funzionale su cui si baserà lo sviluppo dell'applicativo vero e proprio. Per la realizzazione dei prototipi, ad oggi viene utilizzato il software open source PENCIL.

4.10 DEFINIZIONE STANDARD

E' importante che vengano seguiti degli standard nella nomenclatura di classi, costanti e variabili globali. La selezione del nome dovrebbe preferire nomi brevi e significativi che facilitino la lettura del codice.

In generale:

- selezionare nomi mnemonici e brevi per attributi/variabili;
- selezionare nomi significativi per i metodi in modo da riuscire a identificare chiaramente il servizio che forniscono, i metodi rappresentano azioni per tanto i loro nomi dovrebbero includere un verbo. Qualora un metodo svola più azioni aggiungere le "and" o "or" nel nome ,saveOrUpdate,findByNameAndCode ecc.

4.10.1 Classi

I nomi devono essere camelcase.

Esempi:

DirittiPratica
UtenteRuolo

4.10.2 Costanti

Nomi in uppercase.

Esempi:

DEFAULT_WIDTH
MAX_HEIGHT;

4.10.3 Variabili

Nomi in lowercase per variabili locali e con iniziale maiuscole per variabili globali. Utilizzare il carattere underscore per nomi composti.
Esempi:

contatore
Cognome, Nome. Ragione_Sociale

4.10.4 Metodi e funzioni

Nell'implementazione di metodi e funzioni:

- evitare l'implementazione di metodi troppo lunghi (nel caso si utilizzano framework suddividere il codice in base alle responsabilità dei componenti)
- non scrivere metodi con molti parametri, se la firma del metodo contiene più di 4/5 parametri è necessario valutare se è il caso di incapsualare questi parametri in un'apposita classe (DTO/Model)
- ritornare oggetti strutturati ove necessario
- utilizzare le parentesi graffe anche quando non è strettamente necessario

4.10.5 Gestioni eccezioni ed errori

Relativamente alla gestione di eccezioni ed errori si sta convergendo sull'utilizzo di una classe php che intercetta gli errori, li logga ed invia un alert agli sviluppatori del sito.

E' importante evitare l'esposizione di messaggi di errore all'utente e, preferibilmente, di sostituire i messaggi standard dei web server ("404 file not found", etc) con messaggi personalizzati.

4.10.6 Database

La definizione del database e delle relative tabelle/campi dovrebbe essere fatta solo dopo aver definito (aiutati da una buona analisi) le entità coinvolte e le relazioni tra esse. Dalla definizione degli oggetti (tabelle) e delle propritetà (campi) si può poi partire con la creazione (manuale o con l'uso di tool automatici) del database.

Effettuare l'operazione inversa ovvero partire da database datati, senza fk, senza naming convention con relazioni scarsamente definite è della causa principali di problemi che verranno fuori durante la fase di sviluppo di un applicazione sia su piattaforma Java, PHP, Ruby, Grails, ecc.

È quindi molto importante rispettare alcune regole base:

- definire e rispettare una naming convention
  - lowercase per tutti i nomi di tabelle,view,campi,fk,indici ecc.
  - nomi composti separati da "_" per esempio : residenza_indirizzo,residenza_cap ecc.
- corretto utilizzo delle foreign keys

- corretto utilizzo degli indici
- autoincrement

Le tabelle del CMF php hanno adottato generalmente lo standard di avere come prefisso:

- "T_" per le tabelle
- "Tlk_" per le tabelle di lookup
- "Tj_" per le tabelle di join

A questo si antepone un prefisso per identificare le tabelle utilizzate da una singola applicazione o modulo applicativo, esempio:

- "CONVEGNI_" per tutte le tabelle dell'applicazione di gestione Convegni Online;
- "BANDI_" per tutte le tabelle dell'applicazione Bandi Online

4.11 AMBIENTI DI RILASCIO

Lo sviluppo delle applicazioni php sta evolvendo verso l'utilizzo di 3 ambienti sia per i database che per applicazioni:

- Sviluppo
- Test
- Produzione

Ad oggi, nella maggior parte dei casi, non è disponibile un ambiente di test.

4.11.1 Sviluppo

È un ambiente riservato agli sviluppatori, contenente dati di prova; non viene garantita la stabilità del servizio in quanto soggetto a continue prove e rilasci. L'ambiente di sviluppo non ha nessun sistema di autenticazione sofisticato se non l'accesso all'applicazione tramite una password conosciuta dai soli sviluppatori

4.11.2 Test

E' l'ambiente che viene messo a disposizione del cliente per effettuare le prime prove e i primi test sull'applicativo; non contiene dati reali se non quando da testare è proprio la correttezza del dato. L'ambiente ha una stabilità superiore allo sviluppo, tendenzialmente in momenti di testing importanti viene posta particolare cura alla comunicazione ed esecuzione dei rilasci.

4.11.3 Produzione

E' l'ambiente di esercizio dell'applicativo, contenente i dati reali). Le utenze sono nominative, e non è possibile la simulazione di utenze nemmeno ai fini di assistenza. I rilasci in questo ambiente vengono fatti fuori dall'orario di lavoro standard; interventi eccezionali durante l'orario di lavoro che si dovessero rendere necessari vengono comunicati tramite mail.

4.12 REQUISITI DI COMPATIBILITA'

Attualmente le applicazioni web di nuova realizzazione vengono garantiti per la compatibilità con tutti i browser di maggiore diffusione:

- Internet Explorer 8 e versioni successive
- Mozilla Firefox
- Google Chrome

5 Strumenti condivisi

5.1 Come si usa SVN

Subversion (noto anche come svn, che è il nome del suo client a riga di comando) è un sistema di controllo versione.
Subversion offre le seguenti caratteristiche:

- Comprende gran parte delle caratteristiche di CVS.
- Le directory, i cambi di nome, e i metadati dei file sono sotto controllo versione.
- Le commit sono vere transazioni atomiche. Una commit interrotta non lascia il repository in uno stato di incoerenza.
- Come server centralizzato si può usare il server Web Apache, tramite il protocollo WebDAV/DeltaV, oppure un server indipendente che usa un protocollo personalizzato basato su TCP/IP.
- Il branching e il tagging sono operazioni veloci, che richiedono un tempo indipendente dalla dimensione dei dati.
- Il progetto è nativamente client/server, ed è basato su una libreria stratificata.
- Il protocollo client/server invia solo le differenze in entrambe le direzioni, e quindi i costi di comunicazione sono proporzionali alla dimensione delle modifiche, non alla dimensione dei dati.
- I file binari sono gestiti efficientemente.
- L'output dei comandi è analizzabile da un programma esterno, e viene fornito un log opzionale in XML.
- La licenza è Open Source, simile a quella di Apache.
- I messaggi dei programmi sono internazionalizzati.
- I link simbolici sono sotto controllo versione.
- Viene supportato un nuovo formato opzionale del repository, FSFS, che non fa uso di un gestore di database, ma memorizza le revisioni direttamente nel file system.
- Lock dei file per i file inconciliabili
- Completo autoversionamento WebDAV

5.2 Plan.io

Planio è una base perfetta per aiutarti a pianificare e gestire i tuoi progetti online. Le maggiori funzionalità fornite sono: un sistema di ticket con tracciamento orario, gestione collaborativa di file, wiki e forum, un sistema di news, milestones e generazione automatica di diagrammi di Gantt e Roadmap.

5.3 Sincronizzazione di file

Lavorare collaborativamente su dei documenti può essere difficile. Con la nostra pagina centrale di documenti e il repository integrato per il controllo della versione, la condivisione di file è stata semplificata significativamente. In taluni casi la condivisione e la collaborazione può essere impostata utilizzando sistemi più "leggeri" come ad esempio google docs; pur rimanendo vincolante l'utilizzo di plan.io, in caso di necessità la presente procedura potrà essere rivista individuando le eccezioni in cui sia plausibile l'utilizzo di sistemi alternativi, basandole sui "Cluster applicativi"

5.4 Assegnazione di compiti

Il sistema sofisticato di assegnazione di ruoli basato sul controllo d'accesso garantisce agli utenti le giuste autorizzazioni. I compiti possono essere assegnati a partecipanti individuali, il che, combinato con le milestone, rende il tracciamento dello status del progetto un gioco da ragazzi.

5.5 Gestione della conoscenza

Grazie allo strumento Wiki e alla ricerca a tutto testo diventa facile raccogliere e presentare le informazioni in modo strutturato, assicurando che esperienza e competenza siano facilmente accessibili a tutti i membri del team e incoraggiando la protezione della proprietà intellettuale e organizzativa.

6 Sicurezza software

La sicurezza software non è affidata unicamente allo sviluppatore del software stesso, ma è un processo che comprende l'intero ciclo di vita del software e tutte le persone che fanno parte del team di sviluppo.
Certamente la maggior parte delle vulnerabilità le troviamo nella fase di sviluppo e seguendo le linee guida del linguaggio e/o framework utilizzato potremo andare a ridurre la possibilità di lasciare delle vulnerabilità, almeno le più semplici e facili da sfruttare.

Di seguito sono riportare le linee guida per ogni fase del ciclo di vita del software.

6.1 Analisi dei requisiti

In questa fare l'analista dovrà porre attenzione a quale possono essere i dati, le funzionalità alle quali lo sviluppatore dovrà sviluppare con maggiore attenzione, sia per motivi di complessità funzionale, sia per il fatto che si trattino di dati sensibili/riservati che se non adeguatamente gestite possono portare a delle brecce di sicurezza.

6.2 Architettura:

- **Modularità**: dividere il software in parti semi-indipendenti tra di loro
- **Isolamento**: ogni parte del software dovrebbe funzionare anche se le altre parti restituiscono errori o risultati errati
- **Semplicità**: soluzioni complesse sono molto probabilmente più insicure di soluzioni semplici
- **Ridondanza**: se è possibile adottare soluzioni ridondanti per evitare di avere un solo punto di fallimento (a single point of failure)
- **Limitare le risorse consumabili**: limitare le risorse riduce l'impatto di un Denial of Service attack (DOS). Per esempio su un server ci sono due applicazioni che possono utilizzare tutte le risorse della macchina, in caso di un attacco DOS il server collasserà rendendo inaccessibili entrambe le applicazioni, mentre limitando le risorse accessibili il server rimarrà attivo e l'applicazione che non è soggetta all'attacco sarà ancora disponibile.

6.3 Design:

- **Riservatezza (o confidenzialità):** le informazioni gestite dall'applicativo devono essere accessibili direttamente o indirettamente solo agli utenti che ne hanno diritto e che sono espressamente autorizzati a conoscerle
- **Integrità**: le informazioni gestite dall'applicativo devono essere protette da alterazioni, quali modifiche, danneggiamenti o cancellazioni improprie, ad opera di utenti non autorizzati, o anche a causa di eventi accidentali
- **Disponibilità**: le informazioni gestite devono essere sempre accessibili agli utenti che ne hanno diritto, nei tempi e nei modi previsti

- **Principio del Least privilege**: non richiedere più privilegi di quanti ne abbia realmente bisogno. Assicurarsi che l'utenza con la quale verrà eseguito il codice abbia l'accesso ai suoi soli file ed abbia solo i privilegi necessari.
  Allo stesso modo le utenze applicative devono poter accedere alle funzionalità e alle risorse a lei assegnata
- **Principio del Secure by default**: per esempio una password casuale che l'utente deve cambiare piuttosto che un insieme di passwords standard predefinite che molti non si preoccupano di cambiare
- **Principio del Deny by default**: per esempio, durante la convalida dell'input dell'utente accettare solo caratteri che ci si aspetta in ingresso, piuttosto che cercare di bloccare i caratteri non permessi
  Negare l'accesso alle risorse abilitando solo chi possiede le autorizzazioni
- **Principio del Defense in depth**: costruire più livelli di difesa invece di confidare in un solo meccanismo di protezione. Per esempio: convalidare i dati di input utente al punto di entrata, e controllare di nuovo tutti i valori che vengono passati a parti sensibili del codice (come la gestione del file, ecc)

6.4 Sviluppo

6.4.1 Considerazioni generali

- Leggere e seguire le linee guida del linguaggio/framework che si sta utilizzando
- Leggere e seguire le linee guida della OWASP TOP 10 più recente, e la precedente per sapere quali sono le principali vulnerabilità che possono presentarsi nello sviluppo di un software
- Pensare alle conseguenze sulla sicurezza del proprio codice
- Riusare codice sicuro e testato (librerie, moduli, plugins, …)
- Scrivere codice di buona qualità, leggibile e mantenibile (un codice di bassa qualità spesso non è sicuro)

6.4.2 Codice

- **Non fidarsi degli input data**: essi possono pervenire da utenti con intenzioni malevoli che mirano a fare breccia nel sistema (buffer overflow, SQL injection, Cross Site Scripting (XSS), code injection, …)
  Sono considerati input data anche i command-line arguments, file di configurazione (se accessibile a utenti non "sicuri"), variabili di ambiente, cookies e POST/GET arguments.
  Eseguire controlli di sicurezza (validazione ed autenticazione) lato server. Bisogna ricordarsi che le HTTP response header fields (cookies, user-agent, referrer etc.) e le HTTP query string values (da campi hidden o links espliciti) possono essere modificate nel percorso dal client al server
- **Validare tutti gli input data**: considerare tutti gli input data pericolosi fintanto che non vengono sanitizzati e validati, negare by default se non si è sicuri. Effettuare validazioni a più livelli, per esempio validare il dato in ingresso e prima di riutilizzarlo per essere sicuri del dato stesso.
- **Non fare alcuna ipotesi circa l'ambiente di esecuzione**: essere sicuri che il proprio codice non sia soggetto a modifiche malevole dei PATH, CLASSPATH, o di altre variabili di ambiente
- **Attenzioni alle race condition**: il tuo codice viene eseguito in parallelo? Cosa succede se qualcuno esegue due istanze del tuo programma allo stesso tempo?
- **Gestione degli errori e delle eccezioni**: non date per scontato che tutto funzionerà (in particolare operazioni sui file, di sistema e di chiamate di rete, …), occorre gestire le eccezioni, catturandole ed andando ad analizzare i log delle stesse, senza visualizzare a video gli errori (internal error messages, failing SQL query, stack trace, …) ma un messaggio personalizzato di anomalia
- **Fallimento controllato**: se si verifica un errore inatteso e non si riesce a recuperare informazioni sello stesso nei log (applicativi e/o di sistema), avvisare

l'amministratore che procederà ripulire il sistema (cancellare file temporanei, pulire la cache, …) e avvisare l'utente
- **Proteggere passwords e informazioni riservate**: utilizzare algoritmi di cifratura/crittografazione di passwords o di informazioni riservate
- **Fare attenzione alla manipolazione dei file**: se si desidera crearlo, segnalare un errore se esiste già.
  Al momento della creazione, impostare i permessi dei file, se si apre un file per leggere i dati, non chiedere l'accesso in scrittura.
  Controllare se il file che si deve aprire non è un collegamento, con le funzioni messe a disposizione dal linguaggio in uso (prima e dopo l'apertura del file). Utilizzare percorsi assoluti, porre attenzione maggiore quando il nome del file (o parte di essa) proviene da un input data di un utente
- **File temporanei**: qualcuno indovina il nome del file temporaneo, e crea un collegamento da un altro file ad esempio /bin /bash, che il programma sovrascrive. I file temporanei devono avere nomi univoci che sono difficili da indovinare
- **Crittografia**: usare algoritmi, protocolli e prodotti pubblici e sicuri. Non inventare propri algoritmi e protocolli di crittografia né re implementare quelli esistenti

6.5 Test

- **Revisione del codice**: rivedere il codice scritto, interagendo con un altro sviluppatore
- **Quando trovo un bug di sicurezza**: ricercare lo stesso problema all'interno del software per verificare se è presente in altre parti del codice
- **Test mirati**: effettuare dei test con casi limite per verificare che il software non sia vulnerabile in questi casi
- **Usare tools di exploit**: con questi tool possiamo scansionare la nostra applicazione per verificare la presenza di vulnerabilità, da poter così porre rimedio o pianificare un piano di rientro nel caso di vulnerabilità difficili a cui rimediare**.**
  Il tool attualmente utilizzato è VEGA VULNERABILIRIES SCANNER
  (https://subgraph.com/vega/index.en.html)

6.6 Deploy

Prima di procedere con il deploy assicurarsi di aver disabilitato tutti i possibili errori e warning che possono venire visualizzati a video ed avere superato la fase di test.
Nei framework basterà disabilitare il debug e settare l'ambiente di produzione nei file di configurazione degli stessi.