** Ministerul Educaţiei Republicii Moldovei**

**Universitatea Tehnică a Moldovei**

[**Departamentul**](https://utm.md/subdiviziuni-universitare/facultati/facultatea-calculatoare-informatica-si-microelectronica/catedra-calculatoare/) **informatica si ingineria sistemelor**

**Raport**

**La proiectul de practica**

A elaborat: Nicolenco Eugeniu, Gr. C-162

Mentor: Alexei Corduneanu, SRL „Vivat Consulting”

2018

Introducere

Marele tendințe ce apar asăzi în lumea IT creează oportunități pentru studenți și cerințe pe piață așa cum ar fi elaborarea aplicațiilor web responsive și flexibile.

Elaborarea web API presupune parcurgerea a mai multor pași. Exista căi ușoare, scurte, există căi mai lungi dar sigure. Prin cale sigură se înțelege elaborarea aplicației flexibile, ușor de extins și de manipulat cu conținutul ei. Calea ușoară presupune elaborarea aplicației direct în fișierele de bază, de execuție, cu un număr fix de funcții și conținut, astfel de aplicații sunt greu de extins sau de corectat. Pașii de elaborare a aplicației flexibile sunt deduși în mersul lucrarii și exemplificați prin obiect sau explicație de conținut rezonabil.

Ca obiect al lucrării practice am ales de a elabora o aplicație web ce ar prezenta fața unei companii ce prestează servicii foto – video, care ar cuprinde posibilități de contact, promovare, informare și comandă a serviciilor presupuse. Aplicația va fi stilată conform tendințelor și cerințelor contemporane.

Lucrarea este împărțită pe capitole și teme așa cum am lucrat și am învățat în timp. Fiecare capitol este destinat unei părți al dezvoltării aplicației precum și al dezvoltării duhui de echipă ceea ce este nu mai puțin de important pentru a dezvolta o aplicație în termen și cu o funcționalitate modernă; schimbul de idei între colegi, co-dezvoltatori este opțiunea care ne ajută să găsim soluții mai bune chiar atunci cînd ne pare că sunt ascunse de noi și altă cale nu există.

Pentru lucru în echipă este necesar de a stabili niște standarte de comunicare, de programare, care convin tuturor și nu duce colectivul în eroare; pentru astfel de sarcini lumea IT deja a propus o mulțime de soluții și standarte pentru dezvoltatori, care sunt valabile pentru orice limbaj sau mediu de programare; Colecția de instrumente și reguli vor fi descrise în conținutul lucrării și descrise adițional.

**Capitolul I**

* 1. Comunicarea corporativă

Comunicarea este partea fundamentală a relațiilor inter-personale ce compune orice societate, comunitate, grup. Comunicarea realizează legături între diferite părți ale sistemelor precum și diferite sisteme.

Comunicarea într-o corporație presupune conducerea/subordonarea, managementul, etc. Astfel există și reguli de comunicare într-o întreprindere și relații între colegi, relații cu parteneri. Comunicarea înseamnă aproape totul în management, atîta timp cît de calitatea comunicării depinde înțelegerea problemelor cu care se confruntă orice angajat.

Comunicarea verbală cît și convobirea prin telefon ține de cîteva da și nu ce trebuie respectate:

Da –

* Convorbirea trebuie să fie fluentă, clară și destul de sonor într-un limbaj cunoscut de toți participanții.
* La comunicarea în grup trebuie de adresat cui ne adresăm, chiar dacă vorbim întregului grup, e bine să spunem ”Dragi prieteni” sau ”stimate public”.
* Cînd începem convorbirea cu cineva stăin e bine să ne prezentăm primul.

Nu –

* Nu mestecăm în timul convorbirii.
* Nu oferim prea multă informație în perioade scurte de timp.
* Nu folosim abreviaturi în convorbire.

și multe altele.

1.1.2 Comunicarea între corporative/filiale

Cea mai extinsă metodă de comunicare între parteneri/colegi este expedierea e-mail-urilor sau cum se mai numește poșta electronica, care se foloseste pentru convorbiri unu la unu, unu la mai mulți sau conferență.

Comunicarea prin mail are foarte bune oportunități comparativ comunicării verbale sau telefonice, ce ține de securitate, necesitate, conținut, acoperire și altele.

Securitatea convorbirii este ceea pentru ce opteză toți utilizatorii web, și este acel factor de ce folosesc poșta electronică. Securiatatea mail-urilor poate fi aranjată de criptare, expedierea prin canaluri private, VPN etc, tot aici mai putem spune că mail-urile pot fi securizate prin nivele de acces sau parolă.

Necesitatea comunicării prin mail apare atunci cînd avem de expediat fișiere, mesaje cu conținut masiv. Este o bună alegere pentru a expedia sarcini subordonaților sau formulare de completat la grupuri de utilizatori. La fel putem spune că necesitatea putem a o numi ”urgență”, deci putem lăsa un mesaj pentru a fi citit și răspuns mai tîrziu.

Conținutul mesajului poate fi diferit, aceasta este o opțiune foarte bine utilizată de administratori ale diferitor servicii care promovează mesaje cu conținut similar paginilor web, tabele excel, dări de seamă și multe alte feluri de vizualizări.

Acoperirea este elementul nedesponibil chiar și multor messengerelor contemporane, mesajele electronice pot fi accesate din orice loc al lumii unde există internet.

Cele mai importante dialoguri fie între diferite corporații fie între colegi, deoarece mesajele se salvează în diferite servicii, fie depozitele deținătorilor serviciilor mail (*Gmail*) sau depozite locale, fiecare utilizator deține istoria proprie a mesajelor și istoria înmărțită de alți utilizatori, și pot fi accesate în orice timp de oriunde, și pot fi folosite pentru amintire a evenimentelor din trecut sau ca indicii pentru pașii următori.

* 1. Programarea în echipă

Programarea în echipă este modalitatea de funcționare a tutror întreprinderilor în domeniul IT, deoarece lucrînd la un proiect mare este imposibil totul să fie făcut de un singur om. În prezent echipele se structurează pe departamente și subdepartamente în care fiecare are funcțiunea lui; pentru funcționarea corectă și efectivă a unei echipe este necesar ca fiecare participant să introduce un impact în lucru, o idee, o soluție etc.

Pentru un lucru efectiv și economie de timp la programatori se stabilesc cîteva standarte fie pnetru un timp îndelungat în companie, fie doar pe proectul curent. Echipa stabilește șablonul de programare, după ce se propun soluții de dezvoltare și menținere a proiectului din care se aleg cele mai convenabile, cele mai performante. Se stabilește forma creerii variabilelor, funcțiilor, componentelor și structurilor de date, această opțiune este numită *naming conventions* – un set întreg de convenții care se stabilesc fie de echipă/șef de echipă sau de client/reprezentant al comenzii. Pot fi multe tipuri de convenții, însă există seturi cunoscute global care vor fi descrise mai jos:

* camelCaseConventions / \_snake-conventions / slash-conventions – tipuri de variabile (mereu prima literă este mică ”a” așa tip se numește lowerCase – specific varibilelor)
* CamelCaseConvention / … - tipuri de clase în limbaje orientate pe obiecte (prima literă este mare ”A” – specific claselor)

Deasemenea pot fi folosite și diferite prefixe pentru amintirea tipului variabilei, clasei sau metodei. Spre exemplu limbajul PHP strict utilizează simbolul $ pentru orice tip declarabil de programist.

* CAPS\_CASE – se folosește cam rar dar este preferată de unele limbaje de programare cum ar fi Rust, Python, Ruby pentru a specifica date constante.

1.2.2 Șabloane de programare

În programare există multe tipuri de șabloane, primele au fost introduse chiar la începutul erei de programare, acestea au fost – programarea declarativă, logică și programarea orientată pe obiecte.

Acum piața programării este acoperită în majoritar de programarea orientată pe obiecte, însă acum programarea orientată pe obiecte a devenit un domeniu sau o categorie ce cuprinde varietăți de șabloane de priectare și design pentru a face programarea mai efectivă și mai puțin costisitoare în timp.

Șablon de design – așa mai numit *design pattern* este abordare a soluțiilor programării care cu timpul a devenit ceva necesar la crearea unei API de orice complexitate. Alegerea pattern-ului de proiectarea a sistemei îi poate veni fie liderului echipei fie reprezentatului comenzii.

Design pattern-urile acum se utilizează ca elemente de bază, asemenea unor metode incluse în limbaje bine dezvoltate, în framework-uri dezvoltate pe diferite domenii, fiew web sau desktop, în care se include:

* creator de obiecte (prin ierarhii de obiecte) care au incluse metode de bază și pot defini metode noi.
* obiect fără pereche (singleton) – obiecte unice într-o aplicație (cum ar fi ”Cursor”)
* compoziție de obiecte
* decorator și altele

De bază design pattern-urile se împart pe 3 mari categorii: structurale, de creare și de corecție(sau de comportament) din care cele mai semnificative și des utilizate (pe lîngă cele exemplificate mai sus) putem marca următoarele: Iterator, observer, statut, adapter, proxy,punte, prototype, abstract factory (crearea obiectelor abstracte).

Șabloanele de arhitectură însă își extind infulența asupra dezvoltării aplicației decît pe modul de abordare a soluțiilor, și conține idea de structurare a codului, fișierelor și bibliotecilor de date separat.

MVC – (Model-view-controller) – este un pattern des aplicat în construcția aplicațiilor responsive în care se separă model-ul (date, informații din baze sau fișiere) de view (suprafața vizualizată de utilizator [User interface]) și de controller (unitatea ce controlează acțiunile utilizatorului și produce răspuns) una de alta în fișiere aparte în așa fel că modificarea fiecărui component se realizează independent de celelalte. Scopul de bază a acestui concept de proiectare constă în separarea *business*-logicii aplicației de vizualizarea utilizarea acestui șablon permite reutilizarea codului, extinderea lui, la fel cum și întreaga aplicație. Astfe Cel mai semnificativ plus a acestui gen de programare este că la dezvoltarea fiecărui component aparte poate lucra un om sau o echipă, după structura și convențiile stabilite. Astfel de arhitectură face dezvoltarea, ba chiar și aplicația destul de flexibilă și reduce numărul de pietre ce se pot întîlni pe calea realizării aplicației. Deoarece MVC nu are strictețe în utilizare, poate fi utilizat în mod diferit, nu exită un decensămînt stabilit unde trebuie să fie aplicată *business-*logica, ea poate fi aplicată și în controller și în model. MVC utilizează destul de multe șabloane de design, din care cele necesare sunt: Compozitorul (crează o ierarhie din obiecte în formă arborescentă), Observatorul (pentru utilizarea acțiunilor utilizatorului) și Strategie(propune încapsularea datelor, ce și crează separarea componentelor una de alta).

MVC – este alegerea bună pentru începători pentru că conține ajutor ascuns în comportamentul său prin delegarea fiecărui component anumite părți a funcționării; Astăzi există multe instrumente bazate pe MVC care deasemenea sînt ușor de implementat și utilizat, o oportunitate indispensabilă pentru începători în porgramare și amatori. *Schema MVC în anexa 1*

**MVVM** – (Model-View-ViewModel) Asemănător MVC, a fost proiectat ca moștenire de la MVP (Model-View-Presenter) este orientat pe platforme de dezvoltare așa cum Windows Presentation Foundation, Silverlight și ZK framework. La fel se utilizează pentru separarea componentelor una de alta, spre deosebire de alte pattern-uri în MVVM apare ViewModel care cel mai des (dar nu obligatoriu) conține *business-*logica unde se conține modul de funcționarea a aplicației și comportamentul în conformitate cu acțiunile utilizatorului. ViewModeul permite aplicarea conceptului de legare a datelor în două direcții, spre exemplu completarea formei de logare îndată produce răspunsuri pentru corectitudinea completării și reacționează la fiecare schimbare aplicată de către utilizator.

La folosirea acestui pattern view (componenta de vizualizare) nu produce o interfață visibilă direct dar transferă această obligație ViewModel-ului care prin legarea fluxului de date în ambele sensuri produce această interfață pentru a fi mai departă afișată conform parametrilor setați în view.

*Schema MVVM în Anexa 2.*

**PAC**  - (Presentation-abstraction-control) – asemănător MVC are 3 componente de bază, presentation – asemenea View – relizează interfața utilizatorului, abstraction – asemenea model – colecționează și procesează date, și control (asemenea controller) care produce *bussiness-*logica aplicației. Spre diferența de MVC, PAC este utilizată ca o structură ierarhicală de agenți, fiecare consistînd din încă o triadă din componente (PAC). Agenții comunică între ei prin partea de control a fiecărui participant, are ca un mare plus – încărcarea foarte rapidă a aplicației și afișarea ageră a interfeței utilizatorului.

*Schema PAC în Anexa 3.*

**HMVC** – (Hierarchical-Model-View-Controller) – asemenea PAC doar că conține agenți compuși din MVC.

Indiferent de șablonul de arhitectură ales, ehipa trebuie să fie cunoscută cu fiecare din ele pentru că este posbil comanda să fie pe un anumit șablon, și ar fi inadmisibil ca echipa să nu cunoască soluțiile și modul de utilizare a unui-a din ele. Uneori e destul cunoașterea unui din ele și astfel e foarte apropiată și structura celorlalte pe motivul asemănării lor, și componentelor chiar complet comune.

**N Layered Architecture** – este modalitate de creare a aplicției structurată pe nivele (layer) . Tehnica de N Layered este bună de utilizat pentru flexibilitatea aplicației și reutilizarea codului, dacă în MVC avem 3 componente, le mai putem numi nivele, într-o aplicație cu N nivele putem avea mai multe nivele, putem crea o aplicație mai complexă dar și mai flexibilă.

Spre exempllu o aplicație client poate avea următoarele nivele:

* Nivelul de infrastructură : Repozitorii, Baze de date
* Nivelul de domen: Servicii, acțiuni, Obiecte
* Nivelul de aplicație: Servicii, transfer de date, *bussiness-*logica
* Nivelul serviciilor distribuite: Controllerele aplicației
* Nivelul prezentației: Notificările utilizatorului, navigarea
* Nivelul de client: Desktop, mobile, tabletă etc.

Fiecare nivel are rolul său specific și responsabilitatea în aplicație. De exemplu nivelul de prezentare este responsabil de prelucrarea interfețelor utilizatorului în dependență de dispozitivul untilizat și logica comunicării browserului. În arhitectura dată nivelele pot exista fără a fi specificat tipul lor, sau numărul lor. Cel mai des în aplicații există 4 nivele de bază: Baze de date, View, *bussiness*, prezentarea. în această arhitectură nivelele comunică între ele secvențial și se evită comunicarea 1 la mai multe deoarece în cazuri specifice datele trebuie prelucrate într-un nivel conector și nu poate fi evitat.

La fel ca și la MVC, nivelele aplicației sunt izolate unul de altul, producînd independența fiecărui nivel, și necunoștința funcționalității și intrumentelor unui nivel de altu. Această arhitectură este foarte apropiată cu structura organizațională și comunicativă tradițională IT care se găsesc în multe companii, făcînd această arhitectură o alegere ideală pentru multe aplicații-*bussiness* cu efort de dezvoltare mai redus.

Din punctul de vedere tehnologic, avem o mulțime de variante unde putem utiliza pattern-ul dat, de exemplu platforma java, webApi, os-construction. Arhitectura nivelată este strîns-propusă în cazurile cînd un programist nu este sigur ce pattern să utilizeze pentru o aplicație, marcînd acest pattern ca un punct de început pentru alegere.

În concluzie despre N layered architecture pot spune că este ușor de dezvoltat, ce aduce un scor înalt acestui pattern, mai mult pentru că nu este greu de implementat; are o performanță scalabilă, în dependenă de numărul de nivele și utilizarea lor și un nivel înalt de testare.

* 1. Stocarea, Repozitoriu

În lucrul în echipă exită multe opțiuni și nuanțe, una dintre ele este stocarea aplicațiilor, proiectelor curente, fișierelor, coodului și pur și simplu ideelor. Inițial pentru aceasta a venit în ajutor fișierele comprimate și expedierea lor prin e-mail, dar aceasta ocupa prea mult timp, mai pe urmă companiile și echipele de programatori profesioniști au trecut la utilizarea serviciilor *Cloud,*  dar acestea erau costisitoare în bani, deși erau comod de utilizat și aveau posibilitatea de a crea o notificare pentru un grup specificat de persoane cu informație respectivă. În sfîrșit programatorii au creat servicii de stocare dinamică numit repozitoriu, care a implementat o mulțime de funcțiuni pentru manipularea cu fișierele, spre exemplu întoarcerea unui fișier la conținutul ultimei salvări.

1.3.2 Git repozitoriu

Instrumentul git permite crearea repozitoriului cu utilizarea serviciilor de control a versiunii, adică fiecare modificare și salvare a fișierului crează o versiune a fișierului și o păstrează aparte, cu posibilitatea în orice timp de a reveni la ea.

Repoziutoriul git prezintă un catalog a fisierelor de sistemă în care se află configurația repozitoriului, indexul ce descrie locul aflării fișierelor din catalog, operațiile asupra lor și un stoc care păstrează însăși fișierele. Structura stocului fișierelor nu reflectă structura reală a fișierelor aflte în lucru, ea se orientează mai mult pe accelerarea îndeplinirii operațiilor cu repozitoriu. În mod standart repozitoriu se păstează într-o directorie ascunsă în directoria de bază a proiectului, cu numele ”.git”, de fapt orice structură de directorii poate fi convertită în repozitoriu și controlată la schimbări.

Repozitoriul git poate fi local sau virtual; repozitoriul local se păstrează în directoriul proiectului, iar cel virtual pe un server, sau dispozitiv de rețea. Accesul la repozitoriul virtual se face cu ajutorul unui URL specific. Cel mai impresionant în lucrul cu Git repozitoriu virtual este că fiecare participant deține o copie în repozitoriul său local, adică modificările în proiect pot fi mai întîi revizuite de superiorul pe proiect apoi implementate în repozitoriul central (virtual).

Vizual intrumentul este prezentat în forma liniei de comandă cu setul de comenzi bine definite și descrise de autor, aporopo autorul versiunii inițiale este Linus Torvalds (creatorul sistemului de operare Linux); Deasemenea mai există și multe modificări cu comenzi modificate, interfețe vizuale mai ușor de înțeles și utilizat, ca exemplu intrumentul git implementat în windows explorer **TortoiseGit.**

1.3.3 SVN repozitoriu

SVN – la fel cunoscut ca Subversion este o sistemă liberă centralizată pentru controlul versiunilor fișierelor. Subversion se foloseste de o mulțime de programatori în toată lumea pentru simplitatea lui și varietatea de posibilități.

Subversion propune două tipuri de organizare a repozitoriilor; prima variantă folosește baze de date pentru stocarea fișierelor, iar al doilea variant folosește fișiere cu format special în directoriul proiectului, asemănător git. Ambele tipuri de repozitorii oferă suficientă siguranță și ușurință în lucrul cu versiunile fișierelor. Deasemena subversion poate folosi diferite sisteme de memorare a fișierelor și arhivare a lor pentru economie de spațiu la proectele mari, totul revenind la dispoziția utilizatorului.

1.3.4 SVN sau Git ? care este alegerea perfectă ?

Deși ambele instrumente sunt ușor de folosit, și produc aceleași servicii, există și diferențe între ele pentru care programatorii le dau preferința unui sau altui intrument, ce este bun în unul și ce este bun în altul?

Git folosește două tipuri de ropozitorii simultan (local și virtual) pe cînd Svn folosește doar unul, Git folosește 3 comenzi pentru actualizarea repozitoriului virtual, deci și 3 ori mai multi timp pe cînd svn folosește doar 1, Git este cross-platform, adică lucrează și pentru windows și linux etc. concomitent, pe cînd svn poate lucra doar cu 1 într-un timp.

**Capitolul II**

* 1. Elaborarea Web Api

Într-o echipă elaborarea unei aplicații web începe de la planificarea proectului, propunerea ideelor de programatori profesioniști, sau de client, și depistarea căilor mai ușoare în îndeplinirea lor. După aceasta se petrece repartizarea rolurilor și responsabilităților fiecărui participant; pentru aceasta întreaga echipă este divizată în departamente și subdepartamente.

Planul proectului este divizat pe cîțiva pași importanți pînă a ajunge la realizarea aplicației și a proectului întreg.

2.1.2 SDLC (Software-Development-LifeCycle) – este ciclul de dezvoltare a unei aplicații, o serie de stări generale prin care trece orice sistemă programată, indiferent de complicitatea aplicației.

1) Planificarea sistemei în care echipa decide ce și cum va fi creat și ce probleme pot apărea în calea aleasă și să le rezolve din:

* depistarea problemelor, scopurilor și resurselor
* înțelegerea posibilităților rezolvării alternative prin comunicarea cu clientuții, consultanți sau colaboratori;
* posibilități de a crea un product mai bun decît la concurenți

2) Analiza sistemei – partea în care trebuie de determinat și documentat cerințele finale ale sistemei.

3) Design-ul sistemei – decide elementele sistemei, componentei, nivelul de securitate, modulele și arhitectura.

4) Dezvoltarea și implementarea pordusului – partea în care se scrie codul programei, se elaborează partea grafică și responsivă. La această etapă sistema poate fi prezentată clientului.

5) Exploatarea și integrarea sistemei – partea de testare activă a sistemei unde se depistează greșelile admise în logica sistemului sau a careva opțiuni.

6) Susținerea sistemei (maintance) – partea de actualizari periodice a sistemei și menținerea ei în stare performantă, schimbul componentelor la neceistate sau comandă.

Acestea sunt șase etape de dezvoltare a oricărui produs soft, care pot fi supuse modificărilor în dependență de preferințele echipei. Astfel există cîteva tipuri de SDLC, cel mai bine cunoscut și des utilizat de o multitudine de echipe este Agile SDLC.

Agile SDLC constă în împărțirea întregului proiect pe secțiuni numite sprint, care cel mai des durează două săptămîni, unde fiecare sprint este împărțit pe etape conform celor șase a SDLC clasic, în așa fel fiecare 2 săptămîni echipa poate prezenta o parte a proiectului care lucrează conform logicii cerute de client; deasemenea poate fi elaborat mai întîi componentul cerut de client pentru demonstrație, căci ordinea organizării sprinturilor nu are importanță majoră.

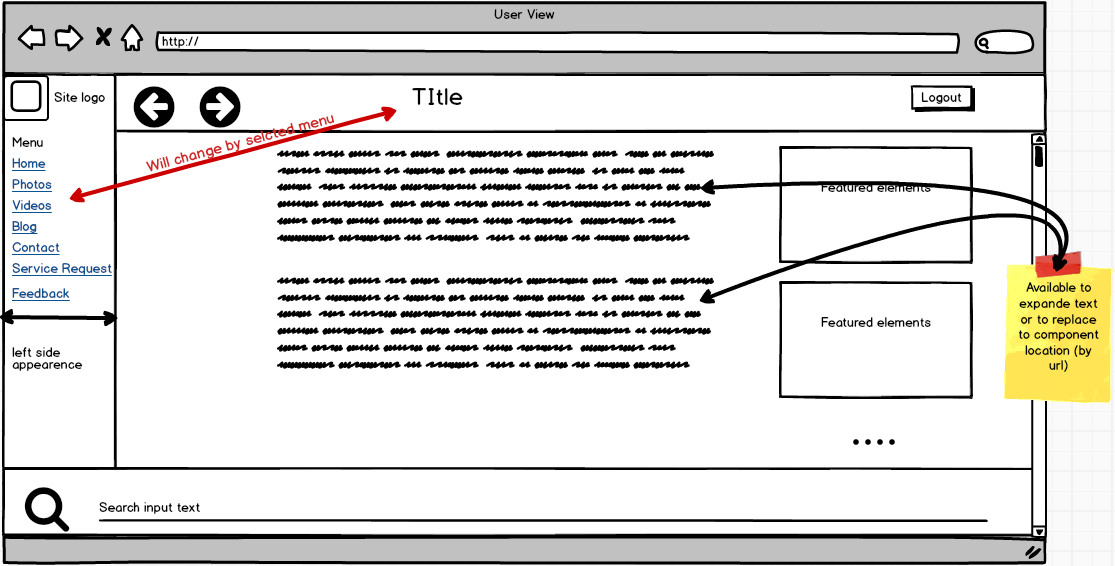
Waterfall - cea mai veche metodologie ce constă din cinci pași asemănători celor de mai sus, planificarea, design-ul, implementarea, verificarea și întreținerea. Un mare minus acestei metodologii, și cred că motivul principal a ignorării acestei metodologii este lucrul asupra întregului proect simultan, adică fiecare component este suspus mai întîi planificării, apoi design-ului ..; Din acest motiv realizarea proectelor voluminoase devine foarte dificilă, practic această metodologie este bună la proiectele de un singur component sau două …

2.1.3 Prezentarea generală, imaginea aplicației

Etapa de planificare și design se finalizează cu o imagine (sau mai multe) ce prezintă cum ar trebui să arate aplicația (sau componentul) în final. Prezentarea machetelor poate să fie realizată de client (prezentînduși dorința) sau de echipa de la companie (prezentînd propuneri), avînd ca scop eliminarea totală a neclarităților de echipa de programiști și designeri. Machetele pot fi atît simple cît și foarte foarte complicate, prezentînd întreaga arhitectură de componente și dezvoltări a aplicației.

Există multe instrumente de prezentare a machetelor, începînd de la desenul de mînă, paint pînă la instrumente mai complicate (photoshop, Apimetrics, IBM Bluemix). Pentru machete aplicațiilor web putem folosi intrumente destul de simple dar cu conținut bogat; pentru proectul nostru vom folosi Balsamiq.

Balsamiq este un set de instrumente pentru crearea machetelor interfeței utilizatorului cu descrieri, efecte, drop-menu și alte efecte de apariție-despariție (simularea vizuală a unei aplicații reale).



*img 1. Machet pagina principală utilizînd Baslsamiq*

*Toate machetele proectului sunt anexate în anexa 3.*

2.1.4 Machetul comportamentului aplicației, *bussiness-*logica

Pentru înțelegerea funcțiunilor aplicației și a ceea ce trebuie să conțină în fiecare component, dezvoltatorii de aplicații au apelat la diagrame de descriere a funcționalității aplicației, și limbaje universale de concepere a arhitecturii. Cel mai bun exemplu și intrument utilizat de companii voluminoase este UML-diagrams. UML – (Unified Modeling Language) prezintă colecție de instrumente pentru descrierea aplicațiilor, componentelor, posibilităților utilizatorului și a totul ce ține de dezvoltarea aplicației. UML folosește diagrame asemănător schemelor-bloc clasice, și împarte descrierea aplicației în cinci părți principale (cinci feluri de diagrame)(sunt posibile mai multe feluri de diagrame).

1. Diagrama claselor – diagrama ce conține descriera claselor, comunicării între ele, pentru fiecare component, dependențele și conținutul lor.
2. Diagrama componentelor – diagrama ce conține descrierea componentelor și relațiile între ele.
3. Diagrama de activitate – diagrama ce descrie comportamentul aplicației întregi la primirea diferitor date de intrare.
4. Diagrama de scenarii de utilizare – diagrama ce descrie interactivitatea între activități și utilizatori.
5. Diagrama de structură – prezintă descrierea arhitecturii generale a aplicației și interacțiunea cu mediul înconjurător (utilizatori, alte aplicații, componente)

În cele mai des situații pentru descrierea unei aplicații este destul de utilizat diagramele claselor deoarece este o structură statică ce descrie atributele claselor, dependența între clase, compoziția aplicației. Cel mai puternic punct este că diagramele pot fi construite de o persoană neimplicată în procesul de dezvoltare a aplicației, fie clinetul sau un arhitect angajat special.

2.1.5 Împărțirea sarcinilor

Cum a fost menționat mai sus, în lucru la un poiect mare echipa este divizată pe departamente și subdepartamente, unde fiecare participant al echipei servește cîte un rol. Fiecare rol îndeplinește oarecare sarcină și pentru menajarea acestor sarcini șefii de departamente au nevoie de un ajutor special pentru controlul angajarii sarcinii și pentru controlul îndeplinirii lor. În acest ajutor managerilor le vine în ajutor masa de sarcini.

În lumea IT masa de sarcini este un instrument virtual (soft) care simulează masa reală în care sarcinile se divizează în patru secțiuni majore - sarcini de angajat, sarcini în proces, sarcini de verificat, și sarcini rezolvate. Cel mai des liderii departamentelor utilizează metodologia scrum sau kanban; Ambele metodologii se bazează pe masa de sarcini, și au puncte comune și deosebiri.

Kanban – metoda de management care realizează principiul ”în termen” și ajută la distribuirea egalată a sarcinilor între angajați. Cu această metodă întreg procesul de dezvoltare devine transparent pentru toți participanții. Kanban are patru principii de bază în managementul sarcinilor:

* Fundamentul se pune pe metode existente de dezvoltare.
* Înțelegerea din timp a efectuării schimbărilor importante
* Respectul către regulile și rolurile existente
* Promovarea inițiativei.

Scrum – set de principii pe care se construiește procesul de dezvoltare care permite împărțirea procesului de dezvoltare în mici segmente de timp controlate de liderul echipei. Scrum este mai bun în componență cu metodologia agile de dezvoltare a aplicației deoarece sprintul este ușor de controlat cu masa de sarcini metodologiei scrum.

Deosebirea generală între aceste două metodologii este că scrum este controlat în timp și are restricție în realizarea sarcinilor angajate.

Principalul component al acestor metodologii este sarcina care conține descierea, nota de complexitate și timpul estimat pentru sarcina dată (în cazul scrum și restricția în timp). De fiecare dată cînd un praticipant se angajează la o sarcină, sarcina își schimbă secțiunea la sarcina în proces și se fixează numele lucrătorului și timpul de început. La finalizarea sarcinii ea se schimbă în secțiunea de verificare, unde este supusă testelor de logică și greșeli în lucru conform cerințelor task-ului, deasemenea poate fi verificată de lider-ul echipei, și în sfîrșit ajunge la categoria de sarcină finalizată.

Pe lîngă toate acestea oportunități managementul echipei și a dezvoltării aplicației mai intervin alte activități capabile a aranja diferite nereuglarități. În jurnalul de zi al echipei sunt adunări zilnice sau generale sau speciale, unde se discută aranjarea sarcinilor, punctele tari și slabe, analiza sarcinilor angajate, posibilități de rezolvare alternativă și consultarea ideilor. Pentru analiza sarcinilor la inițierea