LUNDS UNIVERSITET

Utkast, Rapport, IS-Projekt
2014-05-31

Alexander Sjöman, 890821-6271
Philip Gregorius, 920703-4290
Ebba Gullstrand, 920629-5744
Elias Ulén, 920826-4698

Handledare: Erdogan Ucan

1 Dokumentation av praktikfall
1.1 Bakgrund
1.1.1 Mål
1.1.2 Syfte
1.1.3 Projektbeskrivning
1.2 Systembeskrivning
1.2.1 C# - Fitnessbuddy
1.2.1.1 Felhantering
1.2.1.2 GUI
1.2.1.3 Struktur
1.2.1.4 Kravspecifikation
1.2.1.5 Projektutrustning
1.2.2 EJB - FitnessWebbuddy
1.2.2.1 Felhantering
1.2.2.2 GUI
1.2.2.3 Struktur
1.2.2.4 Kravspecifikation
1.2.2.5 Projektutrustning
1.2.3 Webservice - Igualis Webservice + Clients

1 Dokumentation av praktikfall

1.1 Bakgrund

Vi fick i uppdrag att skriva ett progam som utvecklades i .NET miljön. Programmet skulle använda sig av en databas.

1.1.1 Mål

1.1.2 Syfte

Projektets syfte är att använda kunskaper från tidigare terminer och använda dessa i nya arbetsmiljöer,,,,,,,,,,,

1.1.3 Projektbeskrivning

1.2 Systembeskrivning

1.2.1 C# - Fitnessbuddy

Igualis Fitnessbuddy är en applikation utformad med syftet att assistera den vardagliga personen med viktminskning. Detta i form av ett kalorischema som applikationen genererar baserat på vikt, längd, kön och hur aktivt personen i fråga tränar. Kalorischemat sparas sedan mot vår databas för den inloggade användaren.

Vid utformningen lades det stor tyngd på att göra systemet användarvänligt. Anledningen till detta tillvägagångssätt var att vi inte hade en bestämd målgrupp, applikationen ska vara tillgänglig för alla olika generationer. Personer med eller utan datorvana ska enkelt förstå systemets funktionalitet och navigera fritt inom det.

1.2.1.1 Felhantering

För att alla olika typer av användare ska kunna använda applikationen är funktionaliteten i Fitnessbuddy relativt begränsad, användaren avgränsas med möjligheter att göra fel. Felhanteringen inom applikationen är därmed inte databasrelaterad och har istället stor fokus på vad som matas in av användaren i gränssnittet. Detta görs med hjälp av en extern felhanteringsklass, FailSafe (se bilaga 1)

1.2.1.2 GUI

Gränssnittet utvecklades i XAML. Detta gav oss större möjligheter än Windows Forms för att skapa ett dynamiskt gränssnitt, som vi dessutom kunde sätta en egen prägel på. För att XAML-gränssnittet skulle vara kompatibelt med vår .NET kod är vårt projekt utformat genom en WPF-applikation. Till skillnad från Windows Forms har vi två klasser för varje form, en klass med bakomliggande .NET kod som är länkad till XAML-klassen, vilken innehåller all visuell kod.

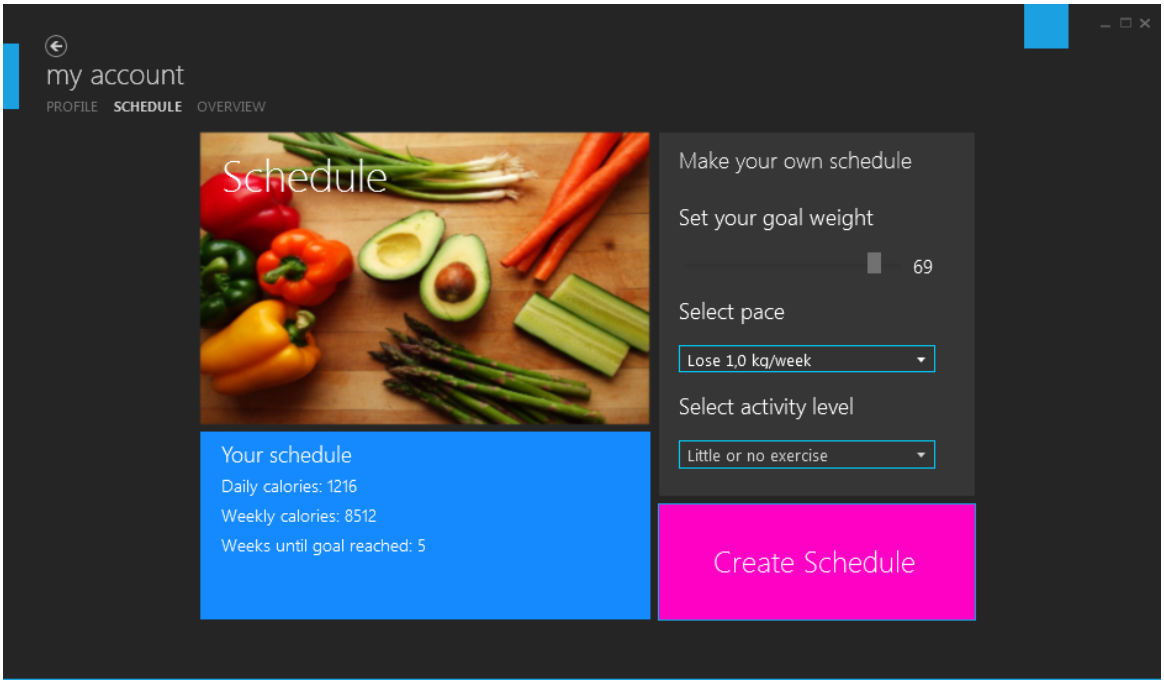
För att applikationen ska fungera så bra som möjlighet oberoende av vilken typ av skärm och upplösning man har, finns ett förutbestämt värde på storleken på alla forms. Inget fullscreen läge finns tillgängligt, det är ingen egenskap som applikationens funktionalitet kräver. Färg- och strukturmässigt har vi i designprocessen varit delvis innovativa men ändå använt oss av beprövade alternativ gällande användarvänlighet.

Vårt GUI är skapat på engelska, dels för att alla ska kunna använda applikationen men även för det kändes mest relevant eftersom all bakomliggande kod är namngiven på engelska.

Vid uppstart av applikationen möts användaren av en inloggingsskärm. Genom en inloggning vet applikationen exakt vilken användare och eventuellt dennes schema som ska hämtas från databasen. Möjligheten att skapa en användare på inloggningsskärmen.

Figur 1 - Fitnessbuddy LogIn

Eftersom användarvänlighet har varit en en prioritet under projektets gång har vi hållit antalet grafiska kompontenter till ett minimum. Användaren har möjlighet att navigera bland tre flikar, profile, schedule och overview. Efter inloggning tas användaren till profile, där personliga uppgifter och mål finns tillgängliga och möjligheten att uppdatera dessa. På schedule-fliken genereras själva kalorischemat. Schemat är personligt och kan bara nås av den användare som är inloggad. Overview-fliken är ett komplement till Schedule, där uppdaterar användaren sin nuvarande vikt och hur många veckor av schemat som är avklarat.



Figur 2 - Fitnessbuddy ScheduleView

1.2.1.3 Struktur

Vi har följt MVC-modellen när vi strukturerat applikationen. Eftersom vi även jobbat mot en databas har vi använt oss av en DataAccessLayer, som hanterar anslutningarna till databasen och exekverar SQL-kommandon. Metoderna i DAL:en få och är generiskt utformade för att slippa massiv kod. T.ex använder vi samma metod (CreateData) för att skapa data i tabellen Account liksom Schedule, men de anropas på olika sätt från gränssnittet.

```
1 reference
public void CreateData(List<string> inputData, string tableName)
{
    SqlCommand command;
    string sqlQuery = string.Format("INSERT INTO {0} VALUES (", tableName);

    for (int i = 0; i < inputData.Count; i++)
    {
        if (inputData.Count - 1 != i) sqlQuery += "'" + inputData[i] + "', ";
        else sqlQuery += "'" + inputData[i] + "')";
    }

    command = new SqlCommand(sqlQuery, con);
    command.ExecuteNonQuery();
}
```

Figur 3 - Generisk metod i DataAccessLayer

Vidare har vi en controller som agerar mellanhand för GUI och DAL:en , som säkerställer att det inte sker någon indirekt kommunikation mellan klasserna och endast genom controllern. I gränssnittet kallar vi sen på funktioner ur controllern.

Vid kodandet av all back-end kod har vi använt generella naming conventions för att följa bästa praxis. Vi har använt s.k lower camel case för variabler (ettExempel) och s.k upper camel case för identifierare (EttExempel).

1.2.1.4 Kravspecifikation

- Följa MVC även fast WPF och XAML använts
- Användarvänligt
- Ingen felinmatning ska kunna göras

- TODOLIST elias
- mer? eller ta bort kravspec? klassik koppling inte här men annat :P rem1nder
- aaa typ varför design. heuristics? golden rules? motivera mer varför vad implementweats

1.2.1.5 Projektutrustning

Mjukvara:

- Visual Studio 2013 Ultimate
- GoogleDrive
- OpenOffice4 Writer
- SQL Server 2008 R2 Management Studio Express (x64) - (English)
- TeamViewer 9

Hårdvara:

- Windows respektive Mac-dator

1.2.2 EJB - FitnessWebbuddy

Syftet med EJB-applikationen är att flytta en vy av C#-applikationen, koda den i Java och ta den till webben. FitnessWebbuddy kan därmed endast söka och uppdatera användare och scheman. Ingen borttagning är möjlig och personliga uppgifter samt inloggningsuppgifter kan inte ändras från webben. FitnessWebbuddy fungerar således som en enklare typ av övervakning av sitt kalorischema, ett komplement till desktop-applikationen om man inte t.ex. har tid att ladda ner den.

1.2.2.1 Felhantering

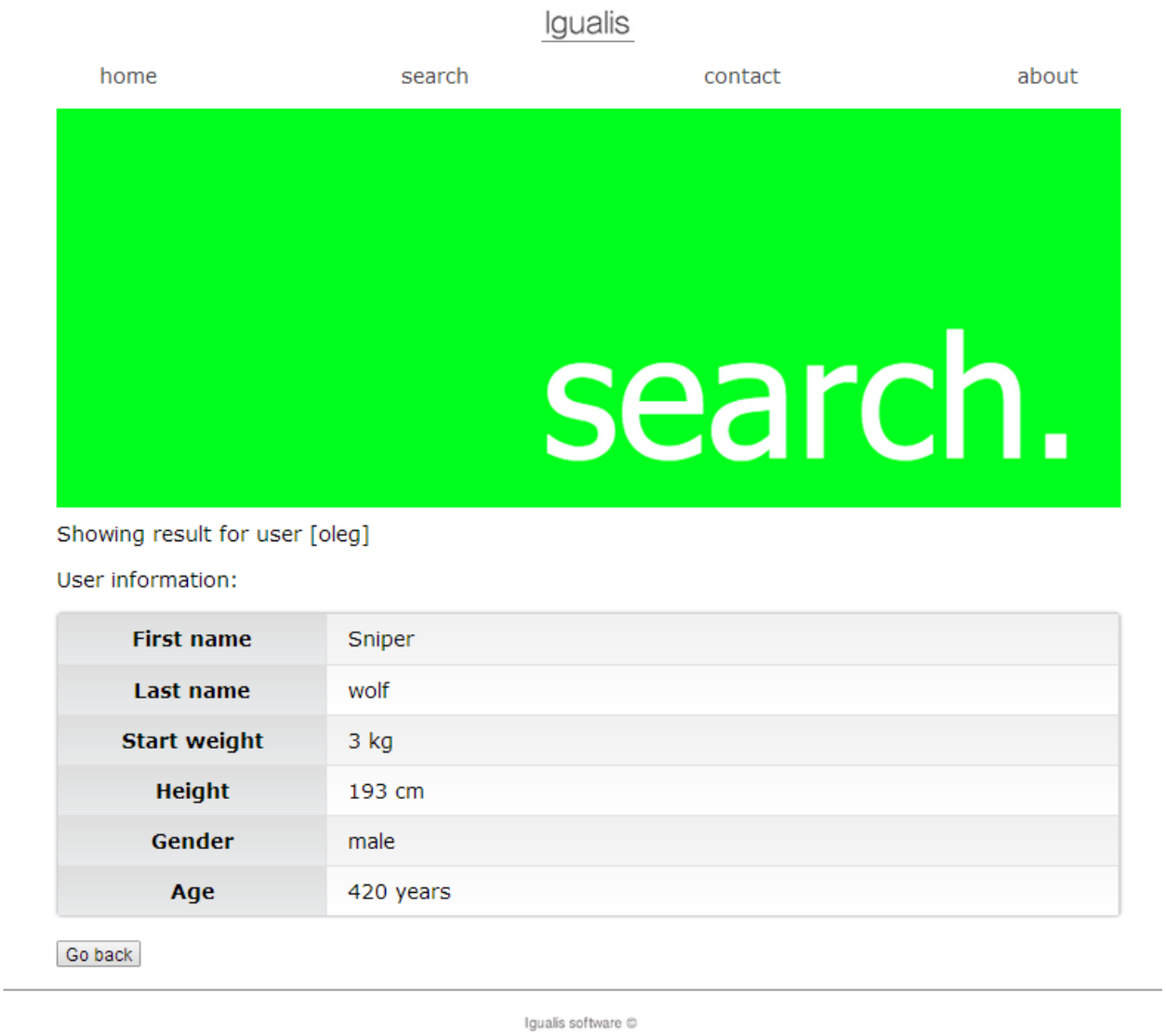
Det finns ingen övergripande felhantering i vår EJB-applikation, men som komplement har vi använt JUnit, vilket är en form av enhetstesting för javaapplikationer. Enhetstesten fungerar som en simulering av användartest, den testar alla metoder i en böna.

1.2.2.2 GUI

Utformningen för EJB-applikationen består av olika .jsp och .html filer och är således ett webgränssnitt. Navigation inom FitnessWebbuddy sker genom fyra olika flikar, home, search, update och testing. För att få en liknande design på alla olika flikar har vi använt oss av en .css fil som länkats till de olika sidorna bakom flikarna. I denna fil satte vi standarder vi ville att .jsp och .html filerna skulle ha.

Eftersom hemsidan endast fungerar som ett uppdaterings- och övervakningsverktyg krävs ingen inloggning. Istället för en inloggningsskärm navigeras användaren till home vid uppstart.

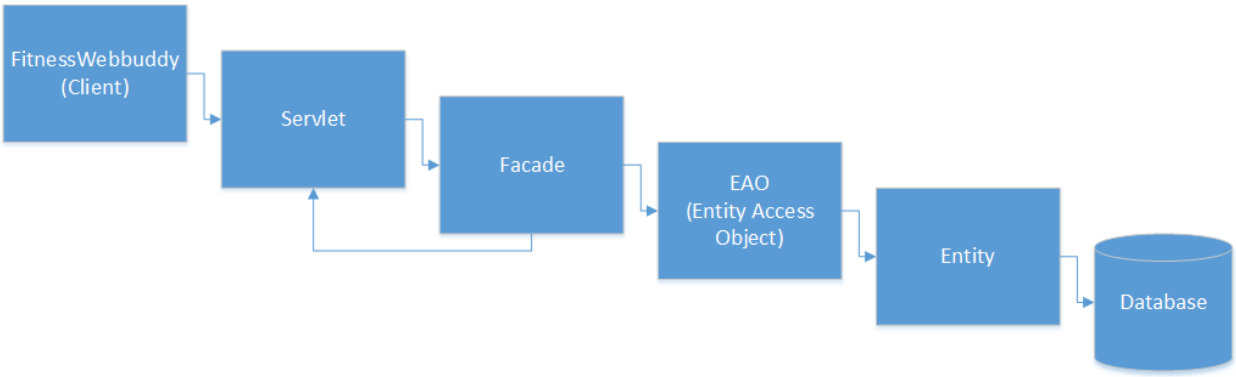
Användaren kan söka mot databasen, t.ex. på sitt användarnamn under fliken search. Klickar man på fliken navigeras man till search.jsp, vilken består av klassisk html-kod kombinerat med javakod.



Figur 4 - FitnessWebbuddy Search

1.2.2.3 Struktur

Strukturen på EJB-applikationen består av olika patterns och komponenter. I .jsp sidorna (client) finns det dynamiska anrop i form av javakod till vår servlet som innehåller metoderna doGet() och doPost(). Vad som skickas till servleten beror på vad användaren matar in i klientvyn. En operation skickas från .jsp sidan till servleten med inmatningen som inparameter. Anropen skickas sedan vidare till en session stateless bean (Facade), vilket i sin tur skickar vidare till en annan session stateless bean, vår EAO.



Figur 5 - FitnessWebbuddy Struktur

1.2.2.4 Kravspecifikation

??

1.2.2.5 Projektutrustning

Mjukvara:

- GoogleDrive
- OpenOffice4 Writer
- SQL Server 2008 R2 Management Studio Express (x64) - (English)
- TeamViewer 9
- Jboss 7.1
- Notepad++
- Eclipse EE

Hårdvara:

- Windows respektive Mac-dator

1.2.3 Webservice - Igualis Webservice + Clients

Igualis Webservice

Beskrivning av gränssnitt från de olika projekten

Egen utvecklad applikation - Igualis Fitness buddy

Vi hade ingen tidigare erfarenhet med XAML när vi började med gränssnittet för vår fitness applikation byggde vi detta kring ett ramverk som vi sedan modifierad. Det var uppbyggt kring tre stycken olika vyer. Ett för inlogg vilket körde vyn direkt på fönstret, detsamma gällde vår settings vy, där användaren kunde ändra information om sin profil. Huvudmenyn kontrollerades genom separata user-controlls vilka var länkade genom koden i vyns fönster, det var således endast länkarna som renderades direkt på vyn och valda vyer från user-controlls laddades in efter förfrågan. Detta minimerade pop-ups och plotter i gränssnittet och gav programmet ett bra flow. Det gjorde dessutom felhanteringen under utvecklingen enklare eftersom man kunde se exakt vilken vy som inte laddades.

Webbservice

Under webbservice hade vi egentligen större planer för gränssnittet men pga tidsbrist han vi inte modifiera tillräckligt mycket. Det vi försökte göra denna gången var den enkla möjligheten i XAML att overrida usercontrolls för att få de att se ut som vi vill.

Bilaga 1 - Förtydligande av klassen FailSafe

```
8 references
public string CheckInput(List<string> input)
{
    foreach (string data in input)
    {
        if (data == "")
        {
            return ErrorMessage("EMPTYINPUT");
        }

        if (!Regex.IsMatch(data, "^[a-zA-Z0-9]+$") || data.Contains(" "))
        {
            return ErrorMessage("ILLEGALINPUT");
        }
    }
    return "CORRECT";
}
```

Figur 1 - Metoden CheckInput i klassen FailSafe, kollar om användarens input är korrekt

```
if (failSafe.CheckInput(dataList) == "CORRECT")
{
    if (ctrl.GetData("username", "Account", "username", dataList[0]) == "")
    {
        ctrl.CreateData(dataList, "Account");
        enableLogInScreen();
    }
    else labelMessage.Content = failSafe.ErrorMessage("ACCOUNTEXISTS");
}
else labelMessage.Content  = failSafe.CheckInput(dataList);
```

Figur 2 - Metoden CheckInput anropas från gränssnittet, t.ex. kollar den att användare inte redan finns vid skapande

```
7 references
public string ErrorMessage(string input)
{
    switch (input)
    {
        case "EMPTYINPUT":
            return "One or more fields are empty.";
        case "WRONGLOGIN":
            return "You have entered wrong username and/or password, please try again.";
        case "PASSWORDMISMATCH":
            return "The specified old password does not match with the current password.";
        case "ILLEGALINPUT":
            return "Only letters and numbers are allowed.";
        case "SCHEDULENOTEXIST":
            return "Schedule does not exist.";
        default:
            return "ERROR";
    }
}
```

Figur 3 - Metoden ErrorMessage i klassen FailSafe som genererar felmeddelanden till gränssnittet