



VYŠŠÍ ODBORNÁ ŠKOLA, STŘEDNÍ PRŮMYSLOVÁ ŠKOLA  
A JAZYKOVÁ ŠKOLA S PRÁVEM STÁTNÍ JAZYKOVÉ ZKOUŠKY

# **PRAKTICKÁ MATURITA**

## **Vývoj aplikací s využitím architektury MVC**

Zadání práce (druhý list) – kopie zadání, kterou žák obdržel od svého vedoucího

**Čestné prohlášení:**

Prohlašuji, že jsem praktickou maturitní práci vypracoval samostatně a použil jsem pouze literatury uvedené v soupisu.

Souhlasím s trvalým umístěním mé práce ve školní knihovně, kde bude k dispozici pro potřeby školy.

V Kutné Hoře dne

podpis

## **Anotace**

Praktická maturitní práce je zaměřena na webovou aplikaci vytvořenou na frameworku ASP.NET podle MVC (Model – View – Controller) architektury a její vývoj. V práci je popsán postup práce se systémem pro verzování GitHub a vysvětlená funkce webové aplikace. Práce také obsahuje stručný popis programovacího jazyka C#, objektově orientovaného programování, frameworku ASP.NET a podrobnější vysvětlení architektury MVC s ukázkami konkrétního kódu z aplikace.

## **Annotation**

This work is focused on a web application created on the ASP.NET framework based on the MVC (Model – View – Controller) architecture and its development. The work contains a description of GitHub workflow and an explanation of the function of web application. It also contains a brief description of the C# programming language, ASP.NET framework and a bit more detailed explanation of MVC architecture with code demonstration.

## Obsah

|   |    |
|---|----|
| 1. Definice a historie webové aplikace .....    | 1  |
| 2. Úvod.....                                    | 1  |
| 3. GitHub.....                                  | 2  |
| 3.1. Slovník pojmů.....                         | 3  |
| 3.2. Založení nového repozitáře v GitHubu ..... | 3  |
| 3.3. Vytvoření větve (branch).....              | 4  |
| 3.4. Commitování a pushování změn .....         | 4  |
| 3.5. Pull request a merge .....                 | 5  |
| 3.6. Klonování .....                            | 6  |
| 3.7. Řešení konfliktů.....                      | 7  |
| 4. C# a OOP .....                               | 8  |
| 5. ASP.NET .....                                | 8  |
| 6. MVC.....                                     | 8  |
| 7. Webová aplikace.....                         | 10 |
| 7.1. Model (databáze) .....                     | 10 |
| 7.2. View (pohled).....                         | 12 |
| 7.3. Controller (řadič) .....                   | 16 |

## 1. Definice a historie webové aplikace

Webová aplikace v softwarovém inženýrství je aplikace poskytovaná uživatelům z webového serveru přes počítačovou síť Internet, nebo její vnitropodnikovou obdobu (intranet). Webové aplikace jsou populární především pro všudypřítomnost webového prohlížeče jako klienta. Ten se pak nazývá tenkým klientem, neboť sám o sobě logiku aplikace nezná.

Schopnost aktualizovat a spravovat webové aplikace bez nutnosti šířit a instalovat software na potenciálně tisíce uživatelských počítačů je hlavním důvodem jejich oblíbenosti. Webové aplikace jsou používány pro implementaci mnoha podnikových i jiných informačních systémů, ale i freemailů, internetových obchodů, online aukcí, diskusních fór, weblogů.

Podstatnou výhodou vývoje webových aplikací stavějících na standardních funkcích prohlížeče je jejich schopnost pracovat podle určení bez ohledu na operační systém či jeho verzi instalovanou na daném klientském počítači. Místo psaní variant aplikace pro Windows, Linux, Mac OS X a další operační systémy stačí teoreticky aplikaci napsat jednou a nabídnout téměř kdekoliv.

V dřívějších typech aplikací typu klient-server měla každá aplikace svůj vlastní klientský program, který sloužil jako její uživatelské rozhraní a musel být instalován na osobním počítači každého uživatele. Aktualizace serverové části typicky vyžadovala i aktualizaci klientských programů na každé pracovní stanici, což zvyšovalo náklady na podporu a snižovalo efektivnost zaměstnanců.

## 2. Úvod

Ještě před samotným vývojem, je potřeba dohodnout se, v jakém jazyce bude program napsaný, v jakém prostředí a jelikož programování od vynálezu OOP už dlouho není o tom, že jeden člověk píše celý program sám, pro větší efektivitu je potřeba rozdělit části programu mezi členy týmu podle jejich předností a zkušeností.

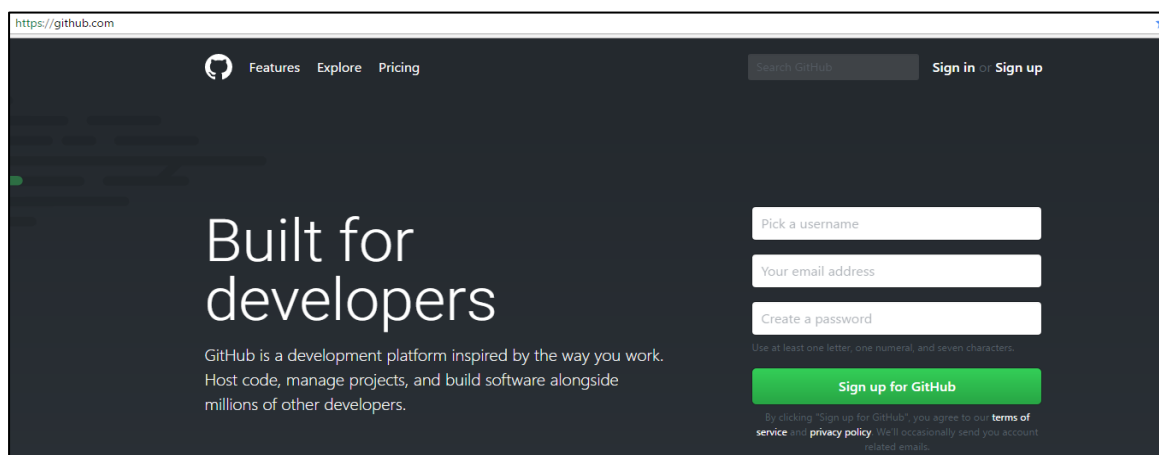
Pracování více lidí na jednom projektu není ovšem tak jednoduché. Struktura programu se pořád na základě úprav mění a jestli všichni členové týmu nemají aktuální verzi,

mohlo by se stát, že změny, které jeden z nich provede už vůbec nejsou relevantní, protože část jeho programu využívá kód, který už v aktuální verzi neexistuje nebo funguje na úplně jiném principu.

Právě pro zabránění takovýmto chybám se při týmovém vývoji využívá Git. Git je systém pro spravování verzí projektu. Aby ale k těmto verzím byl pohodlný a jednoduchý přístup je potřeba je uložit na nějaké dostupné místo. K tomu se využívají služby GitHubu. Ten poskytuje bezplatný hosting a aplikaci s jednoduchým prostředím pro organizaci.

### 3. GitHub

Pro používání služeb GitHubu je potřeba si založit GitHubový účet. To se dá udělat na webových stránkách - [github.com](https://github.com).



Obr. č. 1

Po založení účtu doporučuji stažení desktopové aplikace. Tu najdete na - [desktop.github.com](https://desktop.github.com). Po přihlášení do aplikace můžete založit nový repozitář.

### 3.1. Slovník pojmů

repozitář (repository) – Zastiťuje celý 1 projekt, včetně všech souborů, verzí souborů, větví, úkolů a přístupů.

větev (branch) – Je to vývojová linie vašeho projektu. Paralelně jich povětšinou existuje více. Například si to představte tak, že 1 větev je verze vašeho systému, která je plně funkční a nasazena v ostrém provozu, a ve druhé větvi implementujete nové funkce, nebo opravy. Jakmile je práce na dané větvi dokončena, obvykle ji pak sloučíte s vaší hlavní větví (master).

hlavní (master) větev – Takto označovaná větev obvykle obsahuje vaši produkční verzi projektu, kterou aktuálně používáte v ostrém provozu. Ve většině případů je právě tato větev výchozí větví pro vytváření nových větví.

commit – Uložení provedených úprav a vytvoření skupiny změn na daných souborech. Každé toto uložení vyžaduje váš komentář, například označení co se změnilo.


push – Odeslání všech posledních provedených commitů na server.

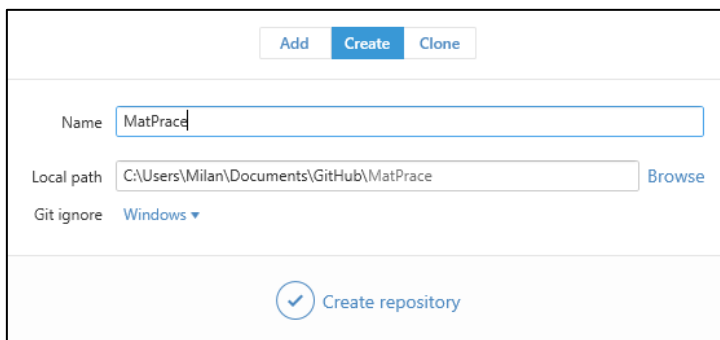
pull – Aktualizace vašich lokálních souborů ze serveru . Vždy to aktualizuje pouze tu větev, pro kterou chcete aktualizaci provést.

merge – Sloučení obsahu větví.

### 3.2. Založení nového repozitáře v GitHubu

Začnete kliknutím na plus

 v levém horním rohu, které vám otevře okno kam zadáte jméno repozitáře a místo kam se má lokálně ukládat.

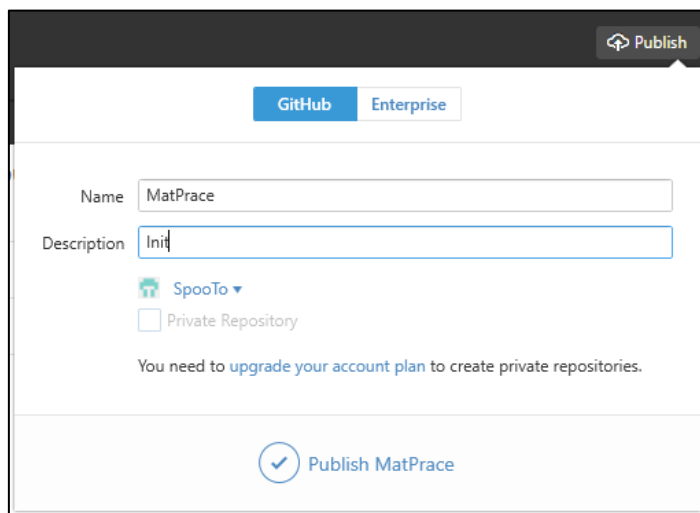


Obr. č. 2



To vytvoří lokální repozitář. Ten se na servery GitHubu uloží až po publishnutí, které najdete v pravém horním rohu.

Můžete si všimnout, že GitHub automaticky vytvořil dvě složky – `gitattributes` a `gitignore`.



Obr. č. 3

`Gitattributes` slouží k rozpoznávání souborových typů a `gitignore` určuje soubory, které bude GitHub při synchronizaci ignorovat.

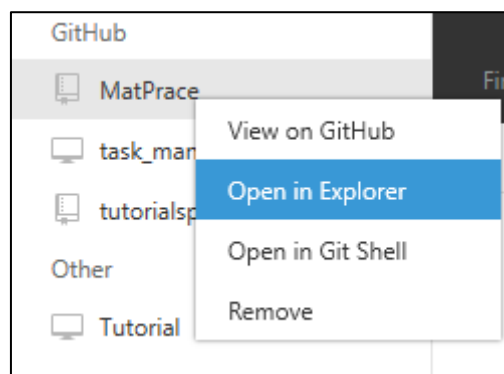
### 3.3. Vytvoření větve (branch)

Větvení umožňuje práci na různých verzích repozitáře najednou. Od vytvoření repozitář obsahuje větev `master`. Tu je znova potřeba publishnout na server.

Když vytvoříte větev od `masteru`, vytvoříte klon `masteru`. Jestli někdo udělá na `masteru` změny, zatím co vy pracujete na své větvi, můžete změny, které udělal pullnout do své větve.

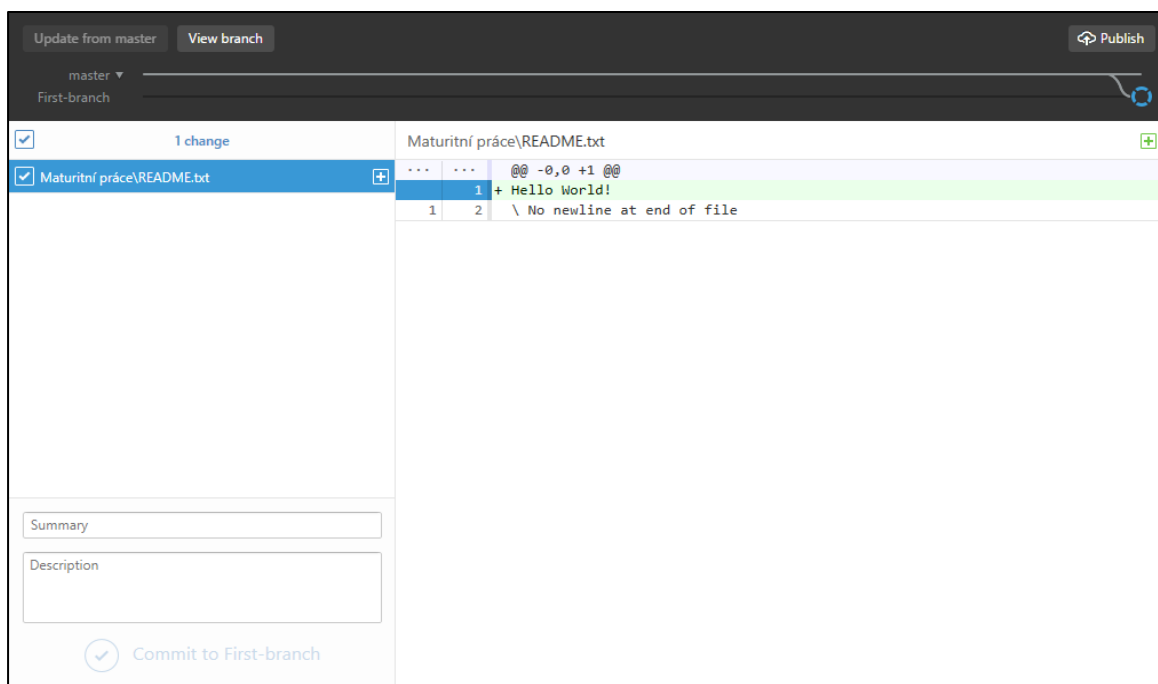
### 3.4. Commitování a pushování změn

Do lokálního souboru se můžete dostat pravým kliknutím na složku repozitáře v GitHubu a následně kliknutím na “Open in Explorer”.



Obr. č. 4

Po vytvoření textového souboru README se v aplikaci ukážou změny v souboru.



Obr. č. 5

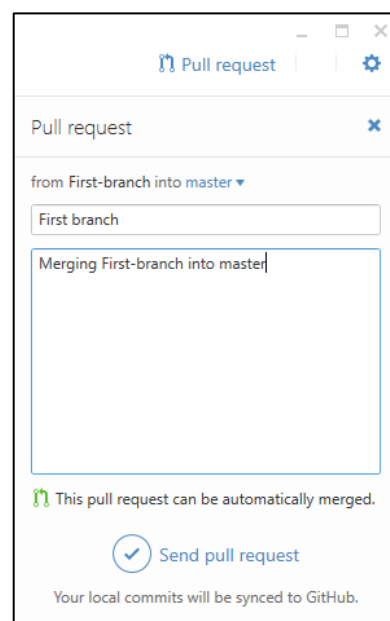
Tyto změny následně můžete commitnout. Stačí vyplnit název změny a případně nějaký komentář vysvětlující co se změnilo.

Tento commit pak můžete zesynchronizovat se serverem (pushnout). To uděláte kliknutím na Sync v pravém horním rohu.

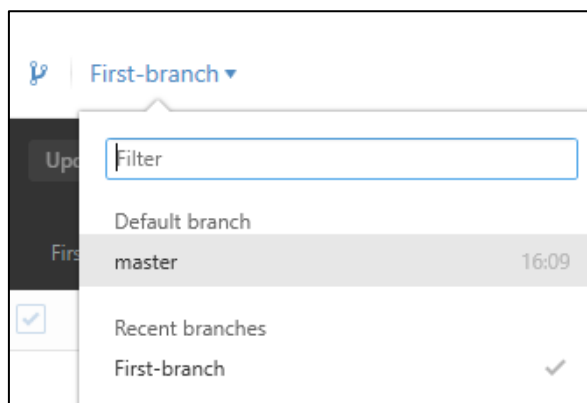
### 3.5. Pull request a merge

Teď můžete změny, které jste ve vedlejší větvi udělali, přivést zpět do masteru. Začneme s pull requestem. Ten oznamuje že jste hotovi se svou větví a je připravena na merge s mastrem. Ten uděláte tak, že přejdete zpět do masteru a porovnáte ho s vaší větví.

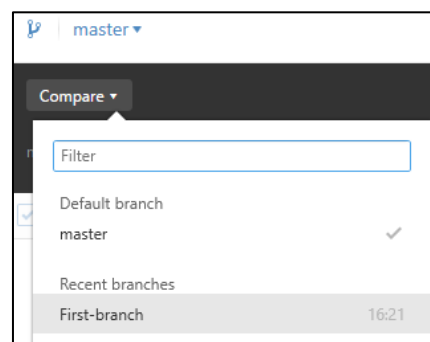
Následně se vám objeví možnost Update from First-branch.



Obr. č. 6



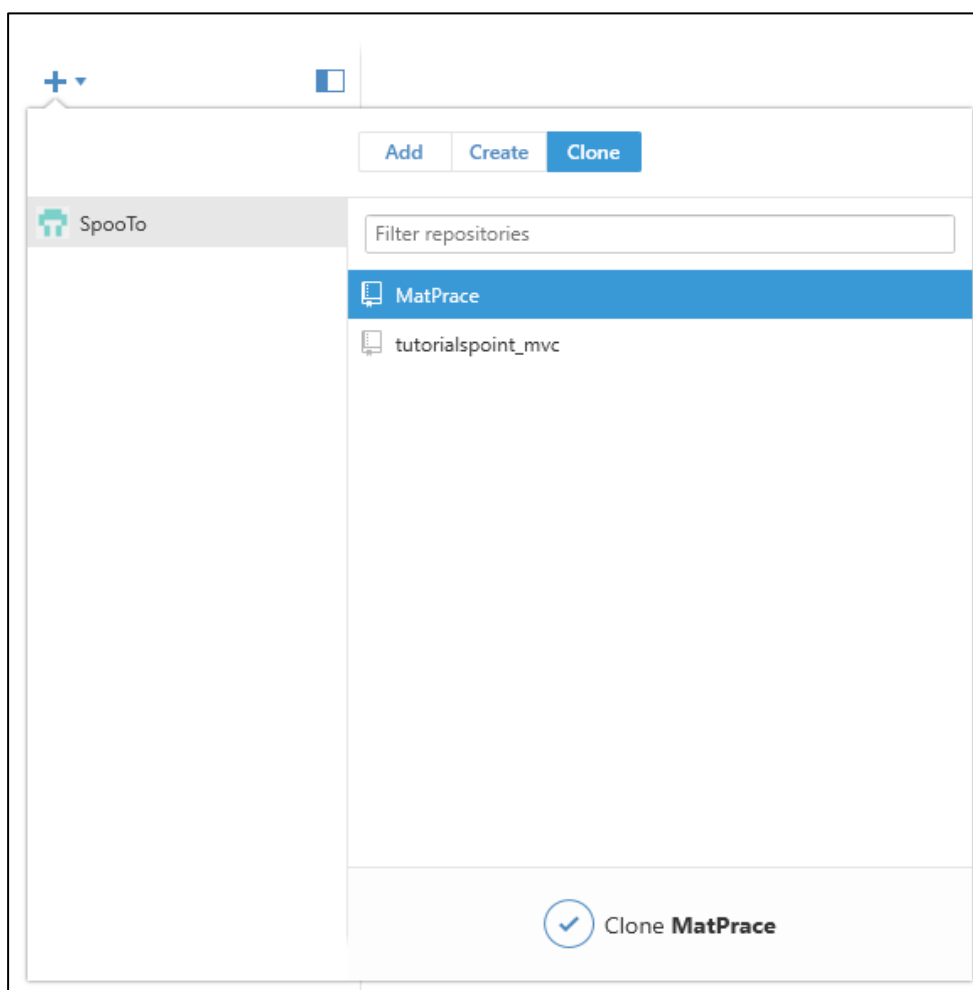
Obr. č. 7



Obr. č. 8

### 3.6. Klonování

Klonování je stahování repozitáře ze serveru na lokální úložiště. Po zvolení repozitáře ke klonování je ještě potřeba vybrat místo k uložení.

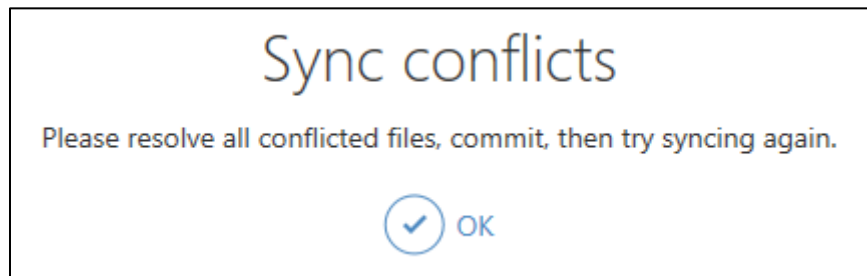


Obr. č. 9

### 3.7. Řešení konfliktů

Ve svém souboru jsem vytvořil textový dokument, který jsem následně pushnul na Git. Celý repozitář jsem na svém notebooku naklonoval a v textovém souboru jsem celý druhý řádek změnil z řady 2 na řadu A. Na svém počítači jsem ale tyto změny nepullnul a tím pádem pracuji na staré verzi. V té jsem druhý řádek vyměnil za řadu B. Změnu jsem následně commitnul

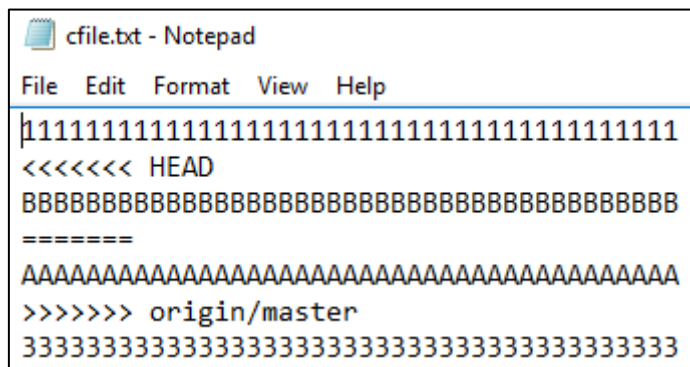
a pushnul. GitHub ale zjistil, že verze, kterou se snažím pushnout, má konflikt s verzí, kterou má na serveru a vyzval mě, abych tento



Obr. č. 10

konflikt vyřešil. Soubor upravil a označil místa kde se konflikt nachází. Pod HEAD je moje verze a od ní je řádkem rovnítek oddělena verze, která je na Gitu. V mém případě je řešení jednoduché. Jedná

se pouze o pár znaků. Může se ale stát, že půjde o několik desítek (i více) řádků kódu. V tom případě řešení není tak jednoduché. Proto je velice důležité mít dobrou organizaci týmu. Je potřebné dohodnout se, kdo bude na čem a kdy pracovat, aby ke konfliktům



Obr. č. 11

nedocházelo, nebo se alespoň co nejvíce omezili.

Když je vyřešena organizace týmu a logistika budoucího kódu může se začít se samotnou webovou aplikací.

Ta je napsaná ve Visual Studio 2015 na ASP.NET frameworku na základě MVC modelu v jazyku C#.

## 4. C# a OOP

C# (také C Sharp) je objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework. C# je založený na jazycích C++ a Java, je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi.

C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení, her atd...

## 5. ASP.NET

ASP.NET je součástí .NET Frameworku pro tvorbu webových aplikací a služeb.

ASP.NET MVC je další oficiální framework postavený na technologii ASP.NET. Tento framework umožňuje snadněji vyvíjet aplikace podle architektury Model-View-Controller.

## 6. MVC

Model-view-controller (MVC) je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má jen minimální vliv na ostatní.

### Princip MVC

Model (model) je doménově specifická reprezentace informací, s nimiž aplikace pracuje.

View (pohled) převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli.

Controller (řadič) reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu.

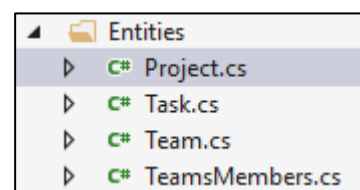
1. Uživatel provede nějakou akci v uživatelském rozhraní (např. stiskne tlačítko).
2. Řadič obdrží oznámení o této akci z objektu uživatelského rozhraní.
3. Řadič přistoupí k modelu a v případě potřeby ho zaktualizuje na základě provedené uživatelské akce (např. zaktualizuje nákupní košík uživatele).
4. Model je pouze jiný název pro doménovou vrstvu. Doménová logika zpracuje změněná data (např. přepočítá celkovou cenu, daně a expediční poplatky pro položky v košíku). Některé aplikace užívají mechanismus pro perzistentní uložení dat (např. databázi). To je však otázka vztahu mezi doménovou a datovou vrstvou, která není architekturou MVC pokryta.
5. Komponenta view použije zaktualizovaný model pro zobrazení zaktualizovaných dat uživateli (např. vypíše obsah košíku). Komponenta view získává data přímo z modelu, zatímco model nepotřebuje žádné informace o komponentě view (je na ní nezávislý). Nicméně je možné použít návrhový vzor pozorovatel, umožňující modelu informovat jakoukoliv komponentu o případných změnách dat. V tom případě se komponenta view zaregistruje u modelu jako příjemce těchto informací. Je důležité podotknout, že řadič nepředává doménové objekty (model) komponentě view, nicméně jí může poslat příkaz, aby svůj obsah podle modelu zaktualizovala.
6. Samotnému konečnému zobrazení výsledku uživateli ještě může u web-aplikací předcházet odpověď ze serveru na klienta, aby si ihned vyžádal obnovení stránky (client side redirect, životnost 0, takže okamžitý): Tím je zaručeno, že při obnovení stránky uživatelem (refresh, F5 v prohlížeči) nevyvolá na serveru požadovanou akci opakovaně, ale že se jedná pouze o obnovení pohledu, nyní už bez požadavku na změnu dat (modelu). Účelem je změna URL a dat http requestu, aby poslední v řadě již nebyl "server-side data-affecting" (ovlivňující model), ale pouze "read-only" (pouhé zobrazení). Celý tento client-refresh (změna URL) se děje automaticky a bez povšimnutí uživatelem.
7. Uživatelské rozhraní čeká na další akci uživatele, která celý cyklus zahájí znovu.

## 7. Webová aplikace

Webová aplikace, kterou jsem napsal pro demonstraci funkce má za úkol pracovat jako task manager – organizátor úkolů. Můžete se do něj zaregistrovat, vytvářet týmy. Přidat se do týmu, nebo z něj odejít. Vytvářet týmům projekty. A v projektech vytvářet úkoly které je potřeba splnit. Nikomu nepřidělené úkoly můžete převzít. Týmy, projekty i úkoly je možné upravovat a smazat.

### 7.1. Model (databáze)

Databáze aplikace je definovaná code-first způsobem. To znamená, že architektura databáze vychází z kódu. V aplikaci jsou kromě entit, které jsou vytvořeny podle šablony, definovány 4 další entity.



Obr. č. 12

Příklad definice entity Projekt:

```
public class Project
{
    public int id { get; set; }
    [Required, MaxLength(50)]
    public string name { get; set; }
    [Required, MaxLength(50)]
    public string description { get; set; }
    public virtual List<Task> tasks { get; set; }
    public int? TeamId { get; set; }
    public virtual Team Team { get; set; }

    public Project()
    {
        tasks = new List<Task>();
    }
}
```

Každá vlastnost třídy Project definuje vlastnost(sloupec) entity v tabulce databáze. Pomocí `System.ComponentModel.DataAnnotations` můžeme zadávat pravidla pro data v databázi. Například `[Required, MaxLength(50)]` bude při vytváření nového záznamu do databáze požadovat aby daná vlastnost byla vyplněná (required) a nebyla delší než 50 znaků (`MaxLength(50)`). Použití vlastnosti `List<Task> tasks` vytvoří 1:n vazbu mezi Project a Task. Kde 1 = Project a n = Task.

Jak tuto definici Visual Studio interpretovalo a podle ní vytvořilo tabulku v databázi si v Server Exploreru můžeme prohlédnout.

|  | Name        | Data Type    | Allow Nulls                         | Default |  |
|--|-------------|--------------|-------------------------------------|---------|--|
|  | id          | int          | <input type="checkbox"/>            |         |  |
|  | name        | nvarchar(50) | <input type="checkbox"/>            |         |  |
|  | description | nvarchar(50) | <input type="checkbox"/>            |         |  |
|  | TeamId      | int          | <input checked="" type="checkbox"/> |         |  |
|  |             |              | <input type="checkbox"/>            |         |  |

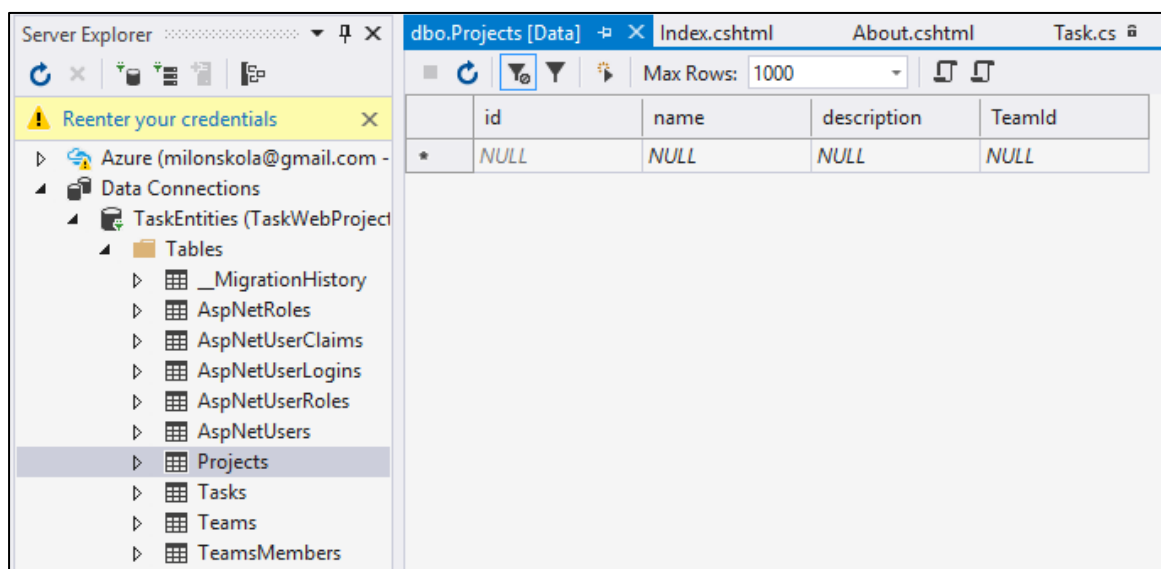
Design

T-SQL

```
CREATE TABLE [dbo].[Projects] (  
    [id] INT IDENTITY (1, 1) NOT NULL,  
    [name] NVARCHAR (50) NOT NULL,  
    [description] NVARCHAR (50) NOT NULL,  
    [TeamId] INT NULL,  
    CONSTRAINT [PK_dbo.Projects] PRIMARY KEY CLUSTERED ([id] ASC),  
    CONSTRAINT [FK_dbo.Projects_dbo.Teams_Team_id] FOREIGN KEY ([TeamId]) REFERENCES [dbo].[Teams] ([id])  
);  
  
GO  
CREATE NONCLUSTERED INDEX [IX_TeamId]  
    ON [dbo].[Projects]([TeamId] ASC);
```

Obr. č. 13

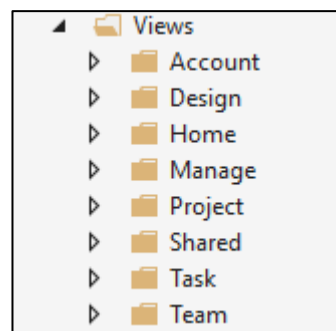




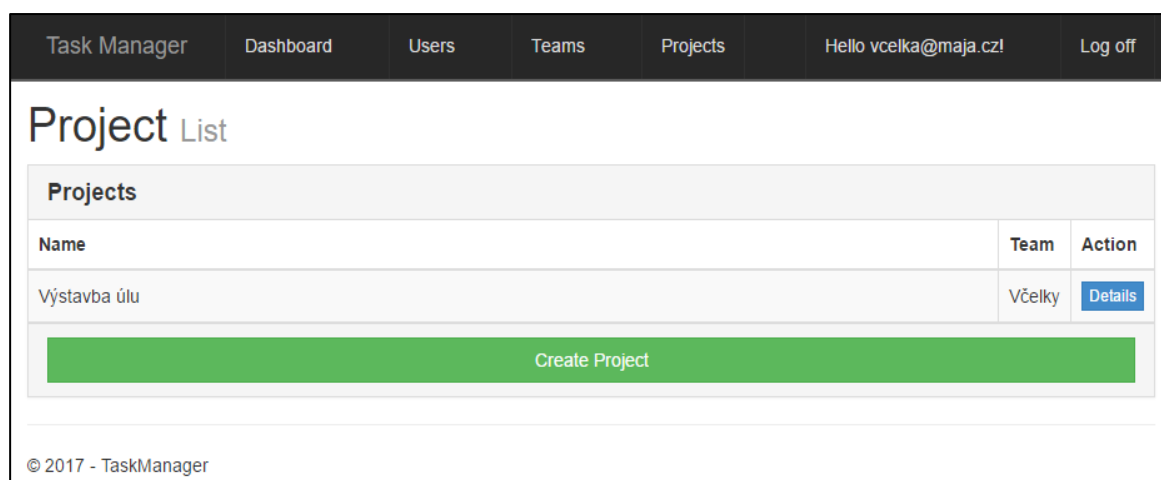
Obr. č. 14

## 7.2. View (pohled)

Složka Views obsahuje html soubory, které se uživatelům zobrazují. Při psaní těchto pohledů je velmi užitečný bootstrap. Ten zajistí kompatibilitu webové aplikace s různými velikostmi browserových oken anebo jinými zařízeními (tablet, smartphone). Pro vysvětlení funkce použijí Views\Project\List.cshtml. Tento View zobrazuje tabulku se všemi projekty, týmy, do kterých projekty patří, tlačítko odkazující na detail projektu a tlačítko pro vytvoření nového projektu.



Obr. č. 15



Obr. č. 16

V hlavičce tohoto souboru jsou definovány data převzaté od controlleru.

```
@model Project
@{
    ViewBag.Title = "Project";
    ViewBag.Subtitle = "List";
    DbSet<Team> teams = ViewBag.teams;
    DbSet<Project> projects = ViewBag.projects;
    SelectList tlist = new SelectList(teams, "id", "title");
}
```

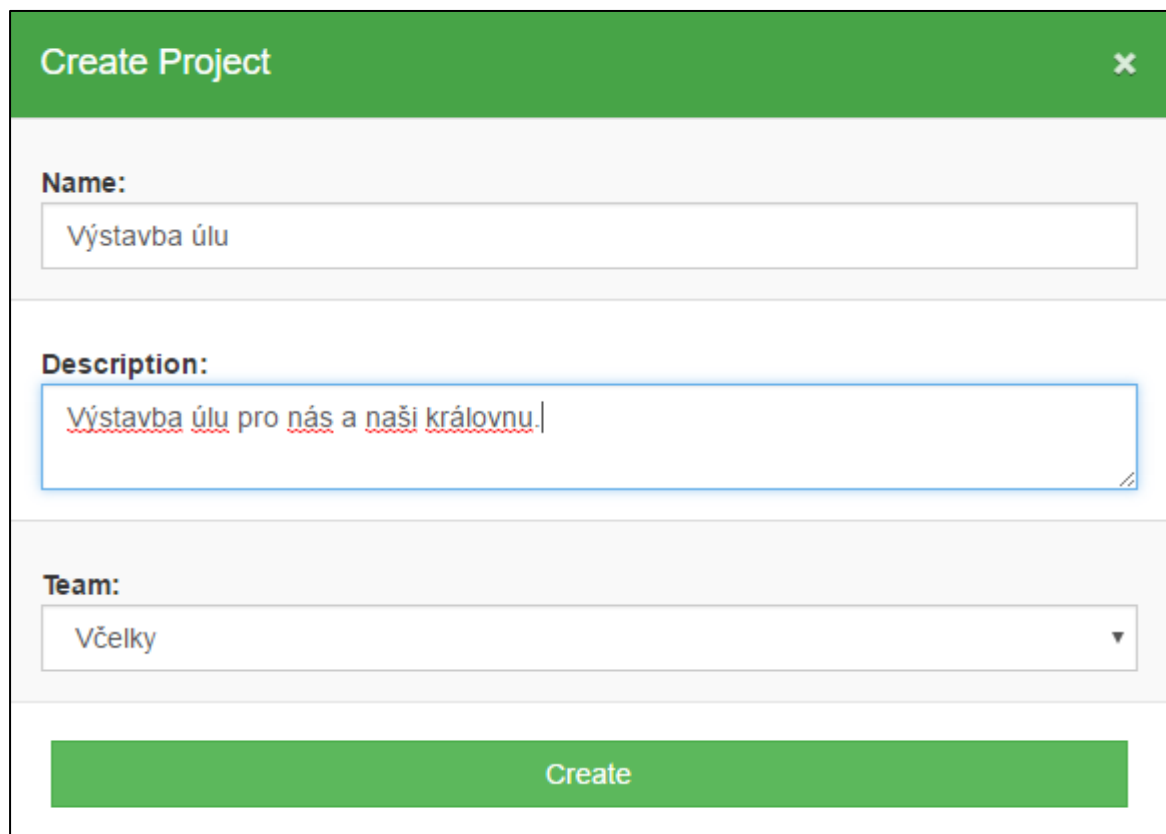
Ty jsou pak použity k vykreslení tabulky se všemi projekty.

```
<div class="panel panel-default">
    <div class="panel-heading">
        <h2 class="panel-title">Projects</h2>
    </div>
    <div class="panel-body panel-table-inserted table-responsive">
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th class="tig-12">Name</th>
                    <th>Team</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody>
                @foreach (Project p in projects)
                {
                    <tr class="middle">
                        <td>@p.name</td>
                        <td>@ProjectManager.GetMainTeam(p).title</td>
                        <td>
                            @Html.ActionLink("Details", "Details", new {
                                id = p.id }, new { @class = "btn btn-primary btn-xs" })
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
    <div class="panel-footer TeamList">
        <button type="button" class="btn btn-success btn-block" data-
toggle="modal" data-target="#Create">Create Project</button>
    </div>
</div>
```

Pomocí @ jsou zde označeny části, které používají C# kód.

Pod tlačítkem Create Project je skrytý formulář pro vytváření nového projektu.

```
<div class="panel-footer TeamList">  
    <button type="button" class="btn btn-success btn-block" data-  
toggle="modal" data-target="#Create">Create Project</button>  
</div>
```



The image shows a modal window titled "Create Project" with a green header bar and a close button (X) in the top right corner. The form is divided into three sections, each with a label and a corresponding input field:

- Name:** A text input field containing the text "Výstavba úlu".
- Description:** A text area containing the text "Výstavba úlu pro nás a naši královnu." with a blue border and a small icon in the bottom right corner.
- Team:** A dropdown menu with the text "Včelky" and a downward arrow.

At the bottom of the modal is a large green button labeled "Create".

Obr. č. 17

```

@using (Html.BeginForm("Create", "Project", null, FormMethod.Post, new {
@class = "form-horizontal form-stripped" }))
{
    <div class="modal fade modal-success" id="Create" role="dialog">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <button type="button" class="close" data-
dismiss="modal">&times;</button>
                    <h4 class="modal-title">Create Project</h4>
                </div>
                <div class="modal-body modal-table-inserted table-
responsive">
                    <div class="form-group">
                        @Html.LabelFor(m => m.name, "Name:", new {
@class = "control-label col-md-2" })
                        <div class="col-md-10">
                            @Html.TextBoxFor(m => m.name, new { @class =
"form-control" })
                        </div>
                    </div>
                    <div class="form-group">
                        @Html.LabelFor(m => m.description,
"Description:", new { @class = "control-label col-md-2" })
                        <div class="col-md-10">
                            @Html.TextAreaFor(m => m.description, new {
@class = "form-control" })
                        </div>
                    </div>
                    <div class="form-group">
                        @Html.Label("team", "Team:", new { @class =
"control-label col-md-2" })
                        <div class="col-md-10">
                            @Html.DropDownListFor(m => m.TeamId, tlist,
new { @class = "form-control" })
                        </div>
                    </div>
                </div>
                <div class="modal-footer">
                    <input type="submit" class="btn btn-success btn-
block" value="Create" />
                </div>
            </div>
        </div>
    </div>
}

```

Tento formulář se po stisknutí tlačítka Create odešle příslušnému controlleru, který podle něj vytvoří nový projekt a odkáže vás na stránku s detailem tohoto projektu.

### 7.3. Controller (řadič)

Složka Controllers obsahuje controllery starající se o jednotlivé viewy. Každý controller obsahuje několik metod typu `System.Web.Mvc.ActionResult`, které pak uživatel přes view používá k manipulaci s daty. Těmto metodám je možné nastavit `[HttpPost]` nebo `[HttpGet]` vlastnost, která určí, že metoda bude volána pouze při `HttpPost` nebo `HttpGet` dotazech. Pro demonstraci použiju `TeamController`. Ten obsahuje metodu `List`.

```
public ActionResult List()  
{  
    Team team = new Team();  
    ViewBag.teams = te.teams;  
    return View(team);  
}
```

Ta vloží do `ViewBagu` list všech týmů v databázi a otevře view `Team/List`. V tomto view uvidíte tabulku se všemi týmy, tlačítko pro otevření jejich detailu a tlačítko pro vytvoření nového týmu. Po kliknutí na tlačítko k vytvoření týmu se ukáže dialogové okno s formulářem, do kterého je potřeba vyplnit informace o novém týmu a s tlačítkem, které zavolá metodu controlleru a pošle data z formuláře.

Příklad:

Vytvoření formuláře, který bude volat metodu “Create” v controlleru “Team”.

```
@using (Html.BeginForm("Create", "Team", null, FormMethod.Post, new {  
    @class = "form-horizontal form-striped" }))
```

Když metoda `List` vytvářela view, vytvořila také nový prázdný tým, který view předala jako “model”.

```
@model Team
```

Když vyplňujete formulář, vyplňujete vlastnosti tohoto týmu.

```
<div class="form-group">  
    @Html.LabelFor(m => m.title, "Name:", new { @class = "control-  
label col-md-2" })  
    <div class="col-md-10">  
        @Html.TextBoxFor(m => m.title, new { @class = "form-control"  
    })  
    </div>  
</div>
```

Kde m je právě nahoře definovaný model.

```
public ActionResult Create(Team team)
{
    int id = tm.Create(team);

    return RedirectToAction("Details", new { id = id });
}
```

Metoda Create tento tým převeze a zavolá metodu tm.Create(team).

```
public override int Create(Team entity)
{
    db.teams.Add(entity);
    db.SaveChanges();
    int id = entity.id;
    return id;
}
```

Ta převeze tým. Vytvoří novou entitu v databázi, uloží změny v databázi, do int id vloží id této nové entity a následně toto id vrátí. Controller toto id převeze, a pomocí RedirectToAction zavolá metodu Details, které předá id.

```
public ActionResult Details(int id)
{
    Team t = tm.Read(id);
    ViewBag.isMember = tm.IsMember(t,
User.Identity.GetUserId());
    ViewBag.AppUsers = new ApplicationDbContext().Users;
    return View(t);
}
```

Ta zase převeze id a zavolá metodu tm.Read(id).

```
public override Team Read(int id)
{
    return db.teams.SingleOrDefault(t => t.id == id);
}
```

Ta podle id najde v databázi tým a vrátí ho. Dále metoda Details vytvoří ViewBag.isMember, do kterého vloží bool na základě toho, jestli je právě přihlášený uživatel členem tohoto týmu nebo ne. Tuto informaci získá zavoláním metody tm.IsMember(t, User.Identity.GetUserId()), kde t = tým pro který to chce zjistit a User.Identity.GetUserId() = metoda, která vrací právě přihlášeného uživatele.

```
public bool IsMember (Team t, string id)
{
    TeamsMembers tm = db.teamsMembers.SingleOrDefault(x =>
(x.userId == id) && (x.team.id == t.id));
    bool answer = false;
    if(tm!=null)
    {
        answer = true;
    }
    return answer;
}
```

Metoda `IsMember` pak vyzkouší vyhledat v databázi v tabulce `TeamMembers` záznam pro tento tým a uživatele. Když záznam nenajde vrátí `false`, když ho ale najde vrátí `true`. Pomocí tohoto boolu view pak umožní uživateli do týmu vstoupit anebo z něj odejít.

Metoda `Details` dále vytvoří `ViewBag.AppUsers`, který obsahuje list všech členů týmu.

Nakonec metoda vrací view s modelem, který obsahuje dříve nalezený tým.

The screenshot shows a web application interface with a dark navigation bar at the top containing links: Task Manager, Dashboard, Users, Teams, Projects, Hello vcelka@maja.cz!, and Log off. The main content area is titled 'Team Včelky'. Below the title, there is a section for the team 'Včelky' with an 'Edit' button and a 'Delete' button. The description of the team is 'Vyrábíme medík.'. Below this, there are two side-by-side sections: 'Members' and 'Projects'. The 'Members' section has a table with columns 'Name' and 'Action'. It lists 'vcelka@maja.cz' with a 'Details' button. Below the table is a green button labeled 'Leave'. The 'Projects' section has a table with columns 'Name' and 'Action'. It lists 'Výstavba úlu' with a 'Details' button. Below the table is a green button labeled 'Create Project'. At the bottom of the page, there is a copyright notice: '© 2017 - TaskManager'.

Obr. č. 18

## Zdroje

Wikipedie. [online]. [cit. 2017-03-28]. Dostupné z:  
[https://cs.wikipedia.org/wiki/ASP.NET#ASP.NET\\_MVC](https://cs.wikipedia.org/wiki/ASP.NET#ASP.NET_MVC)

NuGet Gallery | Microsoft ASP.NET MVC 5.2.3. NuGet Gallery | Home. [online].  
Copyright © 2017 .NET Foundation [cit. 28.03.2017]. Dostupné z:  
<http://www.nuget.org/packages/Microsoft.AspNet.Mvc>

The world's leading software development platform · GitHub. [online]. Copyright © 2017  
[cit. 28.03.2017]. Dostupné z: <https://github.com/>

GitHub – Wikipedie. [online]. Dostupné z: <https://cs.wikipedia.org/wiki/GitHub>

The Linux Kernel Archives. [online]. Dostupné z:  
<https://www.kernel.org/pub/software/scm/git/docs/user-manual.html#ensuring-reliability>

Git – Wikipedie. [online]. Dostupné z: <https://cs.wikipedia.org/wiki/Git>

Webová aplikace – Wikipedie. [online]. Dostupné z:  
[https://cs.wikipedia.org/wiki/Webov%C3%A1\\_aplikace](https://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace)

Microsoft Visual Studio – Wikipedie. [online]. Dostupné z:  
[https://cs.wikipedia.org/wiki/Microsoft\\_Visual\\_Studio](https://cs.wikipedia.org/wiki/Microsoft_Visual_Studio)

Jakubzapletal. [online]. Dostupné z: <http://jakubzapletal.com/zaciname-s-githubem/>

GitHub. [online]. Dostupné z: <https://guides.github.com/activities/hello-world/>

C Sharp – Wikipedie. [online]. Dostupné z: [https://cs.wikipedia.org/wiki/C\\_Sharp](https://cs.wikipedia.org/wiki/C_Sharp)

Model-view-controller – Wikipedie. [online]. Dostupné z:  
<https://cs.wikipedia.org/wiki/Model-view-controller>





VYŠŠÍ ODBORNÁ ŠKOLA, STŘEDNÍ PRŮMYSLOVÁ ŠKOLA  
A JAZYKOVÁ ŠKOLA S PRÁVEM STÁTNÍ JAZYKOVÉ ZKOUŠKY

## KONZULTAČNÍ LIST

Příjmení a jméno žáka: Ondráš Milan

Třída: E4C

Téma maturitní práce: Vývoj aplikací s využitím architektury MVC

Datum kontroly: \_\_\_\_\_

Obsah kontroly:

Hodnocení kontroly: 1 2 3 4 5

Podpis vedoucího práce: \_\_\_\_\_

Datum kontroly: \_\_\_\_\_

Obsah kontroly:

Hodnocení kontroly: 1 2 3 4 5

Podpis vedoucího práce: \_\_\_\_\_

Datum kontroly: \_\_\_\_\_

Obsah kontroly:

Podpis vedoucího práce: \_\_\_\_\_