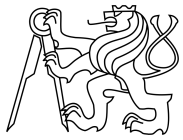
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová aplikace pro kreativní výuku fyziky
Student:	Přemysl Černý
Vedoucí:	Ing. David Buchtela, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je návrh a implementace webové aplikace využitelné pro kreativní výuku fyziky na základních a středních školách tak, aby byla pro žáky snadno pochopitelná.

1. Zanalyzujte uživatelské požadavky (učitelů fyziky ZŠ a SŠ) s ohledem na technické potřeby aplikace a uživatelsky příjemné a pochopitelné ovládání.
2. Proveďte návrh aplikace v souladu s metodami softwarového inženýrství.
3. Implementujte pilotní aplikaci tak, aby splňovala požadavky na vstup a výstup dat používaný ve výuce fyziky ZŠ a SŠ (vstupní data z měřících přístrojů Vernier a výstupy ve formátu CSV a v podobě grafů).
4. Webovou aplikaci otestujte ve zvolené ZŠ nebo SŠ (po dohodě s vedoucím práce).
5. Proveďte analýzu finanční náročnosti pořízení a provozu aplikace a očekávaných benefitů s ohledem na omezené finanční prostředky ZŠ a SŠ.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 11. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Webová aplikace pro kreativní výuku fyziky

Přemysl Černý

Vedoucí práce: Ing. David Buchtela, Ph.D.

8. května 2016

Poděkování

Děkuji panu Mgr. Jaroslavu Reichlovi za pomoc při sběru uživatelských požadavků a při testování aplikace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2016 .....

Odkaz na tuto práci

Černý, Přemysl. *Webová aplikace pro kreativní výuku fyziky.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá implementací webové aplikace, která umožní učitelům fyziky na základních a středních školách zaznamenávat data z měřících přístrojů Vernier a odesílat je na server, kde budou k dispozici v podobě grafů a ve formátu CSV ke stažení. Hlavní důraz je kladen na to, aby aplikace byla srozumitelná a umožnila tak kreativní výuku fyziky, popřípadě zadávání domácích úkolů či laboratorních prací, ve kterých je důležité zaznamenávání fyzikálních veličin a jejich následná analýza, popřípadě porovnání s ostatními veličinami či výstupy z různých měřících přístrojů. Při vytváření řešení je důležité zohlednit skutečnost, že školy disponují omezenými finančními prostředky, takže minimalizace nákladů je hlavní prioritou. K vlastní programové implementaci jsou tedy využity volně dostupné nástroje. Společnost Vernier přímo poskytuje základní program spolu se zdrojovými kódy, které lze dále rozšiřovat a zefektivnit tak celý proces tvorby aplikace na klientské straně. Na straně serveru se pak jedná o jednoduché komunikační rozhraní a zobrazování dat, k čemuž jsem využil Nette framework, jenž je rovněž dostupný zdarma. Konečným cílem bylo aplikaci vhodně navrhnout, implementovat a vyzkoušet na zvolené střední škole, což se nakonec s drobnými nepřesnostmi zmíněnými v závěru povedlo.

Klíčová slova kreativní výuka fyziky, aplikace pro měření, sběr naměřených hodnot, analýza požadavků fyzikářů, zpracování naměřených dat

Abstract

This bachelor thesis deals with implementation of a web application, which would enable teachers of physics in primary and high schools to record data from Vernier measuring devices and send them to the server, where they would be available in the form of graphs and CSV format for downloading. The main emphasis is on the fact that the application should be understandable and thus enable creative education of physics, and also help create homework or laboratory works, where it is important to record physical quantities and perform their subsequent analysis, or comparison with other variables or outcomes from different measuring devices. When creating the solution, it is necessary to take into account the fact that schools have limited financial resources, so minimizing costs is a top priority. For the implementation of the application, freely available tools are therefore used. Vernier company directly provides essential software along with its source code, which can be further expanded and thus the development process of the application on the client side can be made more efficient. On the server side, there is used a simple communication interface and data display, for which I made use of the Nette framework, which is also available for free. The ultimate goal of this thesis is to properly design, implement and test the application in a chosen high school, which with minor inaccuracies mentioned in the conclusion eventually came to a successful end.

Keywords creative education of physics, application for measuring, collecting measured values, physics teachers' requirements analysis, processing of measured data

Obsah

Seznam obrázků

Seznam tabulek

Seznam úryvků kódu

Úvod

V dnešních dobách, kdy dochází k výraznému pokroku v oblasti informačních technologií, se někteří učitelé, firmy a různé zájmové skupiny snaží přijít na to, jak tento pokrok využít ke kreativní výuce na školách, která by zahrnovala aktivní zapojení žáků do problematiky daného předmětu. Nejvíce je tato tendence vidět v předmětech, které souvisejí s přírodními vědami, neboť lze žáky zároveň naučit vědeckým postupům, jež různé podpůrné softwarové programy v dnešních dobách běžně využívají. Vedle prohloubení zájmu žáků v těchto oblastech se tak mohou všichni naučit využívat veřejně dostupné prostředky pro občanskou vědu (angl. citizen science), která díky nejrůznějším zařízením včetně počítačů a chytrých telefonů zažívá celosvětový rozvoj [1]. Avšak i přes všechny pozitivní a neprozkoumané možnosti, které kreativní výuka s využitím informačních systémů nabízí, se ve školském prostředí potýkáme se zásadními problémy. Jedním a největším z nich je značný nedostatek finančních prostředků, což je navíc doplněno neochotou většiny učitelů se problematice kreativní výuky věnovat ve svém volném čase. Vlastní zapojení informačních systému do výuky tkví především v možnosti shromažďování a zpracování zaznamenaných dat, která souvisí s vědeckým pozorováním. Tato data lze využít nejen učiteli k názorné ilustraci fungování přírody, ale případně také profesionálními vědci pro vlastní vědecké účely, kterými myslím například zaznamenávání výskytu druhů tažných ptáků na různých místech v různém období v roce. Tato data jsou tedy obecně cenná nejen pro žáky, ale i pro vědce, kteří je mohou využít k monitorování přírodních jevů.

Kromě dodatečného využití sesbíraných dat se ale na školách potýkáme s aktuálním problémem, kterým je časová neefektivita při laboratorních pracích. Žáci běžně musí naměřenou fyzikální veličinu ručně zaznamenat do tabulky, načež pouze tehdy je možné ze záznamů vytvořit přehledný graf. Samotné zaznamenávání zabírá velikou část časového rozsahu laboratorní práce a odvádí tak žáky od hlavního účelu, který by si měli z výuky odnést, a tím je skutečnost, jak daný jev v přírodě funguje a jak k němu můžeme přistoupit z vědeckého hlediska. Kreativní výuka s možností využití počítačů a k nim při-

pojitelných měřících zařízení tento problém již řeší, avšak většina řešení stále umožňuje pouze lokální práci, takže žáci jsou nuceni data ručně (tj. e-mailem nebo pomocí datových nosičů) předávat učiteli k ohodnocení.

Účelem této práce je podpořit kreativní výuku fyziky analýzou, návrhem a implementací jednoduché snadno pochopitelné aplikace pro zaznamenávání naměřených fyzikálních dat na serveru, kde bude docházet k jejich následnému zpracování. Po celou dobu bude brán ohled na omezené finanční prostředky základních a středních škol. K pochopení této práce jsou nutné alespoň minimální znalosti v oblasti vývoje a fungování softwaru, fyziky a školského prostředí.

Kapitola 1

Cíl práce

Cílem této práce je návrh a implementace pro žáky snadno pochopitelné webové aplikace, která by podpořila kreativní výuku fyziky na základních a středních školách. Tato webová aplikace bude podporována klientským programem, pomocí nějž se budou sbírat data z různých měřících přístrojů od firmy Vernier a odesílat do webové aplikace na serveru k dalšímu zpracování. Záměrem je tedy sesbírat uživatelské požadavky učitelů fyziky s ohledem na technické potřeby aplikace, provést jejich analýzu, návrh implementace a vlastní implementaci pilotní verze aplikace tak, aby splňovala požadavky na výstup dat používaný ve výuce fyziky na středních školách (především výstup ve stažitelném formátu CSV). Během celého postupu je potřeba brát ohled na omezené finanční prostředky základních a středních škol. Aplikace bude následně otestována na dvou praktických příkladech a na vybrané střední škole spolu s žáky a učiteli k získání zpětné vazby.

Kapitola 2

Teoretický úvod

2.1 Kreativní výuka fyziky

V dnešních dobách prudkého technologického rozvoje vzniká spousta nových nápadů, jak výsledky tohoto pokroku uplatnit v nejrůznějších lidských činnostech, počínaje zemědělstvím a konče nejnovějšími vědeckými potřebami zkoumat důkladněji a přesněji svět kolem nás [1]. Tento rozvoj se samozřejmě nevyhnul pozornosti ani školských zařízení, která i přes dosavadní zavedenou tradici již začala vyhledávat možná softwarová a hardwarová řešení pro zkvalitnění výuky. Jako příklad uveďme umístění interaktivních tabulí do učeben, které nabízí možnost žákům názorně probíranou látku ukázat a vysvětlit, a to především pomocí prezentací, videí a v případě přírodních věd i grafických znázornění zkoumaných veličin. Avšak ti učitelé, kteří jsou technologickému rozvoji velmi pozitivně nakloněni, by rádi šli ještě dále, jelikož v dnešních dobách téměř každá rodina vlastní moderní počítače, tablety a chytré telefony, které by mohly poskytnout žákům zábavnou a interaktivní formu vzdělávání a podpořit tak význam myšlenky Jana Ámose Komenského, jež je běžně vyjadřována slovy „škola hrou“. Této tendence si začaly všímat i obchodní společnosti (mezi nimi i firma Vernier), které učitelům nabízí nejrůznější přístroje, měřící zařízení a softwarové programy pro zkvalitnění výuky, aby učitelé mohli myšlenku „škola hrou“ snáze aplikovat. Zmíněné společnosti využívají služeb různých prostředníků, aby mohly své technologie ve školách ukázat a případně v nich učitele vyškolit. Jelikož se jedná o zcela novou věc, je nanejvýš důležité, aby se všechny přístroje a programy daly snadno obsluhovat.

Právě díky mému bývalému učiteli fyziky na střední škole jsem měl možnost do této tématiky nahlédnout a domluvit způsob, jak by se daly přístroje pro měření fyzikálních veličin využít při výuce fyziky k zadávání domácích úkolů, popřípadě jako pomůcka při laboratorních pracích ve školách. Jelikož žáků je v jedné třídě více, bylo potřeba vymyslet způsob, jak jednotlivá měření ať už jednotlivců nebo malých skupinek porovnat bez nutnosti přenášet data pomocí flash disku a bez jejich složitého nahrávání do programů, které

umožňují vytvářet grafy (například Microsoft Excel). Z vlastní zkušenosti během klasických laboratorních prací mohu říci, že zadávání dat do tabulek, pojmenovávání veličin a tvorba grafů vždy zabrala nejvíce času a odsunula tak hlavní význam laboratorní práce – vlastní analýzu naměřených údajů a jejich porovnání, což má žákům ukázat, jak se daný fyzikální jev projevuje ve světě kolem nás.

2.2 Možná řešení

I když je pravdou, že jednotlivé společnosti vyrábějící zařízení pro měření dat již poskytují základní sadu programů pro tvorbu grafů, a částečně tak řeší výše zmíněný problém se zbytečně stráveným časem, data nikde neshromažďují a neumožňují tak jejich porovnání. Žáci tak musí data odeslat učiteli (například pomocí e-mailu), který je až následně poté může vyhodnotit. Školská zařízení disponují omezenými finančními prostředky a nemohou si tak dovolit využívání drahých komerčních nástrojů, popřípadě vytvoření softwarového řešení na zakázku – kromě toho je zřejmé, že samotná kreativní výuka nepředstavuje zas takové přínosy pro žáky ani školy samotné, aby se jí na plný úvazek věnoval byť jen jediný zaměstnanec.

Jelikož se jedná o novou věc využívající dosud ve školách neprověřené technologie, všechna řešení jsou založena na názorném vyzkoušení aplikací přímo ve školách. Z tohoto důvodu není důležitý ani tak výběr platformy a programovacího jazyka, ale spíše jednoduchost na pochopení a přívětivost aplikace pro žáky. Pro shromažďování dat je důležité jejich centralizované ukládání, které může být ve školském prostředí v zásadě zajištěno dvěma způsoby:

1. S využitím vnějšího serveru, k němuž se bude přistupovat pomocí internetového připojení.

2. S využitím serveru ve vnitřní síťové struktuře školy.

Druhý způsob je problematický v tom ohledu, že v základní podobě neumožňuje zadávání domácích úkolů, protože se žáci k serveru nebudou mít ze svého domova jak dostat. Na druhou stranu nabízí mnohem lepší kontrolu ze strany školy samotné, která může v případě problémů server ihned opravit, popřípadě jeho činnost zastavit. Je zřejmé, že serverová část aplikace bude potřebovat i svůj klientský protějšek, který umožní žákům data na server automaticky odesílat. Klientská část bude mít na starost shromažďování dat z jednotlivých zařízení, která budou připojená k danému počítači, jejich kontrolu (především kontrolu jednotky fyzikální veličiny) a odesílání tak, aby bylo zřejmé, od koho data přicházejí a pod jakým názvem/projektem se mají na serveru ukládat.

Tato práce má za cíl zaměřit se na tvorbu aplikace s využitím zařízení pro měření od firmy Vernier, která sama nabízí základní sadu řešení, na kterých lze aplikaci stavět [2]. Bylo by jisté možné alternativně využít pouze

základní knihovny pro ovládání měřících zařízení a celou aplikaci napsat od počátku, nicméně ve školském prostředí, které nelpí na jedinečnosti softwarového návrhu, by se jednalo o zbytečné snažení. Vernier nabízí sadu základních programů pro operační systémy Windows, Mac a Linux s různým stupněm funkčnosti. Použité programovací jazyky zahrnutí především C++, C# a Visual Basic, takže si programátor může sám zvolit, v jakém prostředí se mu aplikace nejlépe programuje.

2.3 Teoretický postup řešení

Vzhledem k tomu, že prvotní funkční požadavky na aplikaci (viz níže) měly spíše abstraktní charakter a vznikaly na základě průběžných schůzek a návrhů, rozhodl jsem se využít agilní způsob [3] vývoje aplikace, kde jsem začal rozšiřováním stávajícího programu od firmy Vernier a přidáváním dalších funkcí podle toho, jak byly požadavky vznášeny. Tomu jsem pak přizpůsoboval i vývoj serverové části aplikace. Následující text se snaží vše shrnout dle tématu dané kapitoly, které vždy odpovídá jedné fázi v životním cyklu vývoje softwaru. Tato témata zahrnují:

- sběr a analýzu požadavků
- návrh řešení
- implementaci serverové i klientské aplikace
- testování a vyvození výsledků

Kapitola 3

Sběr a analýza požadavků

Samotnou analýzu jsem začal zachycením uživatelských požadavků [4], které jsem ale z důvodu nadbytečné práce dále nerozepisoval na případy užití [5]. Pro tento účel jsem si domluvil několik schůzek s panem magistrem Jaroslavem Reichelem (dále jen „učitel“), který vyučuje fyziku na Střední průmyslové škole sdělovací techniky v Panské ulici v Praze a jenž celou záležitost s vývojem webové aplikace pro kreativní výuku fyziky předtím detailně konzultoval s panem magistrem Pavlem Böhmem z Univerzity Karlovy, který se zabývá právě čidly Vernier a jejich nasazováním do výuky. Zpočátku byly požadavky velmi obecné, neboť se jedná o zcela novou věc, a nikdo neměl jasnější představu, jak by taková aplikace měla vypadat. Nejdůležitější byly pro učitele následující tři funkční požadavky:

- Aplikace by měla zachytávat naměřená data z různých čidel Vernier a odesílat je na server.
- Data by se měla nechat dát zobrazit v podobě grafů.
- Při překročení krajní hodnoty by se měla na e-mail odeslat chybová hláška.

Poslední požadavek byl odůvodněn tím, že se předpokládá využití aplikace i při výzkumné práci, kdy je důležité například hlídání správné teploty v místnosti, a bylo by tedy žádoucí, aby badatel nemusel u počítače neustále sedět a kontrolovat aktuální hodnotu teploměru. Požadavek jsem přijal jako vedlejší, protože přímo nesouvisí s hlavním využitím aplikace (kreativní výuka). Po rozebrání si jednotlivých požadavků dopodrobna a po uvažování nad možnými scénáři vyvstaly další důležité funkční požadavky:

- Naměřená data by se měla dát ze serveru stáhnout, ideálně ve formátu CSV.
- Každý by měl mít na serveru svůj vlastní uživatelský účet, který by mohl spravovat více projektů (různých měření).

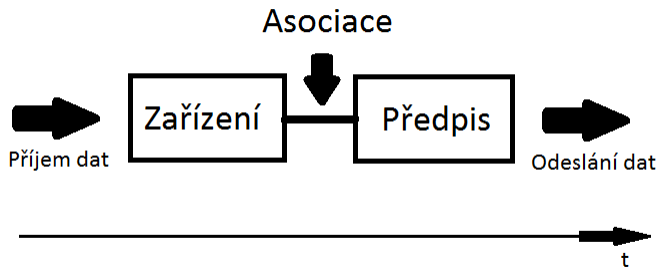
- Graf naměřených hodnot, chybová hláška při překročení krajní hodnoty a aktuální hodnoty ze všech čidel v daném projektu by se měly zobrazovat i na obrazovce v klientské aplikaci.

Se všemi těmito dodatečnými požadavky jsem předem počítal, dokonce jsem navrhl i využití LED diody na jednotlivých přístrojích firmy Vernier. Tato dioda nabízí čtyři stavy (vypnutá, červená, oranžová, zelená), které by mohly sloužit jako identifikátory toho, jestli je dané zařízení aktivní (oranžová barva), jestli je právě aplikací využíváno k měření (zelená barva), a popřípadě jestli došlo k překročení krajní hodnoty (červená barva). Učiteli se tento nápad líbil, avšak jelikož už nás v tuto chvíli nenapadaly žádné další hlavní funkční požadavky, rozhodli jsme se přejít k těm nefunkčním. Co se týče spolehlivosti a bezpečnosti aplikace, učitele nenapadala zatím žádná kritéria, neboť aplikace by podle něj měla být v počátečních fázích velmi jednoduchá pro testování. Dohodli jsme se, že spolehlivost aplikace bude zatím přímo úměrná spolehlivosti serveru (ať už vlastního virtuálního serveru nebo hostingu), na kterém poběží. Hlavní prioritou zatím je mít pouze žádné nebo velmi nízké náklady, proto se učiteli líbil můj nápad využít předpřipravené aplikace, které Vernier sám zdarma poskytuje [2]. Tím nám vznikl požadavek psát klientskou aplikaci v jazyce C++ s využitím frameworku MFC, neboť předpřipravená aplikace od Vernieru je v něm napsaná. Z důvodu jednoduchosti a bezpečnosti jsme pro serverovou aplikaci zvolili český framework Nette pro jazyk PHP. Mezi další hlavní nefunkční požadavky patří:

- Uživatelé se budou v serverové i klientské aplikaci přihlašovat pomocí uživatelského jména a hesla, čímž bude zajištěna základní bezpečnost.
- Veškeré údaje budou uchovávány na serveru včetně jednotky fyzikální veličiny daného zařízení, hraničních hodnot a údajů o uživateli; důvodem je snazší udržitelnost a případná rozšiřitelnost, neboť zdrojový kód v serverové aplikaci lze velmi snadno upravit a nemusí se vydávat nová verze, kterou by si všichni uživatelé museli stáhnout (jako je tomu u klientské aplikace).

Po ujasnění si základních požadavků jsme se společně s učitelem rozhodli je dále analyzovat. Zaměřili jsme se především na možné scénáře použití aplikace. Ze začátku jsme pro jednoduchost předpokládali, že je vše na serveru již připraveno včetně uživatelského účtu a projektu pro zaznamenávání naměřených dat, abychom se mohli napřed zaměřit na klientskou aplikaci, kterou budou žáci přímo obsluhovat. Hlavní scénář, který bude například aplikovaný během laboratorních prací, jsme rozdělili do následujících po sobě jdoucích fází:

1. Uživatel spustí program a zadá své uživatelské jméno a heslo.
2. Program uživateli zobrazí projekty, do kterých se na serveru připojil.



Obrázek 3.1: Předpis zařízení

3. Uživatel zvolí projekt.

4. Program po vybrání projektu uživateli na obrazovce zobrazí názvy jednotlivých k počítači připojených měřících přístrojů a také názvy jednotlivých předpisů zařízení, které jsou na serveru v rámci daného projektu vytvořeny jako zástupci jednotlivých fyzických zařízení.

5. Uživatel přiřadí všechna potřebná měřící zařízení k jednotlivým předpisům.

6. Pokud není na serveru uložená jednotka dané fyzikální veličiny, kterou zařízení měří, program zobrazí dialog s výběrem možných jednotek.

7. Uživatel spustí měření výběrem příslušného příkazu z menu nebo stiskem klávesové zkratky.

8. Program bude průběžně odesílat naměřená data na server.

Během tvorby tohoto scénáře vyvstal zcela nový funkční požadavek. Učitele napadlo, že stejní žáci mohou laboratorní práce provádět vícekrát, buď jako opravy těch nepodařených, domácí dokončení těch, které nestihnou dodělat ve škole, nebo pokud budou laboratorní práce rozdělené do více vyučovacích hodin. V důsledku toho by bylo dobré, aby si program pamatoval, jaké fyzické zařízení bylo naposledy přiřazeno k jakému předpisu (zástupci) v rámci projektu. Jelikož jsem byl s čidly Vernier obeznámen a věděl jsem, že každé zařízení má jedinečný identifikační řetězec, napadlo mě tento identifikátor uchovávat na serveru spolu s jednotkou a ostatními údaji týkajícími se daného zařízení. Jakmile uživatel při dalším měření připojí stejné zařízení k počítači, již automaticky se dané zařízení přiřadí k odpovídajícímu předpisu. Samozřejmě to samé se týká i uložené jednotky měřené fyzikální veličiny, která se nyní automaticky použije pro měření, čímž se ušetří čas při dodělávání laboratorních prací.

Když jsme si s učitelem znovu scénář prošli a shodli se na jeho podobě, rozhodli jsme se nyní zaměřit na klíčovou část serverové aplikace, ke které

budou uživatelé přistupovat pomocí grafického webového rozhraní. Začali jsme opět s tvorbou nejtypičtějšího scénáře:

1. Uživatel se v aplikaci registruje, čímž získá uživatelské jméno a heslo, které bude následně využívat jak serverová aplikace, tak i klientský program.
2. Uživatel se přihlásí do serverové aplikace pomocí jména a hesla.
3. Aplikace zobrazí seznam všech vytvořených projektů pro měření, do kterých má uživatel povolený přístup.
4. Uživatel může vytvořit nový projekt a pojmenovat ho.
5. Uživatel si nechá zobrazit detaily projektu kliknutím na jeho název, který aplikace zobrazí v menu.
6. Uživatel v projektu vytvoří nový předpis zařízení.
7. Uživatel může nově vytvořený předpis zařízení pojmenovat, přiřadit mu typ fyzického přístroje Vernier (pokud ho zná), jednotku (pokud ji zná) a horní a dolní hranici naměřených hodnot, jejichž překročení zobrazí chybovou hlášku v klientské aplikaci.
8. Aplikace uživateli zobrazí graf se všemi naměřenými hodnotami v rámci projektu. Uživatel si může data stáhnout ve formátu CSV.

Samozřejmostí podle učitele bylo, aby uživatel mohl kdykoli své údaje (jméno, heslo, atp.) změnit, odstranit předpis zařízení nebo celý projekt spolu se všemi naměřenými daty.

Po vytvoření těchto nejběžnějších scénářů jsme se společně podívali na základní program od firmy Vernier s názvem „GoIO Measure“, který je možné zdarma upravit, jak již bylo zmíněno výše. Na papíře jsme si s učitelem nakreslili jednotlivé základní obrazovky budoucí aplikace v podobě jednoduchého GUI modelu, při jehož tvorbě jsme narazili na další menší funkční požadavky. Přihlašovací obrazovka v klientském programu kromě uživatelského jména a hesla obsahovala i adresu serveru, na který se bude program připojovat. Učitel navrhl, aby si klientská aplikace při dalším spuštění sama pamatovala posledně zadanou adresu spolu s uživatelským jménem, aby se tyto údaje nemusely při každém dalším spuštění vyplňovat.

Protože původní program od firmy Vernier umožňoval zaznamenávat data lokálně, spolu s učitelem jsme se shodli na tom tuto funkci zachovat a nabízet ji uživateli pomocí hlavního menu. V tomto případě se data nebudou odesílat na server, ale budou se ukládat do vybraného souboru v počítači ve formátu CSV. Lokální měření by rovněž mělo umožňovat nastavit pro jednotlivá zařízení hraniční hodnoty a v případě potřeby zobrazit chybovou hlášku (spolu s odpovídající barvou na LED diodě).

Při detailnějším zkoumání serverové aplikace učitel zažádal, aby bylo možné v hlavním menu nastavit podmínky, za kterých se při překročení hraničních hodnot bude odesílat e-mailové upozornění. Upozornění by mělo být přiřazeno k danému předpisu zařízení v rámci projektu s možností nastavit rozmezí, po kterých se budou e-mailová upozornění odesílat. Učitel navrhl, abychom na této části začali pracovat až po zajištění základní funkčnosti aplikace a splnění hlavních požadavků, neboť se v tomto případě jedná jen o vedlejší požadavek, který není přímou součástí kreativní výuky žáků.

Kapitola 4

Návrh řešení

4.1 Vybrané řešení

Pro serverovou aplikaci jsem zvolil programovací jazyk PHP a framework Nette [6]. Jazyk PHP se řadí mezi nejpoužívanější jazyky pro tvorbu serverových řešení a komunita jeho programátorů dokáže velmi rychle reagovat a radit s řešením problémů [7]. Nette framework je na rozdíl od svých protějšků (například Symfony 2) velmi jednoduchý na pochopení a vyžaduje mnohem méně námahy při psaní základních funkcí, i když je pravdou, že programátora nenutí dodržovat zásady několikavrstvé architektury. Nicméně svou jednoduchostí je pro omezené finanční prostředky školy vhodný (programátor stráví psaním méně času). Protože je navíc jeho hlavní prioritou bezpečnost, jeví se mi jako nejvhodnější volba.

Serverovou část aplikace jsem se rozhodl spustit na vnějším serveru, ke kterému se bude přistupovat pomocí internetového připojení. Důvodem je, že jsem nechtěl zpočátku omezovat možnosti využití aplikace pouze na úroveň laboratorních prací (popřípadě výuky) v samotných školách, pro které by navíc zavádění aplikace na vlastní server ve vnitřní síťové struktuře představovalo zbytečnou přítěž. Navíc se jedná o novou a nevyzkoušenou věc, ke které se většina učitelů bude pravděpodobně stavět skepticky a každá práce navíc by je mohla odradit.

Co se týče vlastní klientské části aplikace, rozhodl jsem se využít základní program firmy Vernier s názvem „GoIO Measure“ [2], který je napsaný v jazyce C++ a frameworku MFC [8] pro operační systém Windows. Tento základní program už sám o sobě poskytuje funkčnost zobrazování právě měřených dat v pěkném grafu, a dokonce umožňuje vybrat mezi připojenými zařízeními k počítači to, které se bude právě využívat. Program rovněž nabízí možnost zvolení jednotky pro měřenou fyzikální veličinu. Bohužel napsání aplikace pro více platforem by dalece přesahovalo rozsah této práce, proto jsem se zaměřil pouze na operační systém Windows, který je především na školách nejpoužívanějším operačním systémem vůbec. Jak již bylo zmíněno, program „GoIO

Measure“ je napsaný v jazyce C++, který jsem z tohoto důvodu zvolil jako programovací jazyk pro klientskou aplikaci, i když jazyk C# je příjemnější a s ním spojená platforma .NET obsahuje více základních funkcí a knihoven pro práci s daty. Nicméně základní funkčnost programu „GoIO Measure“ podle mých prvotních odhadů zdaleka přesahuje nedostatky, které by v porovnání s platformou .NET a jazykem C# mohl jazyk C++ a framework MFC mít [9].

Všechna zvolená řešení jsou otevřená a nezpoplatněná, takže se jedná o ideální případ pro školy, které disponují velmi omezenými finančními prostředky a malou ochotou pouštět se do novátorských projektů. Samozřejmě mám stále na paměti jednoduchost a snadné ovládání celé aplikace.

4.2 Koncept implementace

Klientská aplikace je založená na frameworku MFC a serverová aplikace na frameworku Nette, které oba podporují architekturu MVC (model, view, controller). Rozhodl jsem se tedy této vlastnosti v obou případech využít a jednotlivé funkčnosti dle této architektury do jisté míry oddělovat, i když dle slov učitele bude aplikace nyní sloužit hlavně pro testování myšlenky kreativní výuky fyziky a nebude pravděpodobně v této podobě nasazena v ostrém provozu na školách u různých učitelů fyziky. Architektura MVC ve stručnosti znamená rozdělení aplikace do tří základních částí, a sice na model, view a controller. Model se stará o základní výpočty včetně komunikace (v případě klientské aplikace se serverem a zařízeními) a jsou v něm obsaženy všechny výpočetní algoritmy, které jsou jádrem aplikace. View je určen pro vykreslování veškeré grafiky na obrazovku a controller se stará o komunikaci s uživatelem, díky čemuž ovládá model a view.

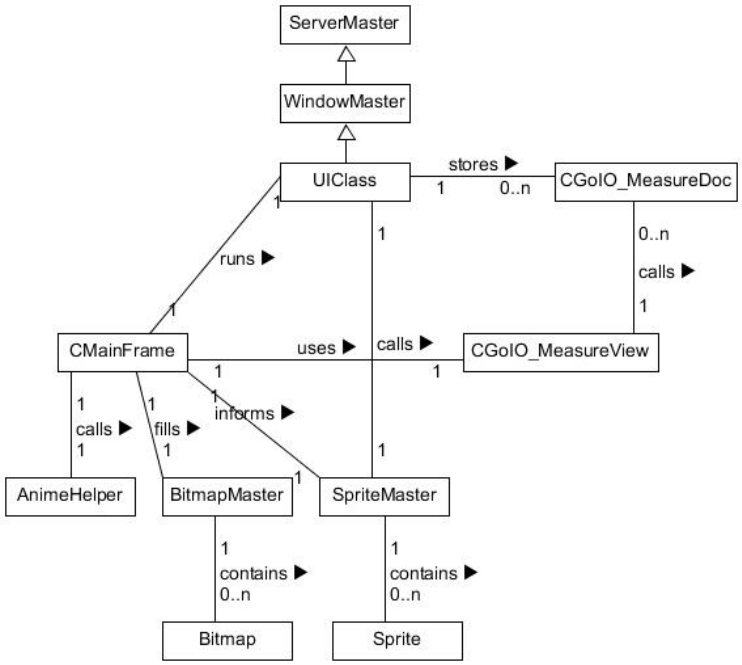
4.2.1 Klient

V případě klientské aplikace se jedná o postupné rozšiřování základního programu „GoIO Measure“, který již obsahuje různě pojmenované třídy, které zapadají do architektury MVC. Jejich původní pojmenování jsem ponechal, a abych tyto třídy odlišil od tříd vlastních, tj. nových, nepřidával jsem k novým třídám jako první znak v pojmenování písmeno „C“, jako tomu bylo u tříd původních, které jsem pouze rozšiřoval a ke kterým jsem přidával nové metody.

Hlavní třídy návrhu, kde každá odpovídá jedné části architektury MVC, jsem pojmenoval a rozdělil následovně:

1. Controller je reprezentován třídou CMainFrame spojenou přímo se základním ovládáním programu.

2. View je reprezentován třídou CGoIO_MeasureView a jejím objektem pView ve třídě CMainFrame.

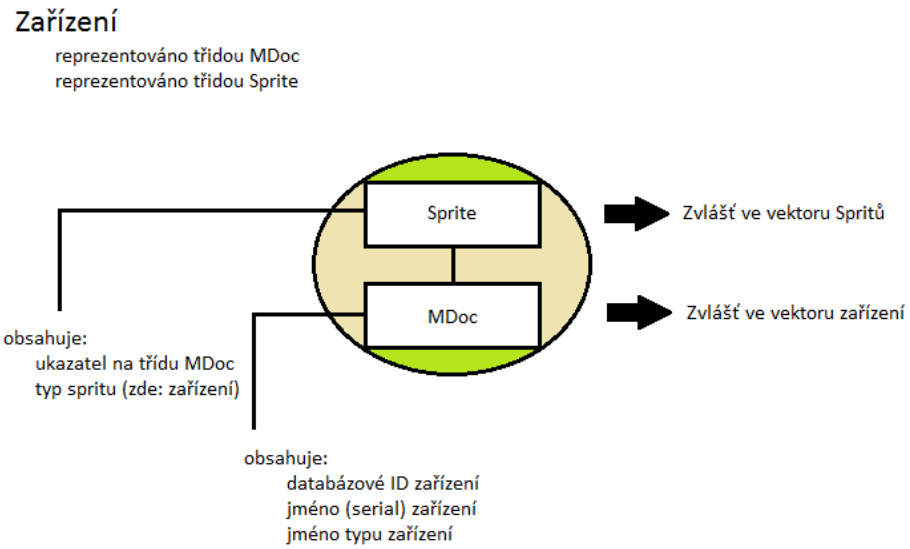


Obrázek 4.1: Diagram základních tříd klientské aplikace

3. Model je reprezentován novou třídou UIClass a jejím objektem pCore ve třídě CMainFrame. Třída UIClass dědí od třídy WindowMaster, starající se o uživatelské údaje, která sama dědí od třídy ServerMaster, jež obstarává veškerou komunikaci se serverem s využitím knihovny cURL [10].

Celý návrhový vzor aplikace se postupem času ukázal být složitější, nicméně tento popis zatím pro další návrhový postup postačí, jelikož jsem zvolil agilní metodiku programování (viz teoretický úvod). Některé třídy jsou podpůrné (pro zpracování bitmap, dialogů, atd.), a není tedy mým hlavním záměrem je v této práci detailně popisovat.

Každé fyzické zařízení je v aplikaci představováno třídou CGoIO_MeasureDoc (dále jen „MDoc“), která je rovněž součástí modelu. Zařízení (např. teploměr) díky této třídě funguje velice jednoduše. Kdykoli je možné spustit či zastavit měření, získávat naměřená data, zjišťovat kalibraci (dostupné jednotky) a nastavovat barvu LED diody (čtyři stavy: zelená, oranžová, červená a vypnuto), která je fyzicky umístěna na každém měřícím zařízení od firmy Vernier. Ve skutečnosti je však v programu zařízení reprezentováno dvěma třídami, a to třídou MDoc a třídou Sprite, pomocí níž se vykreslují objekty na obrazovku.



Obrázek 4.2: Návrh reprezentace zařízení

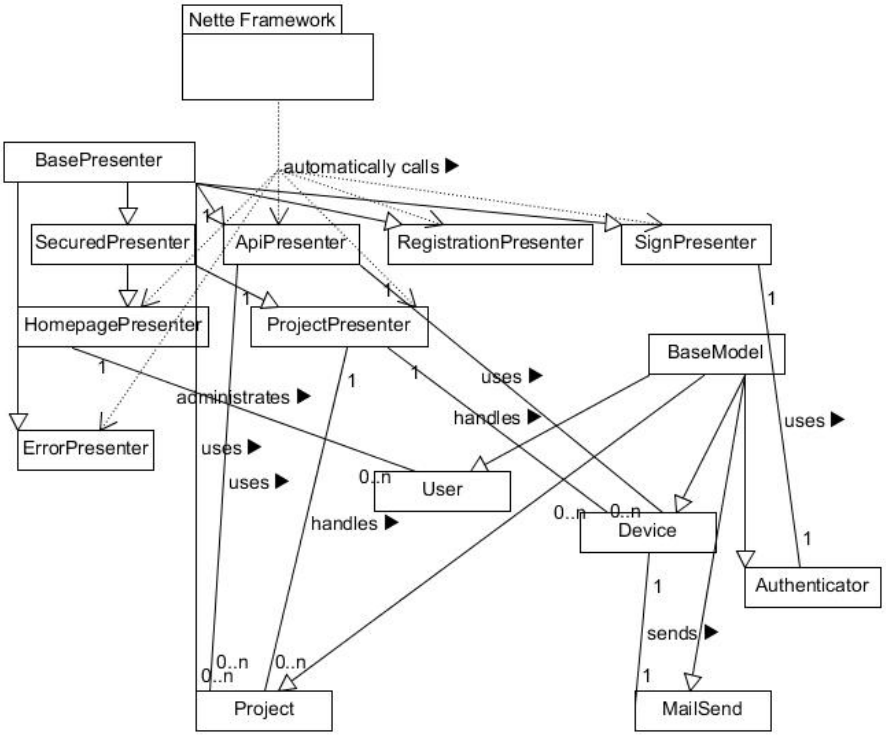
4.2.2 Server

Serverová aplikace je napsána v jazyce PHP s využitím frameworku Nette, jehož strukturu přebírá. Díky tomu je automaticky aplikována architektura MVC, tedy přesněji MVP (model, view, presenter).

Protože modelová část není u serverové aplikace ve srovnání s klientskou příliš komplexní, uvádím ji zde jako první. Návrh programu obsahuje několik tříd modelu – Authenticator, Device, MailSend, Project a User (všechny dědí od základní třídy BaseModel vracející služby pro přístup k databázi). Třída User obsahuje základní metody pro práci s databázovou tabulkou „user“. Pomocí této třídy lze jednoduše upravovat uživatelská data nebo získávat projekty či předpisy zařízení svázané s uživatelem. S třídou User úzce souvisí třída Authenticator, která ověřuje platnost uživatelských dat při přihlášení a obsahuje metodu pro zašifrování hesla pomocí tajného hashe. Heslo se přenáší pomocí URL adresy při komunikaci mezi klientem a serverem a je v tomto formátu uloženo v databázi MySQL.

Další třída s názvem MailSend využívá pouze vestavěnou třídu Nette frameworku pojmenovanou Message k odesílání jednoduchého e-mailu s upozorněním na překročení hraniční hodnoty. Funkce této třídy jsou řízeny objektem třídy Device.

Posledními třídami jsou třídy Project a Device. Jak už název napovídá, třída Project se stará o projekty a třída Device o předpisy zařízení. Obsahují základní metody pro vytváření, editaci a odstraňování projektů, popřípadě předpisů zařízení. Třída Project také obsahuje metodu getDataCSV vytváře-



Obrázek 4.3: Diagram základních tříd serverové aplikace

jící soubor CSV k danému projektu pro export naměřených dat. Třída Device obsahuje metodu setData, která podle komunikace s API uloží naměřenou hodnotu do databáze (zároveň kontroluje, jestli nebyly překročeny nějaké hraniční hodnoty; pokud ano, pak je uživateli podle jeho nastavení zasláno upozornění).

Ke každé větší akci (action), která zhruba odpovídá jedné webové stránce, je přiřazen vlastní presenter (všechny opět dědí od základní třídy BasePresenter vkládající seznam uživatelových projektů do hlavní šablony kvůli zobrazení menu s názvy projektů). Některé presentery dědí od třídy SecuredPresenter, která zajistí to, že se k určitému presenteru dostane pouze uživatel, který je už přihlášen. Přihlášení se provádí pomocí přesměrování na akci (metodu) presenteru Sign. Dalším základním presenterem je třída ErrorPresenter zobrazující příslušnou chybovou stránku (s vysvětlením vzniku chyby) podle číselného kódu chyby (404, ...).

Menší akce webové stránky se spravují užitím nové metody presenteru, která začíná názvem „action“. Ke každé akci (včetně výchozí akce default) je přiřazena jedna HTML šablona. Základní šablona, do které se pak vnořují ostatní šablony podle aktuální akce, se jmenuje „@layout.latte“. Tato šablona

načítá JavaScriptový kód, kaskádové styly a určuje celkovou strukturu stránky pomocí kódu HTML (obsahuje také jednotlivé hypertextové odkazy na jednotlivé presentery programu – hlavní menu). Každá šablona má možnost využívat filtr Nette frameworku s názvem Latte. Pomocí speciálních příkazů přímo v šabloně lze do ní vnořovat programový kód a podmínit vykreslení určitých částí webové stránky, která je s ní spojená.

Kapitola 5

Finanční náročnost

Jak již bylo zmíněno v předchozích kapitolách, současný návrh serverové aplikace využívá pouze Nette framework, který je možné používat a rozšiřovat zcela zdarma. Klientský program je založen na rovněž nezpoplatněném programu „GoIO Measure“ od firmy Vernier a frameworku MFC s využitím rovněž neplacené knihovny cURL pro navázání spojení se serverem. Návrh aplikace, která je součástí této práce, nevyužívá tedy žádný zpoplatněný materiál třetích stran. Finanční náročnost celé aplikace tak v tomto případě odpovídá pouze finanční náročnosti na práci programátora C++ a PHP a provoz serveru nebo zajištění hostingových služeb. Tabulka 5.1 ilustruje odhadovanou finanční náročnost vývoje podobné aplikace, která je prezentovaná v této práci a kterou školám poskytuji zcela zdarma.

Dle názoru učitele je především pro menší školy nemyslitelné podobné náklady na podporu kreativní výuky „svým žákům na míru“ vynaložit. Pevně však věříme, že v budoucích časech budou vznikat neplacené aplikace třetích stran pro kreativní výuku, které budou dostatečně flexibilní a vhodné pro různé obory (fyzika, chemie, biologie, atd.) a které by byly školám poskytované zcela zdarma a nahradily by tak jednoduchý návrh aplikace, který vznikl jako součást této práce. Hlavním benefitem takovýchto aplikací je v úvodu zmíněné zefektivnění výuky.

Tabulka 5.1: Odhad finanční náročnosti

Práce programátora C++ a PHP	Hostingové služby + doména
odhadovaná doba práce: 2 měsíce plného úvazku průměrný plat za 1 měsíc: 32 310 Kč [11]	cca 550 Kč ročně (ceny se výrazně liší u jednotlivých registrátorů domén a poskytovatelů hostingových služeb)
Jednorázově 64 620 Kč (mzda)	Ročně cca 550 Kč (fixní náklad)

Kapitola 6

Implementace klientské části

Klientskou část aplikace jsem začal psát rozšiřováním stávajícího programu „GoIO Measure“, načež jsem implementoval vlastní třídy a jejich metody. Vzhledem k tomu, že jsem stavěl na již existujícím základu a že ze začátku byly požadavky zmíněné v předchozích kapitolách značně nejasné, rozhodl jsem se pro agilní způsob vývoje, kdy jsem jednotlivá rozšíření a změny postupně po malých částech navrhl, implementoval a společně s učitelem (viz výše) otestoval. Cílem této práce není detailně rozepsat každou změnu, nýbrž pouze názorně ukázat návrh a případně implementaci klíčových částí aplikace, a to především z toho důvodu, že aplikace sama je značně robustní a zároveň otevřená následným dalším úpravám, změnám a rozšířením.

6.1 Základní program

Program „GoIO Measure“ obsahoval základní třídy pro model, view a controller (viz návrh řešení). Po několika hodinách strávených zkoumáním kódu jsem zjistil, že jsou programem nejprve volány metody třídy CMainFrame, která dědí od třídy CFrameWnd a která sama o sobě představuje controller. Ten je pak dále rozšířený hlavní třídou samotného programu pro měření s názvem CGoIO_MeasureApp, která mimo jiné určuje i velikost okna aplikace. Struktura objektů, které představují jednotlivá k počítači připojená zařízení, je obsažena v třídě s názvem CGoIO_MeasureDoc, která už podle názvu byla do základní šablony frameworku MFC doplněná programátory firmy Vernier. O vykreslování grafu se pak stará objekt třídy CGoIO_MeasureView, která obsahuje metody pro zobrazení konkrétních výsledků měření přímo v okně aplikace.

6.2 Controller

První z větších tříd, kterou vám nyní představím, je již výše zmíněná třída CMainFrame. Než bude možné popsat další třídy, je nutné se obeznámit s touto třídou, která přímo komunikuje s uživatelem a udává kontext všem ostatním třídám. Napřed je dobré se podívat, jak vypadá základní zdrojový kód této třídy. Nebudu zde rozepisovat detailně všechny metody, nicméně je dle mého názoru důležité, abyste byli seznámeni s její strukturou a deklarací.

```
class CMainFrame : public CFrameWnd
{
private:
        // added
        RECT m_rClientRect;
        int m_iStatus;
public:
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)
        // Attributes
        bool IsCollectingMeasurements();
        void ClearCollectingMeasFlag();
        // Overrides
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual BOOL DestroyWindow();
        // Implementation
        virtual ~CMainFrame();
        afx_msg void OnConnectDev();
        #ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
        #endif
// Protected operations
protected:
        UINT_PTR m_timerId;
        // added
        void SetStatus(int mode);
        // Components
        // Generated message map functions
        afx_msg int OnCreate(LPCREATESTRUCT
            lpCreateStruct);
        afx_msg void OnInitMenuPopup(CMenu* pPopupMenu,
                UINT nIndex, BOOL bSysMenu);
        afx_msg void OnTimer(UINT_PTR nIDEvent);
        afx_msg void OnUpdateGetStatus(CCmdUI* pCmdUI);
```

```
        afx_msg void OnGetStatus();
        afx_msg void OnUpdateSetMeasPeriod(CCmdUI*
            pCmdUI);
        afx_msg void OnSetMeasPeriod();
        afx_msg void OnUpdateStartMeas(CCmdUI* pCmdUI);
        afx_msg void OnStartMeas();
        afx_msg void OnUpdateStopMeas(CCmdUI* pCmdUI);
        afx_msg void OnStopMeas();
        // added
        // VIEW
        afx_msg void OnVUpdateChangeScience(CCmdUI*
            pCmdUI);
        afx_msg void OnVChangeScience();
        afx_msg void OnVUpdateChangeGraph(CCmdUI* pCmdUI
            );
        afx_msg void OnVChangeGraph();
        // SERVER
        afx_msg void OnUpdateConnectDev(CCmdUI* pCmdUI);
        afx_msg void OnUpdateDisconnectDev(CCmdUI*
            pCmdUI);
        afx_msg void OnDisconnectDev();
        // LOCAL
        afx_msg void OnUpdateLoadAllDev(CCmdUI* pCmdUI);
        afx_msg void OnLoadAllDev();
        afx_msg void OnUpdateLoadDev(CCmdUI* pCmdUI);
        afx_msg void OnLoadDev();
        afx_msg void OnUpdateUnLoadAllDev(CCmdUI* pCmdUI
            );
        afx_msg void OnUnLoadAllDev();
        afx_msg void OnUpdateUnLoadDev(CCmdUI* pCmdUI);
        afx_msg void OnUnLoadDev();
        afx_msg void OnUpdateCalibN(CCmdUI* pCmdUI);
        afx_msg void OnCalibN();
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnUpdateActionSetdisplaydepth
                (CCmdUI *pCmdUI);
        afx_msg void OnActionSetdisplaydepth();
        afx_msg void OnUpdateServerUnload(CCmdUI *pCmdUI
            );
        afx_msg void OnServerLoad();
        afx_msg void OnServerUnload();
        afx_msg void OnUpdateServerLoad(CCmdUI *pCmdUI);
        afx_msg void OnLocalSettings();
```

```
        afx_msg void OnUpdateLocalSettings(CCmdUI *
            pCmdUI);
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnSrefresh();
        afx_msg void OnUpdateSrefresh(CCmdUI *pCmdUI);
};
```

Úryvek 6.1: Deklarace třídy CMainFrame

Už podle metod a jejich názvů je zřejmé, že třída sama představuje všechny hlavní funkce controlleru od registrace všech zařízení (modelu) přes vytváření obrazovek a grafů (view) až po spuštění vlastního měření. Na výpisu zdrojového kódu této třídy je patrné, že názvy metod přímo korespondují s akcemi, které vyvolá uživatel. Třída není příliš algoritmicky náročná. Obsahuje však několik klíčových míst, na která je důležité upozornit. Kromě toho, že obstarává komunikaci s uživatelem, určuje stav aplikace (nenačteno, server, lokální měření; metoda SetStatus), otevírá okno programu a řídí modelový cyklus (metoda OnTimer).

Anglické názvy metod přesně odpovídají jejich funkčnosti. Malinko nejasná se může jevit funkce metod OnVChangeScience a OnVChangeGraph, které se však pouze starají o volání objektů tříd view (pohledu). Klíčové slovo „science“ odpovídá vykreslení úvodní obrazovky s detaily aktuálního měření a jednotlivých načtených zařízení, zatímco klíčové slovo „graph“ odkazuje na obrazovku s grafem, ve kterém je znázorněn průběh nedávných naměřených hodnot.

Nejdůležitější metodou třídy CMainFrame je nejspíše metoda s názvem OnTimer, která se stará o zachycení vnitřního cyklu programu a která volá modelovou část. Níže je uveden její zdrojový kód.

```
void CMainFrame::OnTimer(UINT_PTR nIDEvent)
{
        CGoIO_MeasureView *pView = (CGoIO_MeasureView *)
            GetActiveView();
        // Model cycle
        pCore->toCycle();
        if (pView)
        {
                // Draw it
                // neccessary for the right drawing
                if (pCore->getDrawingScreen() == 0)
                        this->OnSize(0, m_rClientRect.
                            right, m_rClientRect.bottom);
                pView->Invalidate();
                pView->UpdateWindow();
        }
```

```
        // Control
        if (pCore->isInitialized())
                SetStatus(1);
        else
        {
                switch (pCore->getWhoIsLoaded())
                {
                        case WHO_LOCAL:
                                SetStatus(2);
                                break;
                        case WHO_NOONE:
                                SetStatus(0);
                                break;
                }
        }
}
```

Úryvek 6.2: Metoda OnTimer

Metoda se spouští cyklicky v předem určené časové periodě (přibližně 30krát za sekundu) a řídí modelový cyklus (pomocí objektu pCore třídy UIClass) a vykreslování (pomocí objektu pView třídy CGoIO_MeasureView).

Zároveň tato metoda mění stav aplikace podle toho, jestli uživatel zvolil lokální či serverové měření. S touto metodou úzce souvisí metoda OnSize, která reaguje na změnu okna aplikace a podle toho mění velikost různých bitmap a hranice spritů, tj. hranice, za které se už daný sprite nemůže posunout. Sprity budou popsány dále v části věnované pohledu (view).

Třída CMainFrame ovšem využívá i objekty některých pomocných tříd, které inicializuje ve svém konstruktoru.

```
CMainFrame::CMainFrame()
{
        pCore = new MUI(); // Initializing the program
            core
        pDocs = new dVect(); // New vector for devices
        pProgramBitmaps = new bVect(); // New vector for
            bitmaps
        pSprites = new SpriteMaster(); // New vector for
            Sprites
        pMaster = new BitmapMaster(); // New info-
            provider
        pSafeAnime = new AnimeHelper(); // New the anime
            for the safe
        m_iStatus = 0;
```

```
}
```

Úryvek 6.3: Konstruktor třídy CMainFrame

1. objekt třídy UIClass
2. vektor objektů třídy CGoIO_MeasureDoc (měřící zařízení)
3. vektor objektů třídy Bitmap
4. objekt třídy SpriteMaster
5. objekt třídy BitmapMaster
6. objekt třídy AnimeHelper
7. proměnná, ve které je uložený stav aplikace

Třída UIClass bude popsána později v modelové části, protože je velice složitá a pro její pochopení je nutná znalost ostatních tříd. Pojďme se nyní společně podívat na třídu CGoIO_MeasureView a objekty ostatních tříd view (pohledu), které využívá, především objekty tříd Bitmap, Sprite, SpriteMaster, BitmapMaster a AnimeHelper.

6.3 View

Třída, která spravuje všechny základní funkce view (pohledu), nese název CGoIO_MeasureView. Základ této třídy byl opět převzat z programu „GoIO Measure“ od firmy Vernier. Třída sama se stará o vykreslování všech objektů do bufferu (datového zásobníku) a poté na obrazovku. Provádí také obsluhu některých událostí, jejichž správu většinou přenechává objektu třídy UIClass. Níže uvádím její deklaraci, kterou posléze ve stručnosti popíšu.

```
class CGoIO_MeasureView : public CView
{
private:
        // backbuffer
        CDC *m_hBBFdc;
        CBitmap *m_BBFbitmap;
        // font
        CFont *m_pFont;
protected:
        void RecordGoIOString(LPCSTR pBuf, int buf_len,
           LPCSTR label);
        // Generated message map functions
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
```

```
        DECLARE_MESSAGE_MAP()
        double graph_history_y_min;
        double graph_history_y_max;
public:
        // Implementation
        virtual ~CGoIO_MeasureView();
        CGoIO_MeasureView();
        DECLARE_DYNCREATE(CGoIO_MeasureView)
        CGoIO_MeasureDoc* GetDocument() const;
        void SetGraphHistory(double y_min, double y_max)
            ;
        void GetGraphHistory(double &y_min, double &
            y_max);
        // Operations
        void OnDraw(CDC* pDC);  // overridden to draw
            this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnInitialUpdate();
        // added
        void DrawScience(CDC *pDC);
        void DrawGraph(CDC *pDC);
        void ReleaseRefreshForIssues();
        void CreateBBF() { m_BBFbitmap = new CBitmap();
            m_hBBFdc = new CDC(); }
        void DisposeBBF() { if(m_BBFbitmap) delete
            m_BBFbitmap; if(m_hBBFdc) delete m_hBBFdc; }
        // Message handlers
        afx_msg void OnLButtonDown(UINT nFlags, CPoint
            point);
        afx_msg void OnLButtonDblClk(UINT nFlags, CPoint
             point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint
            point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint
            point);
        afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt,
             UINT nFlags);
};
```

Úryvek 6.4: Deklarace třídy CGoIO_MeasureView

Hlavní metodou je metoda OnDraw, která předává řízení metodám DrawScience nebo DrawGraph podle toho, jakou „obrazovku“ si uživatel přeje zobrazit, jestli základní s detaily měření a k počítači připojených zařízení nebo graf. Zkrátka vše, co lze vidět v klientské části okna pod hlavním menu, je vykres-

leno zásluhou této třídy a jejích metod, které však opět nejsou algoritmicky příliš náročné, pouze mohou být při prvním zkoumání vyčerpávající.

Na předchozím úryvku zdrojového kódu jste si mohli všimnout, že třída CGoIO_MeasureView provádí i základní obsluhu událostí, jakými jsou stisk tlačítka na klávesnici nebo pohyb myši po obrazovce. Nicméně je potřeba zmínit, že tato obsluha, týkající se událostí pouze v okně aplikace, je v kódu daných metod předána jiným částem programu, například objektu třídy UIClass, který události dále zachytává a zpracovává.

První z pomocných tříd je třída Bitmap. Tato třída byla částečně převzata z knihy [12] a umožňuje pohodlnou práci s bitmapovými soubory (načítání a úpravy). Je vhodné zmínit metodu Draw, která pomocí argumentu iStyle dokáže buď bitmapu roztáhnout podle nastavené výšky a šířky, nebo jednu barvu v bitmapě nastavit jako průhlednou. Třída BitmapMaster zapečeťuje všechny bitmapy, které program využívá. Tato třída je načte ze souboru nebo ze zdrojů uložených v programu. Také se řídí informací, jaký grafický styl si uživatel vybral z hlavního menu aplikace, a podle toho načte příslušné bitmapy a uloží do pomocného souboru (general.ini) poznámku, jaký styl se má opět použít při dalším spuštění aplikace. Tato třída především obsahuje podmínky a objekty pro práci se soubory, aby ve správný čas načetla správné bitmapy (podle příslušného grafického stylu).

Další pomocná třída, která byla také částečně převzata z knihy [12], je třída Sprite. Pomocí této třídy lze vytvořit objekt, který se bude po obrazovce pohybovat, resp. bude se pohybovat nastavená bitmapa (nebo text), kdy se tak pomocí cyklu programu může měnit její pozice. Každému objektu je také přiřazená souřadnice z určující, který objekt se při překrytí dvou objektů zobrazí. Další z pomocných tříd je třída SpriteMaster. I tato třída je částečně převzata z knihy [12]. Pomocí ní lze řídit jednotlivé kolize spritů (objektů), reagovat na ně a samozřejmě všechny sprity spravovat v jednom poli (vektoru).

Poslední třídou, která je zajímavá z hlediska pohledu (view), je třída s názvem AnimeHelper. Tato malá pomocná třída obsahuje metody pro vytvoření animace grafického „trezoru“, který se otevírá po načtení spojení se serverem nebo po výběru lokálního měření.

6.4 Model

Před tím, než vám představím vyčerpávající třídu UIClass, bych vás rád uvedl do problematiky správy zařízení v aplikaci a ostatních podpůrných činností, které obstarávají ostatní menší třídy modelu klientské aplikace.

6.4.1 Zařízení pro měření

O správu jednoho měřícího zařízení se stará objekt třídy CGoIO_MeasureDoc (MDoc). Základ této třídy byl opět převzat z programu „GoIO Measure“ od

firmy Vernier. Její objekt rovněž spravuje naměřené hodnoty, pro jejichž uložení využívá objekt třídy GCircularBuffer, a jednotku měření. Tato třída také umožňuje základní práci se zařízením – získání poslední naměřené hodnoty, změnu stavu LED diody, apod. Třída dále například obsahuje pomocné hodnoty předpisu zařízení ze serveru, hranice hodnot pro lokální měření nebo informaci, jestli se má zobrazit zpráva, že zařízení překročilo maximální či minimální přípustnou hodnotu. Detailnější popis některých jejích metod bude uveden v souvislosti se třídou UIClass.

V programu existuje veliké množství dialogů, které umožňují měnit různá nastavení (většinu z nich volá uživatel z hlavního menu v horní liště aplikace). Musí tedy pro každý z nich existovat dosti rozsáhlá třída. Zde pro příklad uvádím deklaraci třídy SetCalibrationDlg, která řídí jednoduchý dialog pro změnu kalibrace (jednotky) u měřícího zařízení.

```
class SetCalibrationDlg : public CDialog
{
        DECLARE_DYNAMIC(SetCalibrationDlg)
private:
        sVect *m_devices;
        sVect *m_calibrations;
        gVect *m_handles;
        int m_whichDevice;
public:
        int m_whichKalib;
        HSensor m_hChange;
        SetCalibrationDlg(CWnd* pParent = NULL);
        virtual ~SetCalibrationDlg();
        enum { IDD = IDD_SET_CALIB };
        void AddDevice(string name, GOIO_SENSOR_HANDLE *
            pHandle);
protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        DECLARE_MESSAGE_MAP();
public:
        afx_msg void OnCbnSelchangeComboboxex1();
        afx_msg void OnCbnSelchangeCombo1();
        afx_msg void OnCbnSelchangeCombo2();
        afx_msg void OnBnClickedOk();
};
```

Úryvek 6.5: Deklarace třídy SetCalibrationDlg

Třída obsahuje pole objektů zařízení, jejich názvů a kalibrací. Podle toho, jakou kalibraci uživatel zvolí, přiřadí se jednotka pro měření příslušnému objektu zařízení. Celkově jsou v programu využívány následující dialogové třídy:

1. SetLocalDlg
2. DisplayDepthDlg
3. AssocSetDlg (zjednodušená verze SetCalibrationDlg)
4. SetCalibrationDlg
5. ServerLoadDlg
6. SetMeasPeriodDlg
7. SetLocalBoundDlg
8. AddDeviceDlg
9. CAboutDlg

6.4.2 Pomocné třídy modelu

První pomocnou třídou modelu je třída s názvem TimeHelper. Jedná se o velice jednoduchou třídu, která spravuje čas od počátku měření (v milisekundách), dále čas od doby, kdy proběhlo poslední odeslání dat na server, a relativní čas, kolik odeslání dat na server již proběhlo (nutné pro aktualizaci všech dat se serverem).

Další pomocná třída, která se tentokrát zabývá sprity (objekty na obrazovce), které jsou označeny jako sprity pro projekty (názvy projektu), je třída SpriteProjectOperator. Sprity pro projekty se uživateli zobrazují po pravé straně obrazovky po načtení spojení se serverem. Hlavním účelem uložení spritů v této třídě je, aby byly tyto sprity vždy vykresleny přesně pod sebou. Funkčně související třída SpriteCreationHelper má podobný význam jako třída BitmapMaster (zmíněna v části týkající se view). Po zavolání určitých metod sama vytvoří například sprity šipek (umístěných vedle spritů projektů), sprite pro předpis zařízení, přístroj a projekt. Třída se zabývá také inicializací barev pro všechny sprity s textem – barva textu se volí podle barvy definované v grafickém stylu, který uživatel vybral z hlavního menu.

Pomocná třída RefreshHelper se zabývá výpomocí při synchronizaci dat se souborem nebo se serverem. Úzce souvisí s dále zmíněnou třídou UIClass.

```
class RWriteHelper
{
private:
        rlVect m_vRest;
public:
        RWriteHelper() { LoadFromFile(); }
        virtual ~RWriteHelper() {}
        void LoadFromFile();
```

```
        void GetFromRest(MDoc *pDevice);
        void GetToRest(MDoc *pDevice);
        rlVect & GetVect() { return m_vRest; }
};
class RefreshHelper : public RWriteHelper
{
public:
        RefreshHelper();
        virtual ~RefreshHelper() {}
        void RefreshDeviceInfos(devVect *pDInfos);
};
```

Úryvek 6.6: Deklarace třídy RefreshHelper

Úkolem třídy RefreshHelper je, v případě lokálního měření, načíst z dočasného souboru (pokud existuje) hraniční hodnoty a přiřadit je jednotlivým připojeným zařízením podle jejich sériového klíče (který je závislý jak na konkrétním zařízení, tak na USB portu). V případě serverového měření objekt třídy RefreshHelper aktualizuje data uložená v předpisech zařízení (ve zdrojovém kódu: device info).

Ryze „lokalizační“ třídy, které se starají o texty v programu, jsou dvě – ErrorClass a ProgramTexts. Druhá jmenovaná obsahuje pole s texty, které se využívají přímo v programu (např. v dialozích). ErrorClass obsahuje především texty, které se týkají chybových hlášek. Dále pak tato třída uchovává jména souborů, které se v programu generují a čtou. Také obsahuje některé metody pro práci s textem, především pro úpravu textu do formátu URL, převod textu mezi znakovými sadami, apod.

Tímto je ukončený seznam základních pomocných tříd modelu, nyní následuje popis objektu pCore a tříd UIClass, WindowMaster, ServerMaster s dodatečným popisem tříd, které s těmito třemi třídami souvisí.

6.4.3 Komunikace se serverem

První třídou, od které hlavní třída modelu UIClass dědí, je třída s názvem ServerMaster. Tato třída konečně zajišťuje vlastní komunikaci se serverovou částí aplikace. Veškerá komunikace probíhá pomocí speciálně zadané URL adresy, které API rozhraní na serveru rozumí. Napřed se společně podíváme, jak třída vypadá, načež bude uveden její rozbor.

```
class ServerMaster
{
private:
        // static variables ------------
        static string m_sBeforeURL;
        static string m_sAfterURL;
        // data
```

```
        static string m_sWebURL;
        static string m_sUserName;
        static string m_sUserPassword;
        static string m_sHashUserPassword;
        // status
        bool m_bIsInitialized;
        static bool m_bServerDenied;
        static bool m_bNotConnected;
protected:
        static int sendData(devVect *values = NULL, bool
            bThread = true);
        static void sendDataThread(void *sUrl);
        static void sendDataThreadByFile(void *sUrl);
        static ProjectsInfo *getUserProjects();
        // UIClass handling
        static bool refreshServer() { return
           m_bServerDenied; }
        // WindowMaster handling
        static int refreshAll();
        // initializing
        static int setUser(string userName, string
           userPassword);
        static void setWebData(string webURL) {
           m_sWebURL = webURL; }
        static void setUserProject(int projectID) {
           m_iUserProjectID = projectID; }
        static devVect *getProjectDevices();
        static int createDeviceInfo(devVect *devices,
           Value & sdata);
        static void refreshedServer(bool state) {
           m_bServerDenied = state; }
public:
        // public attributes
        static int m_iUserProjectID;
        // methods
        ServerMaster();
        virtual ~ServerMaster();
        static int setNewDeviceInfo(DeviceInfo *pDI,
           MDoc *pDev);
        static void performURL(void *sUrl);
        static int hasMeasuredData(int iDeviceID);
        void setInitialized(bool isInitialized = true);
        bool isInitialized() { return m_bIsInitialized;
           }
```

```
};
```

Úryvek 6.7: Deklarace třídy ServerMaster

Konstrukce každého požadavku na server přes URL vyžaduje adresu serveru, uživatelské jméno a speciálně šifrované heslo pomocí skrytého hashe. Dále je potřeba vědět, jestli bylo navázáno spojení se serverem, jestli během případného odesílání dat nebylo přerušeno internetové připojení či jestli server neodmítl data přijmout. Všechny tyto informace jsou uloženy v atributech třídy.

Třída sama pak na jedné straně obsahuje prosté funkce k uložení dat do těchto atributů, na druhé straně se pak zabývá funkcemi pro odesílání dat na server, synchronizaci dat se serverem, načtení projektu a příslušných předpisů zařízení a funkcí pro kontrolu existence uživatele.

Kód každé z metod se může na první pohled zdát krátký na to, co všechno daná metoda provádí. Je to dáno především přehlednými funkcemi knihovny cURL starajícími se o komunikaci se serverem a také jednoduchým užitím parseru JSONCpp, který se zabývá převodem souboru, naformátovaného podle konvence JSON, do vestavěných datových typů jazyka C++. V neposlední řadě tomu napomáhají i nadefinované datové typy speciálně pro tuto aplikaci, které činí kód čistším a přehlednějším.

Nyní následuje ukázka kódu metody sendDataThread, která se stará o vlastní odeslání dat na server. Tato metoda je volána metodou sendData pomocí jednoduchého příkazu *_beginthread(&ServerMaster::sendDataThread, NULL, sUrl);* kterým se vytvoří nové programové vlákno určené pro vykonání kódu této metody. Nové programové vlákno bude spuštěno nezávisle na aktuálním chodu programu, měření tedy nebude nikdy pozastaveno, a to ani na počítačích s velice pomalým internetovým připojením. Kód této metody se tedy vykonává asynchronně (uživatel má možnost pomocí klávesy W odeslat data na server synchronně, což ale v žádném případě není doporučeno; tato klávesa je určena především pro testování rychlosti komunikace se serverem).

```
void ServerMaster::sendDataThread(void *sUrl)
{
        if (sUrl != NULL)
        {
                // URL
                string *url = (string*)sUrl;
                // helpful variables
                bool bError = false;
                ofstream errorStream;
                CURL *curl_handle;
                stringstream ss;
                Reader reader;
                Value data;
```

```
// initialization
curl_global_init(CURL_GLOBAL_ALL);
curl_handle = curl_easy_init();
struct helpMemoryStruct chunk;
chunk.memory = (char*)malloc(1);
chunk.size = 0;
curl_easy_setopt(curl_handle,
   CURLOPT_URL, url->c_str());
curl_easy_setopt(curl_handle,
   CURLOPT_WRITEFUNCTION,
   helpWriteMemory);
curl_easy_setopt(curl_handle,
   CURLOPT_WRITEDATA, &chunk);
curl_easy_setopt(curl_handle,
   CURLOPT_USERAGENT, "
   FlowerSafeguardClient");
CURLcode res = curl_easy_perform(
   curl_handle);
if (res != 0)
{
        bError = true;
        m_bNotConnected = true;
}
else
{
        m_bNotConnected = false;
        newStream(ss);
        ss << chunk.memory;
        reader.parse(ss, data);
        if(chunk.memory)
                free(chunk.memory);
        if (data["success"].asString()
           != "true")
        {
                if (!m_bServerDenied)
                {
                        m_bServerDenied
                           = true;
                }
        }
}
if (bError)
{
```

```
                        CreateDirectory(ErrorClass::
                            generatedDirectories[3], NULL
                            );
                        errorStream.open(ErrorClass::
                            generatedFiles[3], ios_base::
                            app);
                        if (errorStream.is_open())
                        {
                                errorStream << url->
                                    c_str() << '\n'; //
                                    divider '\n'
                                errorStream.close();
                        }
                }
                // cleaning
                curl_easy_cleanup(curl_handle);
                curl_global_cleanup();
                delete sUrl;
        }
}
```

Úryvek 6.8: Metoda sendDataThread

Příkaz *curl_easy_perform* odešle veškerá dosud naměřená a neodeslaná data na server pomocí jediné URL adresy, a to pro každý připojený senzor (resp. pro jeho předpis). Metoda sendDataThread pak dále testuje, zda byla data skutečně odeslána (test internetového připojení pomocí proměnné res), a pokud ano, metoda rovněž otestuje, zda server data přijal (pomocí pole data[“success”] načteného metodou parse objektu reader). Pokud metoda pouze zjistila, že internetové připojení neexistuje nebo že je velmi slabé, data se uloží pomocí objektu errorStream do souboru a odešlou se ihned, jakmile bude internetové připojení znovu fungovat. Samotné měření se tedy nepozastaví.

Mohl by zde být dále uveden rozbor dalších metod, které v programu plní velice důležitou funkci, ovšem jejich kód je velice podobný kódu metody sendDataThread – a to většinou pouze s tím rozdílem, že se tyto metody dotazují serverového API rozhraní na jinou akci (opět pomocí konkrétní adresy URL).

Přímým potomkem třídy ServerMaster je třída s názvem WindowMaster, která pouze mateřskou třídu ServerMaster rozšiřuje o podpůrné funkce. Kód je sice značně jednoduchý, ale vzhledem k tomu, že se přímo vztahuje k požadavkům navázání úspěšné komunikace se serverem, je umístěn v samostatné třídě, která je tedy mezistupněm mezi třídou ServerMaster a UIClass.

Třída WindowMaster spravuje objekt třídy dialogu, který se zobrazí po spuštění programu. Jelikož se tento dialog opět zobrazí, pokud uživatel špatně zadá přihlašovací údaje, třída WindowMaster pomocí metody startInitialize

napřed načte data z pomocného souboru do dialogu, pokud existují (URL serveru a uživatelské jméno), a potom je v dialogu uchová spolu se změnou, kterou provedl uživatel. Pokud uživatel zadá prázdné údaje (uživatelské jméno nebude vyplněno), data se opět doplní ze souboru.

Do pomocného souboru (config.ini) se data ukládají pouze tehdy, pokud bylo úspěšně navázáno připojení k serveru (pomocí metody ServerMaster::setUser, která připojení testuje pomocí počtu projektů uživatele). Metoda třídy WindowMaster s názvem startInitialize je využívána třídou CMainFrame, která ji spouští tak dlouho za sebou, dokud se uživatel úspěšně nepřipojí k serveru nebo sám dialog nezavře pomocí jiného tlačítka.

6.4.4 Hlavní třída modelu

Třída UIClass je nejkomplexnější a nejrozsáhlejší třídou celého programu. Proto jsem také zdrojový kód jejích metod rozdělil do dvou souborů – kód obecný a kód zpracovávající objekty tříd dialogů, které třída pomáhá zobrazit. Samozřejmě je ze začátku nejdůležitější se obeznámit s deklarací této třídy.

```
class UIClass : public WindowMaster
{
private:
        // static variables ------------
        static int m_iPeriodCheck;
        static int m_iPeriodSend;
        // device-infos
        // mainly due to the sendData method
        devVect *m_pDevStack;
        // VIEW
        Sprite *m_spActiveSprite;
        int m_iWhoIsLoaded;
        int m_iScreen;
        // MAIN
        int m_iDepth; // displayed depth
        bool m_bIsCollectingMeasurements;
        TimeHelper *m_thTime;
        RefreshHelper *m_rhRefresh;
        // local
        int m_iLocalSettings[3];
        string m_sLocalFileName;
        string m_sLocalJustFileName;
        // threading
        static devVect *m_psNewDevices;
        static ProjectsInfo *m_piNewProject;
        static bool m_bCheckThreadExists; // safety
```

```
        static bool m_bInternetNot;
public:
        UIClass();
        void CreateDev();
        void DisposeDev();
        virtual ~UIClass();
        void checkMeasuringStopped();
        // General
        void toCycle();
        void MouseMove(int x, int y);
        void MouseDown(int x, int y);
        void MouseUp(int x, int y);
        void Associate(Sprite *device, Sprite *devinfo);
        bool sendAssocToServer(Sprite *spDevice, Sprite
           *spDeviceInfo);
        void Unassociate(Sprite *pDevice);
        void MouseDoubleDown(int x, int y);
        void KeyDown(UINT nKey, bool bNoMessage = false)
           ;
        int setProject(int iID);
        void disconnect();
        // threading
        static void checkData(void *check);
        // helpful
        void checkCycleValues(MDoc *device, double value
            = 0);
        // Data for View
        int getMeasuringSec() { return (int)(m_thTime->
           GetMeasuring()); }
        void changeDrawingScreen(int screen = 0) {
           m_iScreen = screen; }
        int getDrawingScreen() { return m_iScreen; }
        void setWhoIsLoaded(int iWho) { m_iWhoIsLoaded =
            iWho; }
        int getWhoIsLoaded() { return m_iWhoIsLoaded; }
        int toPaint(CDC *pDC, int iDrive = DRIVE_ALL);
        // Data for Main
        DeviceInfo *getDeviceInfo(MDoc *pDevice);
        void setStartedMeasuring(bool bState) {
           m_bIsCollectingMeasurements = bState; }
        bool isStartedMeasuring() { return
           m_bIsCollectingMeasurements; }
        bool isInMove(int iType = ST_ALL);
        void loadAllDevices(int direction);
```

```
        void loadDevice(int direction);
        void checkLoadAssoc(Sprite *pDev);
        void unLoadAllDevices();
        void unLoadDevice();
        void startMeasuring(bool bAnullTime = true);
        void stopMeasuring();
        sVect *checkDevices();
        // for safety
        // MDoc *openDevice(string tmpstring) { return
            addDevice(getDevice(tmpstring)); }
        MDoc *addDevice(MDoc *handle) { pDocs->push_back
            (handle); return handle; }
        MDoc *getDevice(string tmpstring);
        void changeCalibration();
        void setGeneralSettings();
        void setMesPeriod();
        void changeLocalSettings();
};
```

Úryvek 6.9: Deklarace třídy UIClass

Už podle deklarace třídy UIClass si můžete povšimnout, že je značně složitá a obstarává celou řadu funkcí, bez nichž by klientská aplikace nemohla fungovat. Třída obsahuje „modelový cyklus“ starající se o sbírání naměřených hodnot, sledování přesahu minimální a maximální hraniční hodnoty, atd. Dále také obsahuje práci se sprity, s připojením a odpojením senzorů, práci se správou projektů, předpisů zařízení a průběhu měření. Rovněž také od controlleru přebírá dodatečnou obsluhu některých událostí. Názvy jednotlivých metod dokáží do jisté míry samy o sobě vysvětlit jejich funkci, což je v deklaraci navíc doplněno komentáři.

Za povšimnutí také stojí, že některé atributy třídy souvisí s výstupem některých dialogů a uchovávají jejich výstupy (například proměnná m_iDepth). Nicméně nejdůležitější ze všeho je metoda toCycle, která provádí nejdůležitější úkony v programu. Její kód je příliš dlouhý, proto bude následovat rozbor jen některých jeho nejdůležitějších částí, po jejichž objasnění je pak snadné pochopit fungování metody jako celku.

```
// adding the data to the server
if (m_iWhoIsLoaded == WHO_SERVER)
{
        if ((*i)->isAssociated())
        {
                // if measurements are started; start
                if (this->isStartedMeasuring())
                {
```

```
                        if (!(*i)->IsStarted())
                                (*i)->Start();
                }
                DeviceInfo *info = (*i)->m_pDeviceInfo;
                info->m_vValues.m_dBaseValue.push_back(
                    rMeasurement);
                info->m_vValues.m_tTime.push_back(
                    m_thTime->GetTimeMs());
                this->checkCycleValues((*i),
                    rMeasurement);
        }
}
```

Úryvek 6.10: Část kódu metody toCycle (serverové měření)

Pokud uživatel zvolí serverové měření, pak po základním načtení, iteraci všech senzorů (zde je každý přístroj reprezentován iterátorem (*i)) a iteraci naměřených hodnot (zde jsou naměřené hodnoty reprezentovány proměnnou rMeasurement) se zpracuje tento kód. Přitom je kontrolováno, jestli je senzor asociován s předpisem zařízení a jestli je měření skutečně zahájeno. Pokud tomu tak je, získá se předpis zařízení a hodnoty měření a času se uloží do proměnných objektu jeho struktury. Pak se volá metoda checkCycleValues, která v případě překročení hraničních hodnot uloží zprávu do proměnné objektu CGoIO_MeasureDoc. Zpráva je posléze zobrazena objektem pView třídy CGoIO_MeasureView. Následující ukázka zobrazuje alternativní kód, který se provede v případě, když uživatel zvolí lokální měření. Na začátku běhu uvedeného kódu probíhá opět kontrola, jestli je měření skutečně na daném zařízení zahájeno. Dále se pak kontroluje, jestli si uživatel přeje ukládat data do souboru. Pokud ano, naměřená hodnota se rozdělí na celé číslo a na desetinnou část a uloží se do souboru včetně jednotky. Dále se pak uloží čas ve formátu, který uživatel vybral pomocí dialogu pro nastavení lokálního měření.

```
// adding the data to the local
else if (m_iWhoIsLoaded == WHO_LOCAL)
{
        // if is measurements started; start
        if (this->isStartedMeasuring())
        {
                if (!(*i)->IsStarted())
                        (*i)->Start();
        }
        // write to the file
        if (m_iLocalSettings[LINDEX_EXISTS] ==
            LFILE_ENABLED)
        {
```

```
                ofstream of(m_sLocalFileName, ios_base::
                    app);
                if (of.is_open())
                {
                        char divider = ';';
                        int measurement = (int)
                            rMeasurement;
                        int remains = (int)((
                            rMeasurement - measurement)*
                            CORE_PRECISION);
                        of << (*i)->GetName() << divider
                            ;
                        of << measurement << "." <<
                            remains << divider << (*i)->
                            getUnit();
                        // write time
                        time_t timeNow = m_thTime->
                            GetTime();
                        tm *timeinfo = localtime(&
                            timeNow);
                        if (m_iLocalSettings[
                            LINDEX_SECOND] ==
                            LFILE_ENABLED)
                                of << divider <<
                                    m_thTime->
                                    GetMeasuring();
                        if (m_iLocalSettings[LINDEX_ASC]
                             == LFILE_ENABLED)
                                of << divider << asctime
                                    (timeinfo);
                        if (m_iLocalSettings[LINDEX_ASC]
                             != LFILE_ENABLED &&
                                m_iLocalSettings[
                                    LINDEX_SECOND] ==
                                    LFILE_ENABLED)
                                        of << endl;
                        of.close();
                }
}
```

Úryvek 6.11: Část kódu metody toCycle (lokální měření)

Další kód metody toCycle pak ukládá hodnoty do grafu, odesílá data a synchronizuje projekty a předpisy zařízení se serverem v předem definovaných časových intervalech. Kód většiny dalších metod je také zajímavý, metody se

převážně starají o obsluhu připojených přístrojů, nicméně nepřináší výrazné rozdíly do již výše prezentovaných funkcí. Například metoda checkDevices využívá vestavěných funkcí knihovny Vernier SDK, kde stačí jen pro každý výrobní typ přístroje zavolat speciální funkci, která zjistí sériový klíč daného zařízení, a poté tento klíč uložit do vektoru. Další metody se starají především o práci se sprity – asociace předpisu zařízení a senzoru, dodatečná obsluha událostí (vyvolaných klávesnicí a myší), apod. Důležitá metoda je metoda s názvem sendAssocToServer, která kontroluje, jestli zařízení může měřit v jednotce definované na serveru, pokud ne, asociace se nevytvoří. Další možností je, že na serveru nebude definována jednotka, a v takovém případě metoda zobrazí dialog s výběrem jednotky. Pokud je vše v pořádku, tak se na server pomocí metod mateřské třídy ServerMaster odešle sériový klíč zařízení a provede se asociace (toto je důležité pro automatické připojení senzoru k předpisu, což nastává po první asociaci; toto automatické připojení probíhá pomocí sériového klíče). Popis všech metod by byl zdlouhavý, protože jich je skutečně hodně, avšak nyní uvedu ještě úryvek jedné kratší metody ze skupiny funkcí, které se starají o spouštění dialogů a správu objektů dialogových tříd.

```
void UIClass::setGeneralSettings()
{
        CDisplayDepthDlg dlg;
        dlg.SetServerLoaded(this->isInitialized());
        dlg.m_displayDepth = 0;
        // getting the max value
        dlg.m_displayDepth = m_iDepth;
        if (IDOK == dlg.DoModal())
        {
                // setting the max
                for (diVect i = pDocs->begin(); i <
                    pDocs->end(); i++)
                {
                        if (dlg.m_displayDepth > 0)
                        {
                                m_iDepth = dlg.
                                    m_displayDepth;
                                (*i)->
                                    SetMaxNumMeasurementsInCirBuf
                                    (m_iDepth / (*i)->
                                    GetMeasurementPeriodInSeconds
                                    ());
                        }
                }
        }
```

```
}
```

Úryvek 6.12: Metoda setGeneralSettings

Metoda setGeneralSettings vytváří objekt třídy CDisplayDepthDlg, díky jejíž metodě DoModal otevírá dialog pro změnu počtu zobrazených hodnot v grafu. Pokud uživatel klikne na tlačítko „OK“, hodnota se uloží do atributu objektu třídy UIClass. Na podobném principu pracují i ostatní dialogy.

Třída UIClass představuje hlavní část modelu celé aplikace podle návrhu MVC. Proto musí být přístupna i ostatním částem programu. Třída CMainFrame vytváří objekt pCore třídy UIClass, který je jí přístupný podobně jako objekt pView. Objekt pCore je samozřejmě také hlavně přístupný třídě CGoIO_MeasureView. Celá aplikace je na tomto objektu závislá a podle něj controller určuje stavy aplikace a view vykresluje. Všechny třídy v programu jsou navzájem pevně provázány, přesto však by měly být podle objektově orientovaného návrhu správně zapouzdřeny. Kód různých metod se tedy může libovolně měnit, a i když sice bude daná změna mít náležitý dopad na celý program, běh programu jako celku na jejich vnitřním kódu závislý není (například poškozená metoda zobrazující dialog ho sice nezobrazí, ale program poběží dále a nezmění se například chování výše zmíněné metody toCycle).

Kapitola 7

Implementace serverové části

Serverovou část aplikace jsem napsal v jazyce PHP s využitím frameworku Nette, jehož základní strukturu tedy aplikace přebírá. Díky tomu je automaticky aplikován model MVC. Protože jsem v klientském programu (viz výše) postupoval agilně, změny jsem do aplikace přidával s jednotlivými měnícími se požadavky a snažil jsem se zachovávat maximální otevřenost, rozhodl jsem se zvolit stejný postup i u serverové části aplikace. Proto rovněž dále rozepíšu pouze základní kostru aplikace a její implementaci.

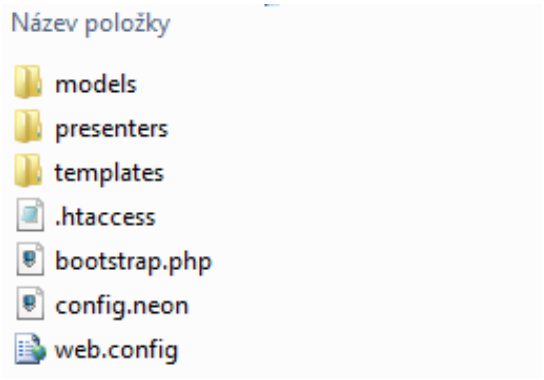
7.1 Základní struktura

Framework Nette nabízí základní kostru, jejímž rozšiřováním lze psát plnohodnotné aplikace, které pak poběží na serveru. Tato kostra přichází s vlastní adresářovou strukturou. Serverová aplikace je v podstatě umístěna pouze v adresářích app a www, které se nachází přímo v kořenovém adresáři. Ostatní adresáře slouží ke správné funkci Nette frameworku.

Adresář www obsahuje kód CSS, knihovny JavaScriptu, obrázky, ikony a další podpůrné materiály pro webovou aplikaci. Důležitými jsou v něm umístěné složky s názvy css a js. Ve složce js je umístěn předpřipravený kód JavaScriptu, který aplikace využívá především pro AJAX (využit pro interaktivní zobrazení dialogu při přidávání nového zařízení do projektu). Serverová aplikace využívá funkce JavaScriptového frameworku jQuery pro podporu AJAX požadavků [13]. Ve složce css je umístěn kód kaskádových stylů pro vlastní webovou stránku. Nejdůležitější je soubor screen.css, který obsahuje základní definice vzhledu webové stránky. Jako výchozí styl písma je použit font Helvetica.

V adresáři app je umístěno jádro vlastní serverové aplikace. Adresář obsahuje složky s modely, presentery a šablonami (vysvětlení viz níže v části presenter).

Kromě souboru .htaccess, určujícím práva přístupu (přímý přístup přes URL je do adresáře app zamítnut kvůli tajnému souboru confing.neon), je



Obrázek 7.1: Adresář app

v tomto adresáři umístěn soubor bootstrap.php. Tento soubor načítá aplikaci jednoduchým voláním *$application->run()*, routy (vzhled URL adresy) a některé služby pro přístup k databázi. Dále se v adresáři nachází soubor config.neon, který obsahuje přístupové údaje k databázi MySQL. Většina kódu je převzata ze základní kostry Nette frameworku.

7.2 Presenter

Nette framework využívá architekturu MVP (model, view, presenter), která je podobná známé architektuře MVC. Pro naši potřebu postačí vědět, že presenter v zásadě odpovídá pojmu controller, a stará se tedy o obsluhu událostí vyvolaných uživatelem, který může k aplikaci přistupovat pomocí webových stránek nebo pomocí API rozhraní v případě, že komunikuje prostřednictvím klientské aplikace.

Ke každé větší akci webové stránky je přiřazen vlastní presenter, kde všechny z nich dědí od základní třídy BasePresenter, která vkládá seznam uživatelových projektů do hlavní šablony (kvůli zobrazení menu projektů). Některé presentery dědí od třídy SecuredPresenter, která zajistí to, že se k určitému presenteru dostane pouze uživatel, který už je přihlášen. Přihlášení se provádí pomocí přesměrování na akci (metodu) presenteru SignPresenter.

Dalším základním presenterem je třída ErrorPresenter, která zobrazuje příslušnou chybovou stránku (s vysvětlením vzniku chyby) podle číselného kódu chyby (404 ...).

Třída ApiPresenter vytváří základní komunikační bránu mezi klientskou a serverovou aplikací. Podle speciálně nadefinovaných rout (tvarů URL) program pozná, kterou metodu (akci) této třídy má zavolat. Názvy metod a jejich parametrů přímo korespondují s požadavky klientské aplikace na server. Pro ukázku je uveden zdrojový kód metody actionProject. Tato metoda podle parametrů jí předaných buď vrátí seznam projektů, nebo vytvoří nový projekt

(tato vlastnost není klientskou aplikací využívána, sama o sobě by neměla v současné koncepci smysl). Zároveň kontroluje, zda má uživatel skutečně právo přistupovat k projektu. Tato kontrola se provádí pomocí třídy User, která pokládá příslušný dotaz databázi.

```
public function actionProject($username, $method, $id,
    $name, $users)
{
        $this->authenticate();
        $user = new User($username);
        // list all
        if ($method === NULL && $id === NULL) {
                $this->reply($user->projects);
        // list by id
        } elseif ($id !== NULL) {
                if (!$user->hasAccessToProject($id)) {
                        $this->error(401, "User ‘
                            $username‘ not allowed to
                            access project #$id.", 3);
                }
                $project = new Project($id);
                $this->reply($project->toArray($user->id
                    ));
        // create new
        } elseif ($method == 'create') {
                $this->verifyArgs(array('name'));
                $project = Project::create($name, $user
                    ->id, $users);
                $this->reply(array('id' => $project['id'
                    ]));
        }
}
```

Úryvek 7.1: Metoda actionProject

Třída HomepagePresenter se stará o všechno, co uživatel může provádět po přihlášení. Vytváří komponenty (formuláře) pro nastavení upozornění překročení hraničních hodnot, změnu údajů uživatele, vytvoření projektu, připojení veřejného projektu a odstranění projektu. Tyto komponenty se vytváří na příkaz handle (doslova „ovladač“) metod, což jsou metody podobné akcím, které ale na rozdíl od nich nemají vlastní šablony (viz část view níže).

V tomto případě je pro všechny handly společná šablona akce default. Formuláře se vytvářejí pomocí jednoduché továrničky Nette frameworku (stačí pouze říci, jaká políčka ve formuláři požadujeme). Metoda, která se pak stará o zpracování formuláře, zpravidla zavolá metodu nějaké třídy modelu, která data uloží do databáze.

Třída ProjectPresenter se stará o všechno, co uživatel vykoná na stránce s otevřeným projektem. Spravuje přidání, odebrání, změnu a reset předpisů zařízení, dále pak spravuje handle stáhnutí souborů CSV, ve kterých jsou uložena naměřená data. Šablona výchozí akce se také stará o zobrazení grafu od firmy Google (pomocí JavaScriptového kódu). Pro zajímavost je zde uveden kód jednoduchého handlu handleSendCSV posílajícího uživateli soubor CSV, který byl nejdříve vytvořený modelem. Handle se také stará o vyprázdnění složky se soubory CSV, které již nejsou potřeba.

```
public function handleSendCSV($project_id)
{
        if ($project_id > 0)
        {
                $files = glob("../data/*.CSV"); // SAFE
                    mode neccessary here! (due to other
                    users)
                foreach($files as $file) unlink('safe://
                    ' . $file); // delete in safe (CSV)
                $project = new Project($project_id);
                $filename = $project->getDataCSV();
                $this->sendResponse(new Nette\
                    Application\Responses\FileResponse(
                    $filename));
        }
}
```

Úryvek 7.2: Metoda handleSendCSV

Třída RegistrationPresenter se stará pouze o registraci uživatele do systému. Vytváří jednoduchý formulář a v metodě, která se stará o jeho zpracování, pak pomocí metody create třídy User vloží údaje do databáze.

SignPresenter je další jednoduchá třída, která se tentokrát stará o přihlášení uživatele. Opět vytvoří potřebný formulář a metoda, která se stará o jeho zpracování, porovná pomocí modelové třídy údaje s databází a v případě úspěchu uživatele přihlásí pomocí funkce Nette frameworku *$this->getUser()->login()*. Třída kromě akce in obsahuje také akci out, která uživatele odhlásí ze systému. Níže je uveden příklad vytvoření a zpracování formuláře akce in (pro více informací o formulářích navštivte [6]).

```
/**
* Sign in form component factory.
* @return NA\UI\Form
*/
protected function createComponentSignInForm()
{
        $form = new NA\UI\Form;
```

```
        if ($this->lang == 'cs')
        {
                $form->addText('username', 'Uzivatelske␣
                    jmeno:')
                        ->setRequired('Prosim␣zadejte␣
                            uzivatelske␣jmeno.');

                $form->addPassword('password', 'Heslo:')
                        ->setRequired('Prosim␣zadejte␣
                            heslo.');

                $form->addSubmit('send', 'Prihlasit␣se')
                    ;
        }
        else
        {
                $form->addText('username', 'Username:')
                        ->setRequired('Please␣fill␣in␣
                            the␣username.');

                $form->addPassword('password', 'Password
                    :')
                        ->setRequired('Please␣fill␣in␣
                            the␣password.');

                $form->addSubmit('send', 'Sign␣in');
        }
        $form->onSubmit[] = callback($this, '
            signInFormSubmitted');
        return $form;
}
public function signInFormSubmitted($form)
{
        try {
                $values = $form->getValues();
                $this->getUser()->setExpiration(
                    SESSION_MIN_GUARD, TRUE);
                $this->getUser()->login($values->
                    username, $values->password);
                $this->redirect('Homepage:');
        } catch (NS\AuthenticationException $e) {
                $form->addError($e->getMessage());
                $this->invalidateControl('errors');
        }
```

```
}
```

Úryvek 7.3: Zpracování formuláře akce in presenteru SignPresenter

7.3 View

Menší akce webové stránky se spravují užitím nové metody presenteru začínající názvem action. Ke každé akci (včetně výchozí akce default) je přiřazena jedna šablona s HTML kódem a Latte příkazy. Každá šablona má možnost využívat filtr Nette frameworku s názvem Latte. Pomocí speciálních příkazů přímo v šabloně lze vnořovat do šablony programový PHP kód a podmínit vykreslení určitých částí.

Základní šablona, do které se pak vnořují ostatní šablony podle aktuální akce, se jmenuje @layout.latte. Tato šablona načítá JavaScriptový kód, kaskádové styly a určuje celkovou strukturu stránky pomocí kódu HTML (obsahuje také jednotlivé hypertextové odkazy na jednotlivé presentery programu, tj. z uživatelského hlediska hlavní menu).

Protože se jedná jen o jednoduchý HTML kód, nemá dle mého názoru cenu zde uvádět a popisovat obsah šablon. Každá z akcí presenteru, které byly zmíněny výše v části presenter, má svou vlastní šablonu.

7.4 Model

Modelová část u serverové aplikace není příliš komplexní. Program obsahuje a přímo využívá několik tříd modelu – Authenticator, Device, MailSend, Project a User (všechny dědí od základní třídy BaseModel vracející služby pro přístup k databázi).

Třída User obsahuje základní metody pro práci s databázovou tabulkou user. Pomocí této třídy lze jednoduše upravovat uživatelská data nebo získat projekty či předpisy zařízení svázané s uživatelem. S třídou User úzce souvisí třída Authenticator, která ověřuje platnost uživatelských dat při přihlášení a obsahuje metodu pro zašifrování hesla pomocí tajného hashe. Heslo se v tomto formátu přenáší pomocí URL při komunikaci klienta se serverem a je také takto uloženo v databázi MySQL.

Další třída s názvem MailSend využívá pouze vestavěnou třídu Nette frameworku Message k odesílání jednoduchého e-mailu s upozorněním na překročení hraniční hodnoty. Akce této třídy jsou řízeny objektem třídy Device.

Posledními třídami jsou třídy Project a Device. Jak už název napovídá, třída Project se stará o projekty a třída Device o předpisy zařízení. Obsahují základní metody pro vytváření, editaci a odstraňování projektů, popřípadě předpisů zařízení. Třída Project také obsahuje metodu getDataCSV vytvářející soubor CSV k danému projektu.

Třída Device obsahuje metodu setData, která podle komunikace s API uloží naměřenou hodnotu do databáze (zároveň kontroluje, jestli nebyly překročeny nějaké hraniční hodnoty; pokud ano, pak je uživateli podle jeho nastavení zasláno upozornění).

7.5 Datová část

Jednotlivé modelové třídy přímo využívají přístup k databázi MySQL a jejím datům, která jsou v databázi uložena v následujících tabulkách:

1. data (jednotlivé naměřené hodnoty představované ID předpisu zařízení, hodnotou a časem)
2. device (předpis zařízení)
3. project (projekt, kde nejdůležitější je jeho název)
4. mail (nastavení odesílaných upozornění na překročení hraničních hodnot)
5. user (uživatel)
6. project_user (přístup uživatele k danému projektu)

SQL kód pro vytvoření tabulek v databázi je dodáván spolu se zdrojovým kódem serverové části aplikace.

Kapitola 8

Instalace aplikace

8.1 Serverová část

Pro využívání serverové části aplikace má uživatel dvě možnosti. Buď může využít již mnou nainstalovanou verzi, která je pro všechny zdarma přístupná pomocí webového prohlížeče na doméně http://www.flowersafeguard.com a do které se stačí pouze zaregistrovat pomocí odkazu, který je přístupný na hlavní stránce. Druhou možností je využít zdrojové kódy, které jsou součástí této práce, a provést ruční instalaci na příslušném vlastním serveru nebo s pomocí hostingu. Součástí zdrojových kódů je soubor s názvem README.TXT, který obsahuje veškeré pokyny v angličtině. Základní kroky instalace jsou v zásadě pouze tři:

1. Uživatel nahraje zdrojové kódy do adresáře na serveru, kde jsou uloženy soubory přístupné pomocí domény.
2. Uživatel upraví parametry v souboru app/config.neon tak, aby aplikace mohla přistupovat k databázi.
3. Uživatel spustí v databázi SQL kód pro vytvoření tabulek, který je k dispozici v souboru app/sql.sql.

8.2 Klientská část

Pro instalaci klientského programu do konkrétního počítače stačí pouze rozbalit instalační balíček do jedné složky a spustit program setup.exe. Po automatickém otevření instalačního okna uživatel napřed nainstaluje ovladače pomocí tlačítka „Instalace ovladačů“, načež spustí vlastní instalaci programu kliknutím na tlačítko „Instalace programu Flower Safeguard“. Ihned poté se otevře známý instalátor pro Windows, který uživatele provede výběrem složky pro instalaci a vlastním procesem instalace, načež na ploše vytvoří zástupce dvou programů, konkrétně „Flower Safeguard“, což je vlastní klientská část

aplikace, a malého prográmku s názvem „StyleMaker Easy“, který slouží pro jednoduchou tvorbu grafických stylů pro hlavní aplikaci a jehož popis není součástí této práce. Uživatel pak klientský program spustí dvojím kliknutím na zástupce „Flower Safeguard“.

Kapitola 9

Praktická měření

9.1 Měření teploty čaje

První malé měření si dovolím použít především pro ilustraci toho, jak programu neznalý uživatel může jednoduše využít základní funkce aplikace. Pro účely měření je předpokládán již založený uživatelský účet na serveru a nainstalován klientský program na počítači. Pro popis detailnějšího měření viz další podkapitolu s názvem „Měření průběhu počasí“.

Na počátku jsem se rozhodl začít měřit teplotu horkého čaje, abych poznal, jestli se čaj již ochladil na správnou teplotu (aby se dal bez problémů pít). V tu chvíli mě napadlo využít možnost e-mailového upozornění, které by ve správný čas dorazilo do složky na mém chytrém telefonu, který by následně přehrál zvukové upozornění.

Přihlásil jsem se tedy pomocí svého účtu do serverové části aplikace, načež jsem pro toto měření založil nový projekt, kde jsem jako nový předpis zařízení přidal teploměr. U tohoto zařízení jsem nastavil minimální teplotu 50 stupňů Celsia a maximální 100 stupňů, neboť jsem si myslel, že teplota 50 stupňů je pro začátek pití ideální. K tomuto předpisu zařízení jsem pak na hlavní stránce serverové aplikace přidal upozornění v rámci projektu, abych byl v případě překročení hraničních hodnot upozorněn zprávou na e-mail.

Když jsem byl spokojen s nastavením na serveru, připojil jsem fyzické zařízení (teploměr) k počítači, vložil konec teploměru do právě připraveného horkého čaje a spustil klientskou aplikaci pomocí zástupce na ploše s názvem „Flower Safeguard“. Po spuštění programu jsem vyplnil své uživatelské jméno a heslo. Adresu serveru jsem vyplňovat nemusel, neboť automaticky je napoprvé v programu vyplněna URL adresa http://www.flowersafeguard.com, která přímo odkazuje na tu verzi serverové aplikace, kterou jsem využíval. Po vyplnění údajů se načetl seznam projektů v podobě spritů po pravé straně okna aplikace. Ihned jsem zvolil správný projekt, načež jsem přetáhl obrázek teploměru do správného předpisu zařízení v rámci tohoto projektu. Program sám pak po asociaci otevřel dialog, kde jsem jako jednotku zvolil stupně Celsia.

Jakmile bylo vše připraveno, pomocí klávesy „M“ jsem spustil vlastní měření.

Pak stačilo již jen čekat. Na začátku měření LED dioda na teploměru svítila zeleně, což znamená, že zatím bylo vše „v pořádku“. Nebylo tedy nutné posílat upozornění, že teplota čaje překročila hraniční hodnotu. Nicméně po asi půl hodině se LED dioda rozsvítila červeně a do mé e-mailové schránky přišel nový e-mail oznamující, že zařízení naměřilo hodnotu nižší, než je nastavená minimální hodnota (v tomto případě 50 stupňů Celsia). Můj chytrý telefon přehrál zvuk a já věděl, že je načase se pustit do pití čaje.

Ihned poté jsem pozastavil měření, znovu otevřel internetový prohlížeč se serverovou aplikací a nechal si zobrazit detail příslušného projektu. Na úplném počátku určitou dobu trvalo, než se teploměr ohřál na teplotu čaje. Pak už byl pokles teploty čaje pravidelný až do doby, kdy jsem měření zastavil.

9.2 Měření průběhu počasí

Nyní následuje popis detailnějšího měření, které mohou s drobnými úpravami využít učitelé spolu se svými žáky v rámci kreativní výuky fyziky. Jedná se o měření počasí v průběhu slunečného dne, kdy se měří teplota, tlak a množství světla. K tomu jsem přidal měření teploty uvnitř domu, kdy jsem využil nejen další měřící zařízení, ale zároveň i další počítač (a tedy klientský program), který odesílal data do stejného projektu na serveru. Všechny veličiny byly pochopitelně měřeny pomocí měřících senzorů firmy Vernier (dva teploměry GO-TEMP, jeden barometr BAR-BTA a jeden luxmetr LS-BTA).

Přípravu na vlastní měření jsem zahájil brzy ráno. Napřed jsem se rozhodl nastavit měření na tom počítači, který by měřil počasí venku. Všechny sensory jsem pomocí USB připojil k notebooku a umístil je tak, aby měřily stav počasí za oknem.

Všechny tyto tři senzory tedy bude obsluhovat jeden počítač, který z nich bude sbírat data a odesílat je na server. Po správném zapojení jsem přešel dále do domu a k dalšímu notebooku připojil dodatečný teploměr pro měření teploty uvnitř. Tím byla zahájena vlastní příprava zařízení a počítačů. Ihned poté jsem se pod svým nově zaregistrovaným jménem na serveru, konkrétně v online verzi Flower Safeguard, přihlásil a na hlavní stránce založil nový projekt s názvem „Počasí“. Jakmile jsem si otevřel jeho detail, přidal jsem k němu čtyři předpisy zařízení, které odpovídaly jednotlivým přístrojům pro měření. Jelikož jsem se rozhodl nevyužít možnosti e-mailového upozornění ani možnosti upozornění na překročení hraniční hodnoty pomocí LED diody, nastavil jsem u všech předpisů maximální hodnotu jednoduše na 1000 jednotek.

Potom jsem na obou počítačích otevřel klientskou aplikaci stejně jako v případě měření teploty čaje. V programu jsem se pak přihlásil a dvojím kliknutím myši na název projektu jsem ho otevřel. Poté jsem ke každému předpisu zařízení přiřadil přístroj a zvolil pro každý senzor jednotku, ve které bude daný přístroj měřit. Poté jsem pomocí hlavního menu nastavil periodu měření pro

Obrázek 9.1: Měření počasí za oknem s ochrannou sítí

Obrázek 9.2: Přidání předpisů zařízení v projektu na serveru

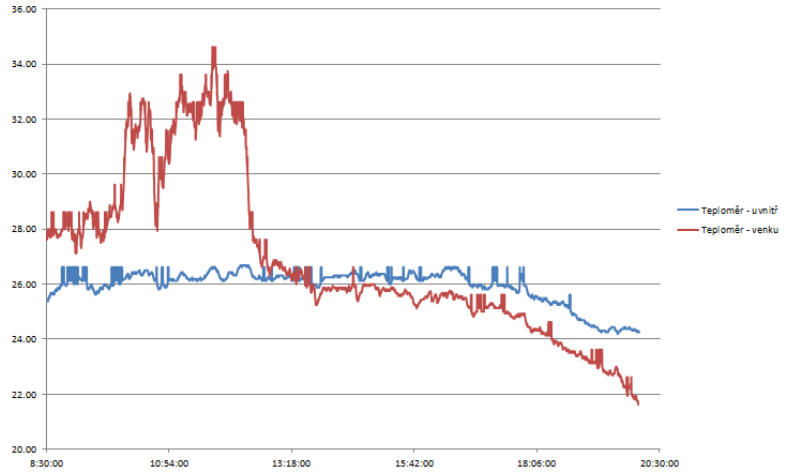
každé zařízení na 2000ms, neboť měření v průběhu celého dne skutečně nevyžaduje častější zaznamenávání hodnot. Když bylo vše připraveno, na obou počítačích jsem spustil vlastní měření pomocí klávesy „M“.

Měření jsem zahájil přibližně v 8:30 a ukončil zhruba ve 20:00. Během procesu měření nebyl registrován žádný výpadek ani přerušení. Internetové připojení fungovalo stále a bez problémů. Nicméně program sám naštěstí měření nepřeruší, i když samotné internetové spojení selže, jelikož se data mezitím ukládají do souboru. Jakmile se pak opět spojení obnoví, data se okamžitě odešlou na server.

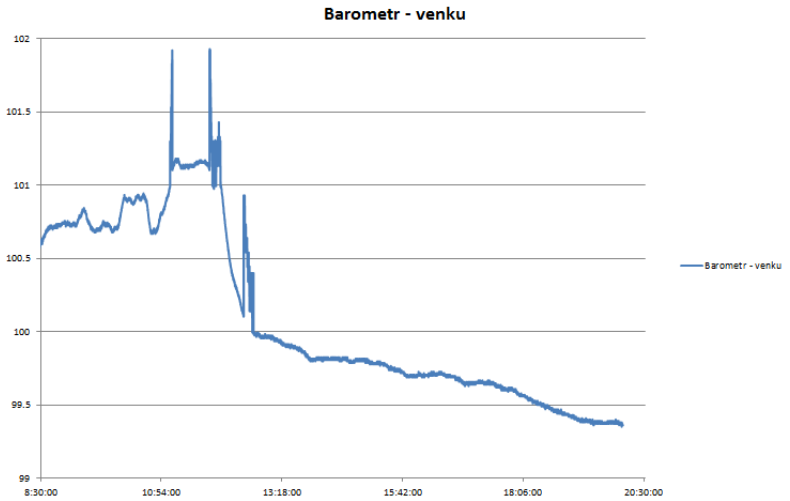
Když jsem kolem osmé hodiny večerní měření zastavil (opět pomocí klávesy „M“), odpojil jsem jednotlivá zařízení od obou počítačů a rozhodl se, že data odeslaná na server analyzuji. Přihlásil jsem se opět do serverové aplikace pomocí webového prohlížeče a otevřel projekt s názvem „Počasí“. Naměřené hodnoty ze všech senzorů se společně zobrazovaly v projektovém grafu, nicméně jsem se rozhodl, že bude lepší si jednotlivé naměřené hodnoty vyexportovat ve formátu CSV a zobrazit pomocí tabulkového procesoru. V projektu jsem tedy vybral možnost „Export (CSV)“ a soubor s daty jsem uložil na pracovní plochu počítače.

Rozhodl jsem se grafy zpracovat v programu Microsoft Excel 2010. Vytvořil jsem nový dokument a vložil do něj externí data z vytvořeného souboru CSV na pracovní ploše. Zde se mohou objevit jisté komplikace. Program Microsoft Excel po přímém otevření souboru CSV nenabízí možnost změnit kódování, proto mohou vznikat problémy s diakritikou. Tomu lze předejít vytvořením nového dokumentu XLS a načtením dat ze souboru CSV pomocí možnosti Data -> Načíst externí data -> Z textu (v nabídce je důležité zvolit kódování UTF-8 a jako oddělovač použít středník ‘;’). Další problém může nastat se systémovými oddělovači desetinných míst, tj. desetinnou čárkou. Hodnoty mají samozřejmě desetinná čísla oddělená desetinnou tečkou. Opět zde existuje jednoduchý postup. Stačí otevřít nabídku Rychlý přístup (vedle tlačítek Vzít zpět a Provést znovu) -> Další příkazy... -> Upřesnit. Zde stačí do políčka „Oddělovač desetinných míst:“ vložit tečku a kliknout na tlačítko „OK“. Nastavení oddělovače desetinných míst je nutné provést před načtením dat ze souboru CSV. Po načtení dat jsem v dokumentu vytvořil další sloupeček, ve kterém jsem čas uložený pomocí „UNIX timestamp * 1000“ převedl do časového formátu Excelu. Dále jsem vytvořil tři grafy (pro oba teploměry, luxmetr a barometr) a přesunul jsem je na samostatné listy. Tyto grafy jsou mnohem názornější a můžeme už z nich vyčíst různé informace pro důsledné sledování počasí.

Co se týče rozdílu mezi teploměrem umístěným uvnitř budovy a teploměrem umístěným venku, je vidět, že teplota uvnitř budovy je mnohem stabilnější (což je pro budovu dobrá vizitka). Teplota venku se v této době pohybovala v intervalu přibližně 22°C až 34°C. Přibližně v půl sedmé v podvečer se již začalo pomalu stmívat, což ukazuje graf luxmetru, který však jinak není zajímavý. Tlak začal pravidelně a postupně klesat přibližně v jednu hodinu odpoledne.



Obrázek 9.3: Graf naměřených teplot v závislosti na čase



Obrázek 9.4: Graf naměřeného tlaku v závislosti na čase

Z grafu je vidět skutečnost, že tlak značně kolísal mezi jedenáctou hodinou dopolední a první hodinou odpolední, z čehož vyplývá, že tento čas během teplého slunečného dne může být pro osoby se zvýšenou citlivostí či zdravotními problémy značně nepříjemný, zvláště pokud vykonávají náročnější fyzickou aktivitu.

Pokud by měření vykonávali například žáci u sebe doma v rámci domácího úkolu, mohly by se vyjevit zajímavé skutečnosti týkající se stability teploty u nich doma či tlaku a množství přirozeného osvětlení v jejich oblasti. Je jen na učitelích, jak poznatky uvedené v této práci rozvinou a jestli žákům zpříjemní výuku fyziky praktickými ukázkami.

Kapitola 10

Výsledná zkouška a zhodnocení

Výslednou zkoušku návrhu aplikace, který je součástí této práce, jsem provedl společně s panem magistrem Jaroslavem Reichelem (učitel) na Střední škole průmyslové techniky v Panské, kdy jsme se rozhodli aplikaci prezentovat před třídou žáků technického lycea, pro něž je fyzika jedním z hlavních předmětů a z nichž většina skládá z fyziky školní maturitní zkoušku. Napřed jsem učiteli ukázal jednotlivé funkce podle toho, jaké jsme si stanovili prvotní požadavky (viz sběr a analýza požadavků) na vlastní aplikaci. Žáci projevovali o aplikaci zájem, především se jim líbila názorná ukázka měření teploty v místnosti, kde se po překročení spodní hodnoty odeslalo na mobil e-mailové upozornění. Nicméně žáci sami hlavní myšlenku vývoje aplikace (kreativní výuka fyziky) vnímali okrajově a neměli jasné představy, jak by se aplikace mohla v budoucnu využít při laboratorních pracích. Sám si uvědomuji, že použití aplikace je přímo úměrné nápaditosti a ochotě učitelů se vůbec do kreativní výuky pouštět. Během prezentace jsem ještě předvedl zapojení teploměru a sonaru, který snímal vzdálenost od nejbližšího předmětu. I když učitele již nenapadaly žádné další funkční požadavky na serverovou aplikaci či klientský program, dospěli jsme k závěru, že bude nutné především upravit grafické vzezření aplikace a počkat, až se trend kreativní výuky více uchytí.

I když jsem zprvu očekával větší nadšení a přímé zapojení aplikace do výuky, což se v této fázi zatím bohužel nestalo, i tak jsem byl rád za zpětnou odezvu a všeobecnou pochvalu od učitele i ostatních pedagogů. Učiteli samotnému se především líbilo splnění všech funkčních požadavků a celková funkčnost aplikace, která během celé doby prezentace pracovala v pořádku, a kde nedocházelo ke ztrátě dat během přenosu z klientského programu na server.

Z prezentace ve třídě před žáky tedy vyplynuly tři hlavní body:

1. „Zprofesionalizovat“ vzhled webové aplikace i klientského programu.

2. Vytvořit praktické ukázky měření, navrhnout schéma laboratorních prací s využitím aplikace a motivovat učitele fyziky k jejímu využívání. Zde

se jedná o velice abstraktní otázku, kterou jsme s učitelem dále nespecifikovali, jelikož ani jeden z nás neměl zatím ponětí, jak by se mohli ostatní učitelé a školy motivovat.

3. Upravit aplikaci tak, aby bylo možné vykonávat všechny funkce jak na serveru, tak i v klientském programu. To se týká především vytváření projektů a úpravy hraničních hodnot pro měření. Tento dodatečný požadavek však podle učitele není klíčový.

Po skončení prezentace jsem učiteli i žákům poděkoval za zpětnou vazbu. Všechny tři výše zmíněné body jsem vzal v potaz a rozhodl se, že na nich v budoucnu při další příležitosti zapracuji.

Kapitola 11

Možnosti rozšíření

Nyní se zaměřím na možnosti rozšíření stávající aplikace. Především je nutné ji pořádně vyzkoušet na školách a pro případné rozšíření a úpravy získat zpětnou vazbu, jelikož školské prostředí má vlastní nároky, které klasický programátor s výpomocí jen několika málo učitelů může pouze těžko odhadnout. Případně bude nutné pozměnit grafický návrh aplikace a texty chybových hlášek, které se žákům zobrazují, jelikož mezi jednotlivými typy škol je opravdu veliký rozdíl a každá může mít trochu jiné představy a potřeby, jak by aplikace měla fungovat. Nicméně základní návrh aplikace jsem (viz výše) dokončil a učitelům z mé bývalé střední školy (Střední průmyslová škola sdělovací techniky Panská, obor Technické lyceum) se líbila. Avšak stejně jako se vším, co se týká uplatňování nejnovějších technologií, je důležité být trpělivý a počkat, až se podobná snažení více uchytí jak mezi učiteli, tak mezi žáky. Bylo by pěkné rovněž nalézt uplatnění jak data sesbíraná ze škol (například údaje o počasí) využít v rámci občanské vědy pro vědecké účely.

Pokud by se aplikace více uchytila, bylo by samozřejmě žádoucí rozšířit klientský program na více platforem a popřípadě využít i chytré telefony a jejich vestavěné čipy, například teploměr nebo zjišťování polohy pomocí technologie GPS, pro měření těch nejzákladnějších veličin. Zde se opět dostáváme k důležitosti učitelů, jejichž zkušenosti by mohla případná společnost, která by měla zájem o tvorbu nové nebo rozšiřování stávající aplikace, využít ve své tvorbě a umožnit tak aplikaci vhodně navrhnout pro konkrétní řešitelné úkoly v rámci výuky. Mezi takové patří například zjišťování průběhu denní teploty v místě bydliště žáka, sledování změny oblohy v průběhu dne (pomocí fotoaparátu) a mnohé další, které by jistě kreativní učitelé dokázali pro své žáky vymyslet. Současný svět nabízí plno rozmanitých technologií a možností jejich využití, což je potřeba brát v budoucí tvorbě v potaz.

Existuje tedy nespočet možných rozšíření, jejichž důležitost musí učitelé ověřit přímo ve třídách spolu se žáky. Aplikace by měla mít příjemné grafické rozhraní, jednoduché ovládání a skutečně studentům a učitelům pomoci ušetřit cenný čas při laboratorních pracích. Pevně doufám, že současný sice

pomalý, ale zároveň stálý vývoj v oblasti kreativní výuky a občanské vědy přiláká zájem autorit a (neziskových) společností, které by měly zájem spolupracovat s učiteli a vymyslet, navrhnout, implementovat a otestovat další funkční aplikace, které by více zpříjemnily, oživily a především učinily výuku nejen fyziky více efektivní. Současně s tím by se mohla využít možnost povzbuzení zájmu žáků o přírodní vědy a technické obory.

Závěr

Zájem o kreativní výuku fyziky, tj. pro žáky zábavnou výuku s využitím moderních technologií a nápadů, se začíná na školách projevovat v podobě stále častějšího využívání nejrůznějších měřících přístrojů a programů pro uchovávání a analýzu naměřených dat. Učitelé se snaží přijít na to, jak žákům výuku zpříjemnit a celkově ji zefektivnit. Aplikace, která byla prezentována v této práci, vznikla v důsledku nápadu shromažďovat a zobrazovat naměřená data všech žáků na jednom místě, aby se ušetřil čas při laboratorních pracích, který by se mohl využít efektivněji například různorodostí naměřených veličin pomocí různých přístrojů, načež by následovala společná analýza. Navržená aplikace splnila všechny základní funkční požadavky a počítala i s tím, že školy disponují velmi omezenými finančními prostředky, takže vhodnost využití již navržených a neplacených programů byla vřele vítána. Na názorných ukázkách jsem v této práci poukázal na praktické využití aplikace například při měření průběhu počasí v domově žáků, kde by se dala sesbírat zajímavá data. Je jen na učitelích, jaké další domácí úkoly a laboratorní práce pro své žáky vymyslí. Bylo by rovněž pěkné v budoucnu spatřit napojení kreativní výuky (a tedy i této aplikace) na sbírání cenných dat při občanské vědě, což leží v rukou vědců, kteří se možností občanské vědy v současné době zabývají.

V souladu s požadavky umí aplikace po důkladném otestování bezchybně odesílat naměřená data z jednotlivých měřících přístrojů firmy Vernier na server, kde je následně uchovává v databázi. Data se dají v rámci každého projektu (měření) zobrazit v podobě grafu a stáhnout ve formátu CSV pro následnou dodatečnou analýzu. Uživatelé se do aplikace přihlašují pod svým uživatelským jménem a heslem, které je stejné jak pro serverovou část aplikace, tak i pro klientský program. Navíc aplikace umí nastavit hraniční hodnoty měření, jejichž překročení je v klientské aplikaci představováno změnou stavu LED diody na konkrétním přístroji, zatímco v serverové aplikaci je po řádném nastavení odesláno e-mailové upozornění na uživatelovu e-mailovou adresu. Všechny tyto funkce byly otestovány a během dokončení této práce bezchybně fungovaly.

I když aplikace a její potenciál vyvolal mezi oslovenými učiteli a žáky značné nadšení, bohužel se v současné době nepodařilo učitele do praktického zapojení aplikace do výuky motivovat. Nezbývá než sledovat trendy ve školství a aplikaci přizpůsobit a zjednodušit tak, aby pro učitele její používání mělo výraznou přidanou hodnotu. Toho lze po rozhovoru s učiteli dosáhnout „zprofesionalizováním“ a zlepšením vzhledu klientské i serverové části aplikace, vytvořením praktických ukázek laboratorních prací a zjednodušením aplikace tak, aby bylo možné provádět stejné úkony jak na serveru, tak v případě klientské aplikace. Opravdu je důležité na učitele nahlížet jako na náročnější zákazníky, pro jejichž potřebu je software vyvíjen. Jsem si vědom nedostatků vyplývajících z neprofesionálního vzhledu aplikace a nenázornosti, což bohužel při značném objemu práce na tvorbě aplikace ustoupilo do pozadí. Nicméně bych rád na těchto a dalších nedostatcích v budoucnu zapracoval.

Dalších teoretických možností rozšíření aplikace je pochopitelně nespočet a vše záleží na nápaditosti učitelů a ostatních lidí, kteří se kreativní výukou zabývají. Důležité je vše při tvorbě aplikace konzultovat s učiteli fyziky a sbírat jejich nápady, které pak mohou vést například k rozšíření aplikace na mobilní a tabletové platformy, kde by se využívaly vestavěné měřící čipy, a školy by tak mohly ušetřit finance, které by jinak využily pro nákup příslušných měřících přístrojů. Kreativní výuka fyziky by dle mého názoru měla jít ruku v ruce s nejmodernějšími a hlavně obecně využívanými technologiemi, aby žáci mohli na fyziku nahlížet jako na moderní vědu, která se stále vyvíjí a má jim co nabídnout, i když oni sami neaspirují na to, aby se stali profesionálními vědci nebo učiteli. Změny ve školském prostředí probíhají sice značně pomalu, nicméně názornost budoucích rozšíření této nebo jiných podobných aplikací spolu s ochotou učitelů má dle mého názoru potenciál výuku značně zefektivnit a zpříjemnit jak pro učitele, tak především pro žáky. Důležité je dle mého názoru také zapojení třetích stran, které přístroje pro měření a praktické aplikace školám nabízí.

Závěrem bych chtěl objektivně říci, že tato práce splnila své zadání, neboť základní aplikace pro kreativní výuku fyziky funguje a je připravena k okamžitému využití a rozšiřování. Vše od počátku návrhu, finanční náročnosti až po výhled do budoucna jsem v rámci svých možností zanalyzoval a v této práci popsal. Uvědomuji si, že značný časový tlak a složitost aplikace odsunuly do pozadí jiné důležité věci včetně grafického návrhu a praktických ukázek, nicméně, jak už jsem zmínil, jsem ochotný se na vývoji aplikace dále podílet a poskytovat cenné rady učitelům i třetím stranám, které mají zájem o podporu názorné výuky fyziky na školách.

Literatura

[1] JEPSON, P.; LADLE, R. J.: *Nature apps: Waiting for the revolution [online]*. 12. 10. 2015 [vid. 19-11-2015]. Dostupné z: `http://www.ncbi.nlm.nih.gov/pubmed/26458392`

[2] VERNIER SOFTWARE & TECHNOLOGY, LLC: *Software Development Kits [online]*. 2015 [vid. 19-11-2015]. Dostupné z: `http://www.vernier.com/downloads/software-development-kits/`

[3] MLEJNEK, J.: *11. přednáška Softwarové inženýrství [přednáška]*. [vid. 20-04-2016]. Fakulta informačních technologií, České vysoké učení technické v Praze, Praha.

[4] KANISOVÁ, H.; MÜLLER, M.: *UML srozumitelně*. Brno: Computer Press, druhé vydání, 2007, ISBN 80-251-1083-4.

[5] KRÁTKÝ, T.: *3. přednáška Softwarové inženýrství 2 [přednáška]*. 20. 10. 2015 18:00. Ballingův sál, Národní technická knihovna, Praha.

[6] NETTE FOUNDATION: *Nette Framework [online]*. 2008- [vid. 19-11-2015]. Dostupné z: `https://nette.org/`

[7] VRÁNA, J.: *1001 tipů a triků pro PHP*. Brno: Computer Press, první vydání, 2010, ISBN 978-80-251-2940-1.

[8] MICROSOFT: *Framework (MFC) [online]*. 2015 [vid. 19-11-2015]. Dostupné z: `https://msdn.microsoft.com/cs-cz/library/k9kb0kba.aspx`

[9] PRATA, S.: *Mistrovství v C++*. Brno: Computer Press, třetí vydání, 2007, ISBN 978-80-251-1749-1.

[10] CURL AND LIBCURL: *CURL and LIBCURL [online]*. [vid. 17-03-2016]. Dostupné z: `https://curl.haxx.se/`

[11] PLATY.CZ: *Průměrný hrubý měsíční plat programátora PHP [online]*. [vid. 03-03-2016]. Dostupné z: `http://www.platy.cz/platy/informacni-technologie/programator-php`

[12] MORRISON, M.: *Naučte se programovat počítačové hry za 24 hodin*. Brno: Computer Press, první vydání, 2004, ISBN 80-251-0371-4.

[13] THE JQUERY FOUNDATION: *jQuery [online]*. [vid. 21-04-2016]. Dostupné z: `https://jquery.com/`

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application programming interface

CSS Cascading style sheets

CSV Comma-separated values

GPS Global positioning system

GUI Graphical user interface

HTML Hypertext markup language

LED Light-emitting diode

MFC Microsoft foundation classes

MVC Model-view-controller

MVP Model–view–presenter

PHP Hypertext preprocessor

SDK Software development kit

SQL Structured query language

URL Uniform resource locator

USB Universal serial bus

UTF UCS transformation format

XML Extensible markup language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `client` ..... adresář s instalačním balíčkem klientské aplikace
- `src` ..... zdrojové kódy
  - `impl_client` ..... zdrojové kódy implementace klientské aplikace
  - `impl_server` ..... zdrojové kódy implementace serverové aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `assignment.pdf` ..... zadání práce ve formátu PDF