STŘEDNÍ PRŮMYSLOVÁ ŠKOLA SDĚLOVACÍ TECHNIKY
110 00 Praha 1, Panská 856/3,
☏ 221 002 111, www.panska.cz, sekretariat@panska.cz

ABSOLVENTSKÝ PROJEKT

Dlouhodobá měření

Autor: **Přemysl Černý**

Studijní obor: **78-42-M/01 Technické lyceum**

Školní rok: **2012/2013** Kód třídy: **09M**

ANOTACE:

Flower Safeguard je komplexní software, který umožňuje uživateli pomocí měřících senzorů firmy Vernier sledovat např. teplotu ovzduší pokoje, změny pohybu v pokoji, atd. Naměřená data jsou okamžitě odesílána na server, kde se kontrolují a případně se okamžitě odesílají definované zprávy na uživatelský e-mail. Aplikace může přijímat data od více zařízení, může jí spravovat několik uživatelů, … Její ovládání je intuitivní a je vhodné i pro méně zkušené uživatele.

ANNOTATION:

Flower Safeguard is a comprehensive software application which enables the user, using measuring devices of Vernier company, to watch f. e. a temperature of the air in the room, change of moving objects in the room etc. The measured data are immediately sent to the server where they are checked, and defined messages are sent to the user's e-mail. The application can receive data from several devices in the same time with a possibility of creation more users on the server, … Its control is convenient even for lesser experienced users.

ÚVOD

STRUČNÝ POPIS

Flower Safeguard je komplexní software, skládající se z klientské části (vlastní program) a serverové části, který umožňuje uživateli pomocí měřících senzorů firmy Vernier sledovat, např. teplotu ovzduší pokoje, změny pohybu v pokoji (bezpečnostní využití), atd. Výhoda tkví v okamžitém odesílání dat na server, kde se kontrolují a případně se okamžitě odesílají definované zprávy na uživatelský e-mail.

Tím však možnosti aplikace nekončí, na server se mohou data odesílat z několika různých zařízení i s možností vytvoření více uživatelů na serveru. Pak následují doplňkové možnosti, jako lokální měření, změna grafického stylu aplikace včetně vytvoření vlastního, apod. Program je navrhnut velmi intuitivně včetně textu případných chybových hlášek, a proto je tedy vhodný i pro běžné a méně zkušené uživatele.

Vývoj započal nápadem firmy Edufor, s. r. o. distribující v České republice měřící přístroje firmy Vernier a Mgr. Jaroslava Reichla. Firma hledala program, který by jim umožňoval sledování průběhu měření z různých senzorů firmy Vernier, aniž by obsluha musela být trvale fyzicky přítomna u daného počítače, který měl měření zaznamenávat. Po počátečních konzultacích jsem se nabídl, že se tento nápad pokusím zrealizovat, ačkoli jsem věděl jen velmi málo o měřících senzorech firmy Vernier. Zato jsem však již delší dobu psal programy a aplikace v C++ s využitím WinAPI, což byl také jeden z důvodů, proč jsem se tohoto úkolu ujal.

Realizací tohoto nápadu je přímo předmět této práce, program Flower Safeguard.

POUŽITÉ TECHNOLOGIE

- C++ (WinAPI, MFC)
- PHP (Nette framework)
- Vernier SDK, CURL library, JSON notation, MD5 technology, ...

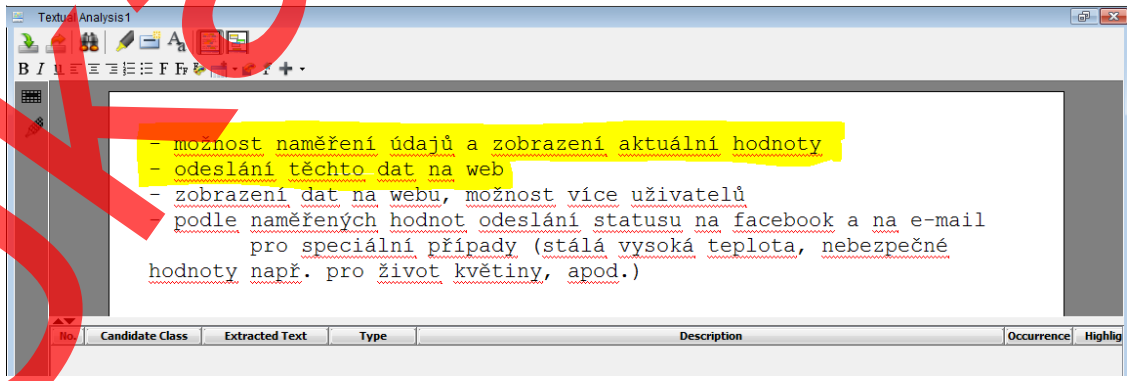
TEORETICKÁ ČÁST

KLIENTSKÁ APLIKACE

ANALÝZA

Před vlastním programováním aplikace by měla být vždy provedená stručná analýza, co se vlastně od programu požaduje. Pro analýzu jsem využil standardizovaný jazyk UML v prostředí Visual Paradigm.

Napřed jsem stručně sepsal požadavky zadání (obr. 1).



Obr. 1

TECHNICKÁ DOKUMENTACE

ÚVOD

Tato technická dokumentace je určena spíše odborníkům v oblasti programování pomocí jazyků C++ (WinAPI) a PHP, ale je koncipována tak, aby ji pochopili i méně zkušení uživatelé programu. Program sám je velice komplexní, vysvětlení každé třídy, metody a parametru by bylo velice vyčerpávající a zdlouhavé, proto jsou zde popsány jednotlivé celky kódu programu a z něj jsou pak vyzdviženy ty nejdůležitější části.

Program Flower Safeguard je rozdělen na dvě části – klientskou, napsanou v jazyce C++, a serverovou, napsanou v jazyce PHP. Tyto dvě části spolu komunikují pomocí informací předaných v URL a pomocí JSON notace.

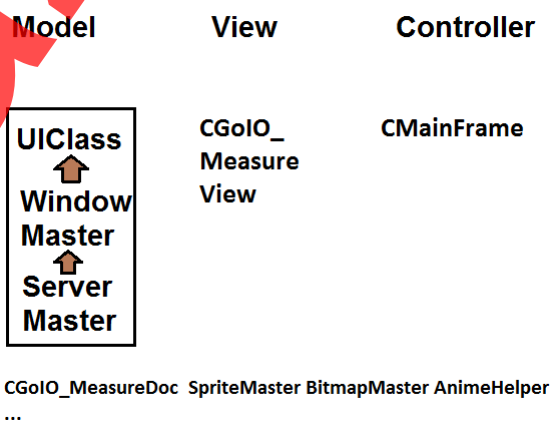
KLIENTSKÁ ČÁST PROGRAMU

Klientská část programu je zpracována v programovacím prostředí Microsoft Visual Studio 2010. Aplikace je napsána v jazyce C++ s využitím platformy MFC, protože jsou tyto technologie využity již předpřipraveným základním programem od firmy Vernier s názvem GoIO_Measure. Tento program poskytl základní stavební kámen celé klientské aplikace, především je v něm názorně vysvětleno používání vestavěných funkcí v dynamicky linkovaných knihovnách pro práci s vlastními měřicími přístroji. Díky platformě MFC jsem mohl jednoduše aplikovat model MVC (model, view, controller), který využívá i serverová část programu. MVC znamená rozdělení aplikace do tří základních částí, a sice na model, view a controller. Model se stará o základní výpočet včetně komunikace se serverem a zařízením a jsou v něm obsaženy všechny výpočetní algoritmy, které jsou jádrem celé aplikace. View je určen pro vykreslování veškeré grafiky na obrazovku a controller se stará o komunikaci s uživatelem, díky čemuž ovládá model a view.

V aplikaci jsem tento návrh implementoval takto:

4. Controller je reprezentován třídou CMainFrame spojenou přímo se základním ovládáním programu jako šablona;
5. View je reprezentován třídou CGoIO_MeasureView a jejím objektem pView;
6. Model je reprezentován třídou UIClass a jejím objektem pCore.

Tento implementovaný návrh je názorně ukázán na obrázku (obr. 8). Jak je vidět v jeho dolní části, existuje řada tříd, které komunikují se všemi třemi částmi modelu MVC. Dále je důležité si všimnout, že třída UIClass je potomkem jednoduchých tříd WindowMaster a ServerMaster. Samozřejmě by všechny metody a proměnné mohly být implementovány v jediné třídě, ale dědičnost jsem zde použil především pro přehlednost celého kódu.



Obr. 8

Nyní následuje krátký popis jednotlivých tříd spolu s vysvětlením nejdůležitějších parametrů, metod a algoritmů.

HLAVNÍ TŘÍDA APLIKACE

Hlavní třída aplikace je CGoIO_MeasureApp, jejíž základ byl převzat z programu GoIO_Measure od firmy Vernier. Obsahuje nutné volání platformy MFC, určuje velikost okna a spouští controller.

Další hlavní částí aplikace jsou soubory *General.h* a *GeneralDialog.h* obsahující globální proměnné a konstanty využívané celou aplikací.

Velice důležitým souborem je bezpochyby také *FlowerSafeguard.rc* obsahující reference na všechny zdroje a obsahující také definice dialogů (jak, kde a co mají zobrazit).

Poznámka

V celé aplikaci se snažím využívat maďarskou notaci všude tam, kde to jen lze (označení typů proměnné v názvu), dále se snažím přesně pojmenovávat třídy, metody i proměnné podle toho, k čemu jsou určené, a místo polí využívat vektory, které jsou v C++ velice dobře implementované a existuje zde více vestavěných metod pro práci s nimi.

POMOCNÁ TŘÍDA – BITMAP

První z pomocných tříd je třída Bitmap. Tato třída byla částečně převzata z knihy [12].

Tato třída umožňuje pohodlnou práci s bitmapovými soubory – načítání, úpravy, apod.

```
class Bitmap
{
private:
	CImage *m_hBitmap;
	int m_iIndex;
	int m_iWidth;
	int m_iHeight;

protected:
	void Dispose();

public:
	Bitmap();
	Bitmap(string szFileName);
	Bitmap(UINT uiResID);
	virtual ~Bitmap();

	// attributes
	void setIndex(int i) { m_iIndex = i; }
	int getIndex() { return m_iIndex; }

	BOOL Create(string szFileName);
	BOOL Create(UINT uiResID);
	void Draw(CDC *pDC, int x, int y, int iStyle = STYLE_NONE,
		COLORREF crTransColor = RGB(255, 255, 255));

	void SetWidth(int width) { m_iWidth = width; }
	void SetHeight(int height) { m_iHeight = height; }
	int  GetWidth() { return m_iWidth; };
	int  GetHeight() { return m_iHeight; };
};
```

Obr. 9

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.

POMOCNÁ TŘÍDA – SPRITE

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); pomocí cyklu se tak může měnit pozice, apod. Každému objektu je také přiřazená souřadnice Z určující, který objekt se při překrytí dvou objektů zobrazí.

```
class Sprite
{
private:
 // added
 int m_iType;
 string m_sLabel;
 COLORREF m_crLabelColor;

 bVect       *m_pBitmap;
 Bitmap            *m_pHoverBitmap;
 bool                        m_bHovered;
 int         m_iNumFrames, m_iCurFrame;
 int         m_iFrameDelay, m_iFrameTrigger;
 RECT        m_rcPosition;
 POINT       m_ptVelocity;
 int         m_iZOrder;
 RECT        m_rcBounds;
 BOUNDSACTION  m_baBoundsAction;
 BOOL        m_bHidden;
 BOOL        m_bDying;
 BOOL        m_bOneCycle;
 int                         m_iColorIndex;

protected:
 void        UpdateFrame();

public:
 // public attributes
 MDoc *m_pDevice;                    // DEVICE
 DeviceInfo *m_pDeviceInfo;          // DEVICEINFO
 int m_iProjectID;                           // PROJECT
 bool m_bPauseOnSize;                        // during the association

 // added
 void SetType(int iSpriteType) { m_iType = iSpriteType; }
 int GetType() { return m_iType; }
 string GetLabel() { return m_sLabel; }

 void SetBaseColor() { m_crLabelColor = pMaster->getTextColorA(); }
 void SetJustColor(COLORREF color) { m_crLabelColor = color; }
 void SetLabel(string sLabel) { m_sLabel = sLabel; m_crLabelColor = pMaster->getTextColorA(); }
 void SetLabel(string sLabel, COLORREF crColor) { m_sLabel = sLabel; m_crLabelColor = crColor; }
 void SetJustLabel(string sLabel) { m_sLabel = sLabel; }
 COLORREF GetJustColor() { return m_crLabelColor; }

 Sprite(Bitmap* pBitmap, Bitmap *bHover = NULL);
 Sprite(bVect *Bitmaps, Bitmap *bHover = NULL);
 Sprite(Bitmap* pBitmap, RECT& rcBounds,
  BOUNDSACTION baBoundsAction = BA_STOP, Bitmap *bHover = NULL);
```

```
        // initializing
        static int setUser(string userName, string userPassword);
        static void setWebData(string webURL) { m_sWebURL = webURL; }
        static void setUserProject(int projectID) { m_iUserProjectID = projectID; }

        // helping methods
        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; }
};
```

Obr. 21

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 odmítl data přijmout. Všechny tyto informace jsou uloženy v proměnných v části *private*.

Třída sama pak na jedné straně obsahuje prosté funkce k uložení dat do těchto proměnných, 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 metoda provádí. Je to dáno především přehlednými funkcemi knihovny CURL starající se o komunikaci se serverem a také jednoduchým užitím parseru JSONCpp, který se zabývá převodem souboru, naformátovaným podle konvence JSON, do vestavěných datových typů jazyka C++. V neposlední řadě tomu pomáhají i nadefinované datové typy speciálně pro Flower Safeguard, které činí kód čistším a přehlednějším.

Nyní následuje ukázka kódu metody *sendDataThread*, starající se o vlastní odeslání dat na server. Tato metoda je volána metodou *sendData* pomocí jednoduchého příkazu `_beginthread(&ServerMaster::sendDataThread, NULL, sUrl);` kterým 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 vykonává asynchronně (pozn. většinou se kód 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;
```

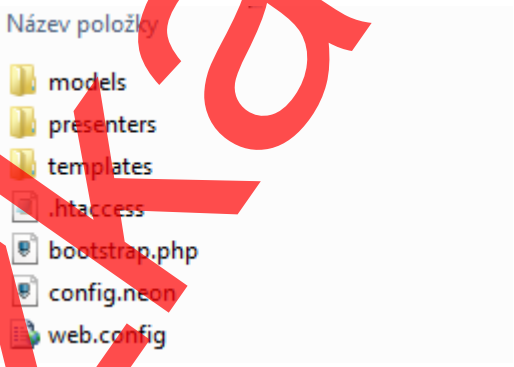
```
CURL *curl_handle;
stringstream ss;
Reader reader;
Value data;

// initialization
curl_global_init(CURL_GLOBAL_ALL);
curl_handle = curl_easy_init();

// testing the internet connection
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; // checking internet connection
}
else
{
        m_bNotConnected = false; // checking internet connection
        // checking everything is all right
        newStream(ss);
        ss << chunk.memory;
        reader.parse(ss, data);
        if(chunk.memory)
                free(chunk.memory);
        // error, again
        if (data["success"].asString() != "true")
        {
                if (!m_bServerDenied)
                {
                        // communication is broken
                        m_bServerDenied = true;
                        // do not save the data
                }
        }
}

// write error
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);
```

```
html, body { margin: 0px; padding: 0px; }
body { font-family: "Helvetica", "sans-serif", "Arial"; font-size: medium; color: rgb(10, 120, 10); margin: 0px auto;
div.heading { padding: 5px; margin: 0px; }
div.sidebar {  padding: 5px; margin: 0px; float: left; width: 200px; text-align: left;  }
div.content { padding: 5px ; margin: 0px auto; width: 600px; float: right; text-align: left; }
div.footer { clear: both; margin: 10px; }
div.flash {color: black; background: rgb(230, 255, 230); border: 1px solid green; padding: 1em; margin: 1em 0;}
div.center { text-align: center; }

a { color: rgb(10, 120, 10); }
a:focus { font-weight: bold !important; }
a:hover { font-weight: bold; }
a[href^="error:"] {background: red; color: white;}

form {margin: 1px; padding: 0px; }
input[type=text] { border: solid rgb(160, 250, 160);}
input[type=text]:hover { border-top: solid rgb(200, 255, 200); border-left: solid rgb(200, 255, 200); }
input[type=text]:focus { border-top: solid rgb(200, 255, 200); border-left: solid rgb(200, 255, 200); }
input[type=password] { border: solid rgb(160, 250, 160);}
input[type=password]:hover { border-top: solid rgb(200, 255, 200); border-left: solid rgb(200, 255, 200); }
input[type=password]:focus { border-top: solid rgb(200, 255, 200); border-left: solid rgb(200, 255, 200); }
input[type=button] { font-size: 90%; margin: 1px; padding: 0px; }
input[type=submit] { font-size: 90%; margin: 1px; padding: 0px; }

*[indent] { padding-left: 1em; }
.highlight { background-color: rgb(230, 255, 230); }
ul { margin-top: 5px; }
li { color: #111; }

h1 { margin: 10px; font-size: xx-large; }
h1.header { color: rgb(10, 120, 10); background: rgb(230, 255, 230) url(../images/background-heading.jpg); }
h2 { margin: 5px; font-size: x-large; }
h3 { margin: 5px; font-size: large; }
```

Obr. 32

ADRESÁŘ APP

V adresáři *app* je umístěna vlastní serverová aplikace (její jádro). Adresář obsahuje složky s modely, presentery a šablonami - viz obr. 33.



Obr. 33

Kromě souboru *.htaccess* určujícím práva přístupu (přímý přístup přes URL je do něho zamítnut, kvůli souboru *config.neon*) je 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* obsahující přístupové údaje k databázi MySQL. Většina kódu je převzata z Nette frameworku.

TŘÍDY MODELU

Protože modelová část u serverové aplikace není příliš komplexní, je zde proto uvedena jako první.

Program obsahuje a přímo využívá několik tříd modelů – Authenticator, Device, Mail, 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 v tomto

formátu se přenáší pomocí URL při komunikaci klient – server a je také takto uloženo v databázi MySQL.

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

Poslednímy 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, popř 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, tak je uživateli podle jeho nastavení zasláno upozornění).

ZÁKLADNÍ PRESENTERY

Ke každé větší akci webové stránky je přiřazen vlastní presenter (všechny opět dědí od základní třídy BasePresenter vkládající 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. Toto 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, ...).

ZÁKLADNÍ ŠABLONY

Menší akce webové stránky se spravují užitím nové metody presenteru začínající názvem *action*. Ke každé akci (včetně výchozí akce default) je přiřazena jedna 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 vnořovat do šablony programový kód a podmínit vykreslení určitých částí.

TŘÍDA APIPRESENTER

Tato třída vytváří základní komunikační bránu mezi klientskou a serverovou aplikací. Podle speciálně nadefinovaných rout (tvaru URL) program pozná, kterou metodu (akci) této třídy má zavolat. Názvy metod a jejich parametrů přímo korespondují s požadavky klietnské aplikace na server.

Pro ukázku je uveden zdrojový kód metody *actionProject* (obr. 34).

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, zdali má uživatel skutečně právo přistupovat k projektu. Tato kontrola se provádí pomocí třídy User, která pokládá příslušný dotaz databázi.

PRAKTICKÁ MĚŘENÍ

MĚŘENÍ PRŮBĚHU POČASÍ

ÚVOD

Popis praktických měření, která jsem uskutečnil sám v rámci testování aplikace, lze využít i jako návod pro práci s programem. Jak bylo ale již řečeno v úvodu celé práce, ovládání programu je poměrně intuitivní a snad pro každého uživatele srozumitelné.

Dne 30. 4. 2012 jsem zahájil praktické měření, které mělo za úkol sledovat průběh počasí pomocí tří fyzikálních veličin – teplota, tlak a osvětlení.

Všechny veličiny byly měřeny pomocí měřících senzory firmy Vernier (teploměr GO-TEMP, barometr BAR-BTA a luxmetr LS-BTA); senzory byly umístěny na přímém slunci. Pro zajímavost jsem umístil také jeden teploměr do budovy, abych porovnal teplotu naměřenou venku a uvnitř.

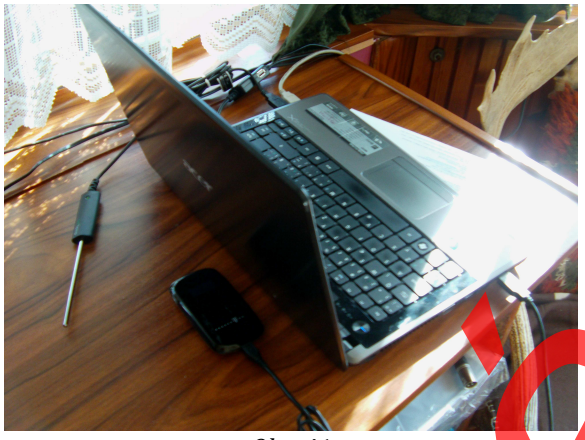
PŘÍPRAVA

Přípravu jsem zahájil brzy ráno. Senzory pro měření jsem umístil na takové místo, abych je mohl mít propojené s počítačem a zároveň aby měřily stav počasí venku.

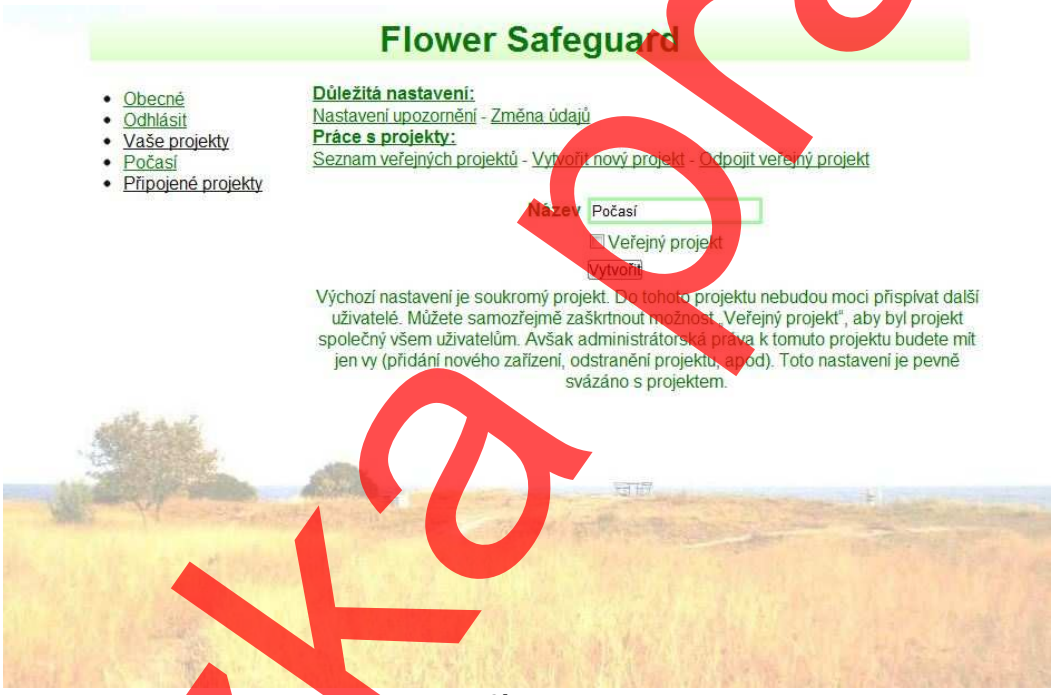
Obr. 40

Všechny sensory jsem pomocí USB připojil k notebooku. K dalšímu notebooku v jiné místnosti jsem připojil jiný teploměr, abych mohl sledovat teplotu uvnitř budovy.

Dále jsem se zaregistroval a přihlásil na serveru [10] jako nový uživatel a vytvořil jsem nový projekt „Počasí“ pro toto měření.



Obr. 41

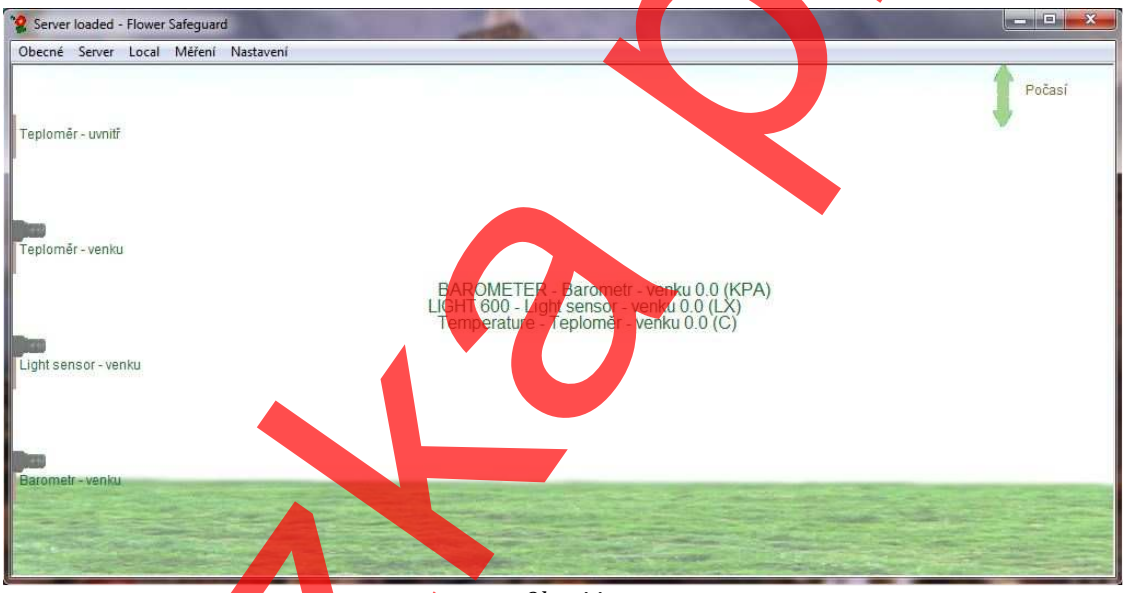


Obr. 42

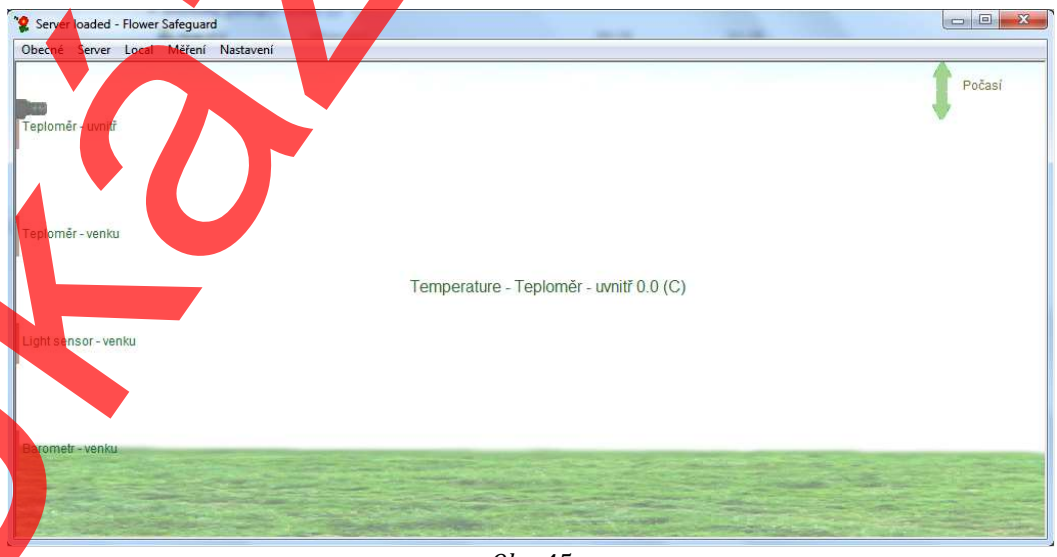
Po vytvoření projektu jsem si tento projekt otevřel a přidal nové předpisy zařízení pro každý přístroj (horní hranici jsem nastavil na 1000, protože v tomto druhu měření nebudeme potřebovat žádná upozornění).

Na obou počítačích jsem otevřel klientskou aplikaci, v té jsem se přihlásil a dvojím kliknutím myši na název projektu jsem ho otevřel. Poté jsem ke každému předpisu zařízení přiřadil přístroj a zvolil pro každý sensor jednotku, ve které bude přístroj měřit (periodu měření jsem pro každé zařízení nastavil na 2000 ms).

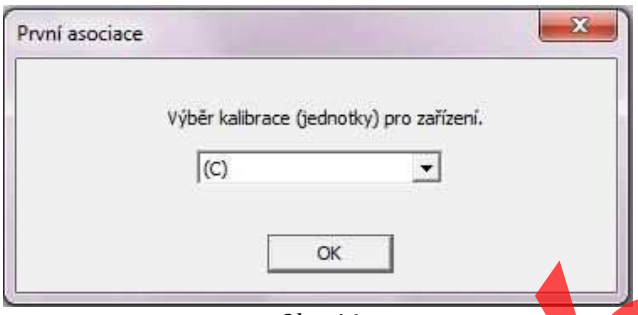
Obr. 43



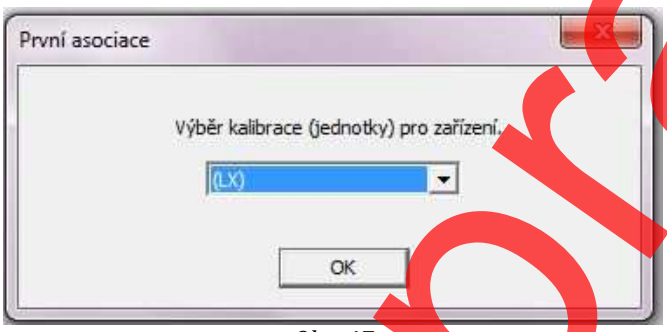
Obr. 44



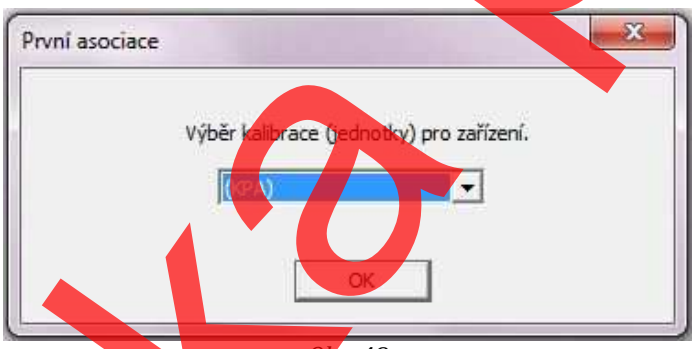
Obr. 45



Obr. 46



Obr. 47



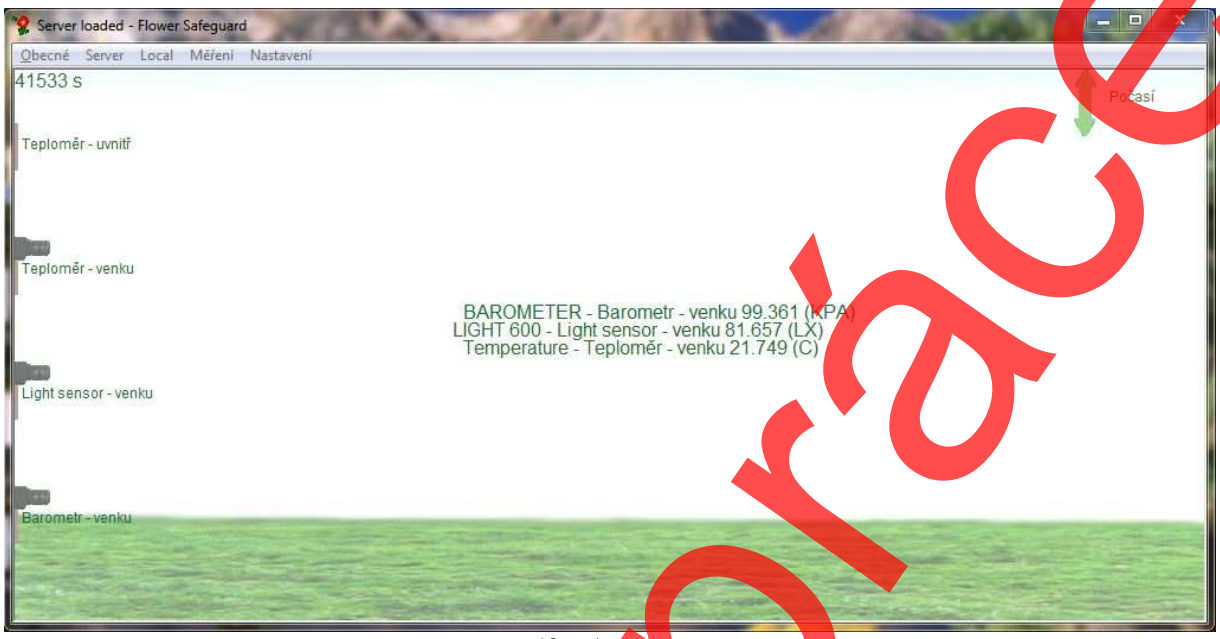
Obr. 48

Tímto byla příprava hotová a mohlo začít vlastní měření.

MĚŘENÍ

Měření jsem zahájil přibližně v 8:30 a ukončil zhruba v 20:00. Během procesu měření nebyl registrován žádný výpadek ani přerušení. Program sám naštěstí měření nepřeruší, i když selže internetové spojení – data se mezitím ukládají do souboru a jakmile se opět obnoví internetové spojení, data se okamžitě odešlou na server.

Před ukončením měření okno programu na prvním počítači vypadalo tak, jak ukazuje obr. 52.



Obr. 52

VÝSLEDKY MĚŘENÍ

Po ukončení měření a odpojení všech zařízení jsem se opět přihlásil na server pomocí prohlížeče a otevřel jsem projekt „Počasí“. Graf v projektu vypadal tak, jak zobrazuje obr. 53.

Obr. 53

Z grafu samotného není mnoho poznat; můžeme pouze vidět, že teplota (venkovní i uvnitř domu) a tlak neměly žádné extrémní výkyvy. Nejlépe však bude si tento graf rozdělit na několik grafů pro každý typ senzoru. Proto jsem v projektu 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 Přizpůsobit panel nástrojů 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 (obr. 54 - obr. 56).

Tyto grafy jsou mnohem názornější a můžeme už z nich vyčíst různé informace obecně pro důsledné sledování počasí.

Přibližně v půl sedmé v podvečer se již začalo pomalu stmívat a tlak začal klesat přibližně v jednu hodinu odpoledne. 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 nás dobře; během měření nebyla budova dodatečně vytápěna). Teplota venku se v této době pohybovala v intervalu přibližně 22 °C až 34 °C. Naměřená data jsou uložena v souboru *praktickemereni1.xls*.

Obr. 54

ZÁVĚR

Pomocí programu Flower Safeguard lze efektivně provádět i dlouhodobá (a důležitá) měření pomocí senzorů firmy Vernier. Komunikace programu s těmito senzory byla otestována na několika měřeních s teploměrem, luxmetrem, barometrem a čidlem polohy. Přidání případně dalších senzorů do programu nebude již tak náročné.

Současně bylo otestováno, že program může spolehlivě provádět měření a zaznamenávat měřená data u tzv. prioritních měření. U nich je potřeba informovat uživatele o průběhu měření tehdy, pokud budou naměřeny hodnoty mimo předem daný interval hodnot daného měření. Tyto informace jsou ve stávající verzi programu odesílány uživateli na zadanou e-mailovou adresu. Poslat např. SMS zprávu na mobilní telefon nebo se spojit s jiným moderním přenosným zařízením již není výrazný problém. Tato funkce programu může sloužit nejen k pobavení, ale může mít i velmi důležité praktické využití - např. při ochraně budov v nepřítomnosti majitele nebo ostrahy. Program samotný – jak klientská, tak serverová část – se choval velice stabilně po celou dobu měření. Současně s tím nebyla přetěžována operační paměť či další důležité komponenty počítače, jejichž přetížení či zpomalení by omezovalo další paralelně konanou práci na počítači.

Již na minulých řádcích jsem naznačil některá případná vylepšení programu. Další napadnou jistě toho čtenáře, kterého tato problematika zajímá a který najde řadu dalších praktických využití mého programu.

ZDROJE, LITERATURA, ODKAZY

Zdroje, odkazy a použitá literatura:

[1] Stránka projektu: http://injectionsoft.com
[2] CURL library: http://curl.haxx.se
[3] Edufor, s. r. o.: http://www.edufor.cz
[4] Google: http://www.google.com
[5] Stránky J. Reichla: http://www.jreichl.com
[6] JSONCpp parser: http://jsoncpp.sourceforge.net
[7] Microsoft Dreamspark: https://www.dreamspark.com
[8] Nette framework: http://www.nette.org
[9] Systémy Vernier: http://www.vernier.com
[10] Serverová část aplikace: http://fsafe.injectionsoft.com
[11] BRÁZA J., *PHP 5: začínáme programovat*. Praha, Grada Publishing, 2005
[12] MORRISON M., *Naučte se programovat počítačové hry za 24 hodin*. Praha, Computer Press, 2004
[13] PRATA S., *Mistrovství v C++*. Praha, Computer Press, 2007
[14] TÖPFER P., *Algoritmy a programovací techniky*. Praha, Prometheus, 1995

„Hypotézy vytvářím, avšak hypotézám - ani svým vlastním - nevěřím do posledního písmene.“ – Isaac Newton