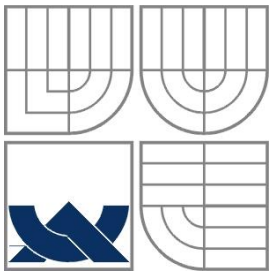
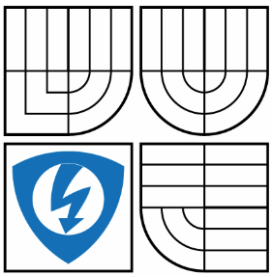


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SBĚR A VYHODNOCOVÁNÍ DAT Z TESTOVACÍCH STANIC

Collection and analysis of data from testing stations

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. David Hák

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Marie Havlíková, Ph.D.

BRNO 2015

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Bc. David Hák
Ročník: 2

ID: 162238
Akademický rok: 2014/15

NÁZEV TÉMATU:

Sběr a vyhodnocování dat z testovacích stanic

POKYNY PRO VYPRACOVÁNÍ:

1. Popište proces měření LED Modulů předních světlometů se zaměřením na sběr zdrojových dat pro realizaci databáze.
2. Navrhněte a realizujte databázi pro ukládání naměřených dat v Microsoft SQL.
3. Vytvořte webovou aplikaci v ASP .NET pro převod naměřených dat ze souborového serveru ve vygenerovaném formátu XML pomocí local části, která dále umožní zobrazování trendů měření LED modulů, exporty trendů do požadovaných formátů RTF, DOC nebo PDF.
4. V souladu s požadavky externí firmy Automotive Lighting, s.r.o realizujte automatické korekce testovaných parametrů na základě souboru naměřených dat.
5. Dosažené výsledky diskutujte a zhodnoťte.

DOPORUČENÁ LITERATURA:

[1] BÍLÁ, J., KRÁL,F. Databázové a znalostní systémy, Skriptum ČVUT FS, Praha, 1999
[2] Firemní literatura

Termín zadání: 9. 2. 2015 ***Termín odevzdání:*** 18.5.2015

Vedoucí práce: Ing. Marie Havlíková, Ph.D.
Konzultanti diplomové práce: Ing. Martin Dovica, AL Jihlava

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav automatizace a měřicí techniky

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Abstrakt

Diplomová práce je soustředěna na tři základní části – popsání teoretické a praktické části měření LED modulů předních světlometů automobilů, poté návrh a realizace způsobu sběru a uchovávání naměřených zdrojových dat a následně vytvoření serverové aplikace umožňující skrze webové rozhraní výběr a zobrazení požadovaných dat.

Klíčová slova

Automotive lighting, sběr a analýza dat, databázový systém, LED, SQL, monitorování, vyhodnocování dat, C#, ASP.NET, Microsoft Visual Studio

Abstract

The thesis is focused on three basic parts – description of the theoretical and practical parts of measurements of LED modules front automotive headlamps, then design and realization of system for collecting and storing measurement data source and finally creation of server applications allowing via a web interface selection and display the required data.

Keywords

Automotive lighting, collection and analysis of data, database system, LED, SQL, monitoring, evaluating data, C#, ASP.NET, Microsoft Visual Studio

Bibliografická citace:

HÁK, D. *Sběr a vyhodnocování dat z testovacích stanic*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 83s. Vedoucí diplomové práce byla Ing. Marie Havlíková, Ph.D..

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Sběr a vyhodnocování dat z testovacích stanic jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **18. května 2015** ……
podpis autora

Poděkování

Rád bych tímto poděkoval panu Ing. Martinu Dovicovi za poskytnutí tématu diplomové práce a následné odborné vedení a podporu.

Dále děkuji vedoucí diplomové práce Ing. Marii Havlíkové, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **18. května 2015** ……

podpis autora

Obsah

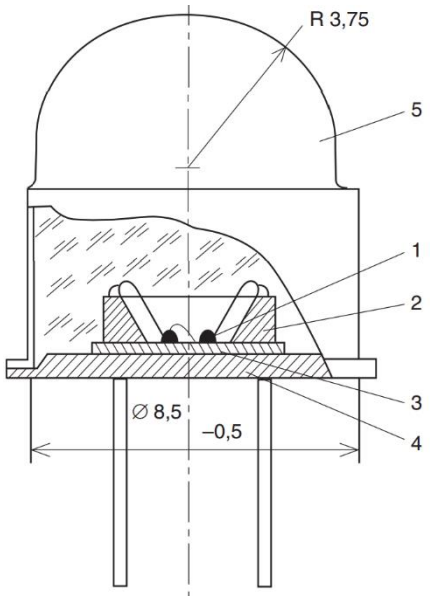
1 ÚVOD

Automotive Lighting (dále jen AL) je největším producentem automobilových světlometů v Evropě, kterou vlastní firma Magneti Marelli z koncernu Fiat. [2]
Jihlavské zastoupení AL vyvíjí a vyrábí přední světlomety do automobilů, vývojové oddělení v Jihlavě je druhé největší v rámci AL. Mezi přední zákazníky patří BMW, Mercedes, VW, Škoda, Renault, Honda, Peugeot a Porsche. [1]
V poslední době se u prémiových typů vozů těchto značek prosazuje technologie LED v předních světlometech, jenž zajišťuje extrémně vysokou životnost světlometů (delší než je plánovaná životnost celého automobilu), výrazně nižší spotřebu energie v porovnání s xenonovými či halogenovými typy světlometů, produkci osvětlení blížící se kvalitě denního světla a dále i vyšší možnosti využití prostoru ve světlometech – větší možnosti upravení designu světlometů. [1]
Vzhledem k vysoké životnosti LED ve světlometech se již při návrhu nepočítá s možností výměny LED modulu ve světlometu, což klade vysoké nároky na kontrolu těchto LED modulů již při výrobě. Aby byla tato kontrola pro pracovníky na oddělení kontroly výrobků co nejvíce zjednodušena (menší nárok na technickou odbornost pracovníků, kratší čas na kontrolu jednoho světlometu, menší pravděpodobnost chybného vyhodnocení výsledků pracovníkem), byl firmou AL vytvořen nástroj pro testování LED světlometů – LED Module Check. [2]
V současné době tyto nástroje slouží k vyhodnocení jednotlivých testů LED světlometů. Nejsou zde žádné možnosti sběru a analýzy těchto dat, které mohou posloužit k optimalizacím, prognózám či analýzám, jaký vliv mají různé postupy/materiály na kvalitu a vlastnosti výsledného produktu. Jelikož se jedná o velmi cenná data, byl zadán úkol na vytvoření serverové aplikace s webovým rozhraním, která by byla schopna sbírat data z těchto měřících nástrojů, ukládat je do databáze a následně provádět nad těmito daty analýzy, prognózy a další požadované aplikace.

2 LED SVĚTLOMET

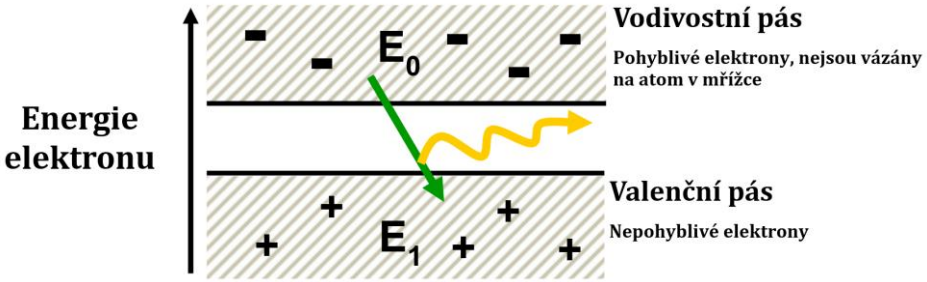
2.1 LED

LED (z anglického Light Emitting Diode) je polovodičová součástka obsahující PN přechod emitující světelné záření, je-li buzen průchodem elektrického proudu. Přestože její princip byl znám již ve 20. letech minulého století, tak se až v roce 1962 objevily první realizace v praxi. Postupným zdokonalováním LED (nové základní materiály, technologické postupy, rozšíření sortimentů o další barvy vyzařovaného světla, zvýšení účinnosti, zvýšení životnosti) našly uplatnění v širokém spektru možných aplikací. [3]



Obrázek 1 - Základní konstrukční uspořádání světelné diody se dvěma krystaly [3] 1) polovodič s přechodem PN 2) reflektor 3) keramická destička odvádějící teplo 4) podložka 5) polokulová čočka

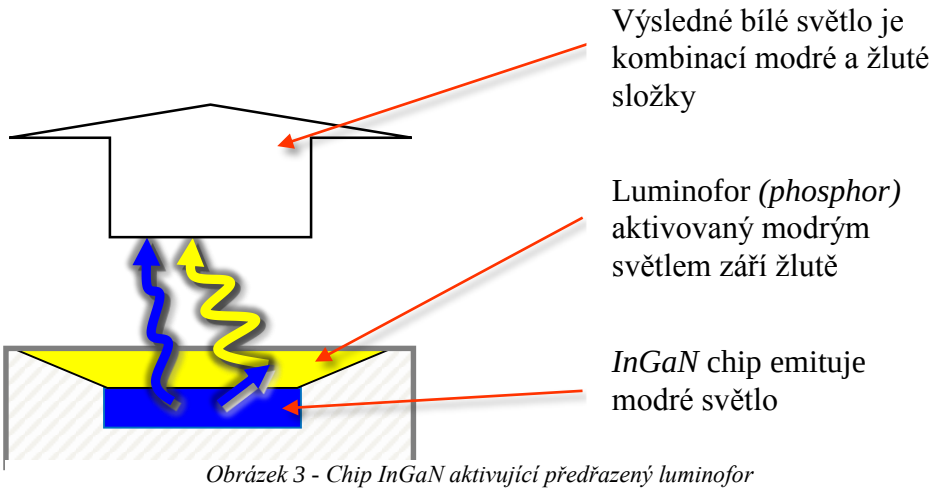
Konstrukce základního typu LED je naznačena na obrázku 1. K vytvoření polovodičových přechodů PN se používají zejména polovodiče typu A^{III} B^{V} vysoké čistoty, legované malým množstvím vhodných příměsí, které vytvářejí buď přebytek elektronů (materiál typu N), nebo jejich nedostatek, a tedy přebytek děr (materiál typu P). Na rozhraní obou polovodičů vzniká již zmiňovaný přechod typu PN. Při správném směru průchodu stejnosměrného proudu tímto přechodem probíhá vzájemné přibližování elektronů a děr k PN přechodu a tím dochází k jejich rekombinaci. Během rekombinace elektronu a díry se uvolní určité kvantum energie, které je vyzářeno ve formě fotonu mimo strukturu LED (emise světla), viz obrázek 2. Elektrická energie se tímto přemění přímo na světlo určité barvy (záleží na vlnové délce, která je závislá právě na kvantu vyzářené energie), čímž je zajištěna vysoká účinnost LED a tím i minimální tepelné ztráty. Právě vysoká účinnost přeměny elektrické energie na světelnou (až 590 lm/W u zelených LED) předurčuje vysokou škálu využití a nahrazení klasických světelných zdrojů (žárovka, zářivka, halogenová žárovka a jiné). [3]



Obrázek 2 - Struktura energetických hladin polovodičového krystalu, emise světla [2]

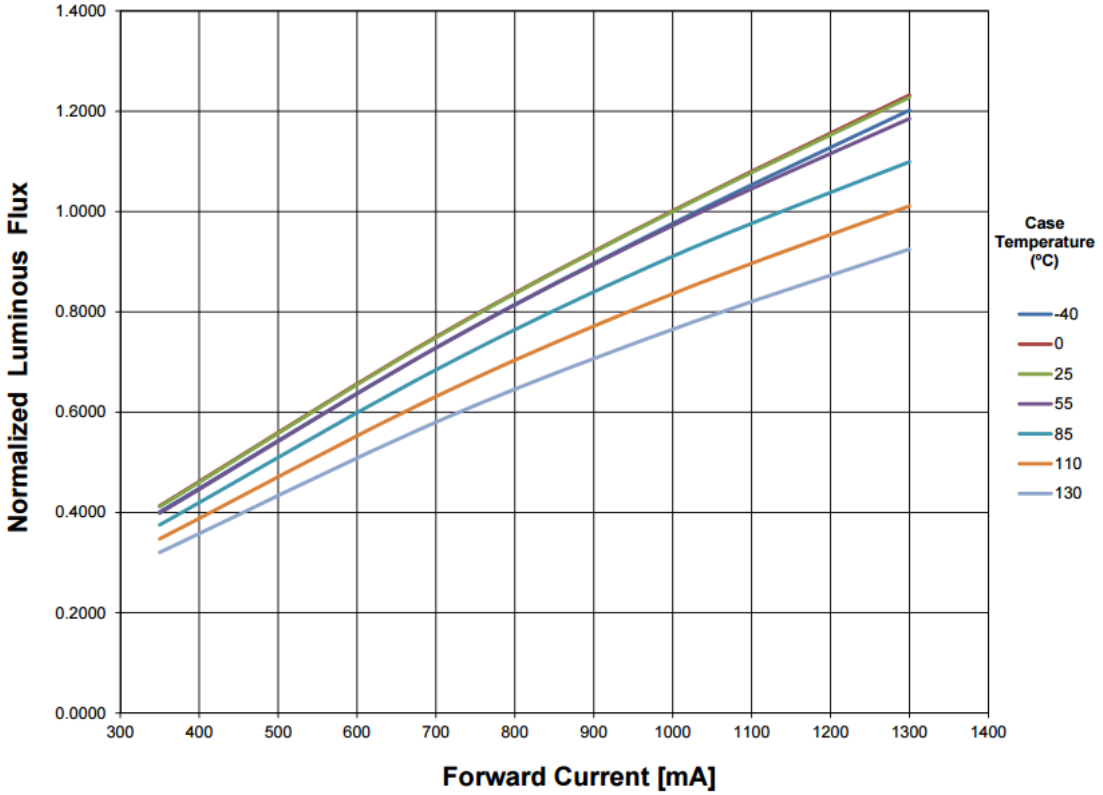
První typy LED emitovaly pouze červené světlo, po nich se na trhu objevily LED emitující zelené, oranžové, žluté a nakonec velmi důležité modré světlo, jenž umožnilo vyvinout LED emitující bílé světlo. Z principu totiž není možné vyvinout LED přímo emitující bílé světlo (emitují vždy pouze světlo o úzké vlnové délce), avšak lze upravit LED emitující modré světlo tak, aby výsledná barva emitujícího světla byla bílá. K tomuto lze využít dvou způsobů: [3]

1. Klasické míšení světel – pokud se využijí LED emitující červené, zelené a modré světlo, pak je výsledný světelný tok bílý. K tomuto způsobu je však nutné využívat specifický HW a SW, navíc výsledný jas je nižší a v důsledku nerovnoměrného opotřebovávání jednotlivých barevných LED a jejich nižšího jasu oproti ostatním dochází k nežádoucímu zabarvení bílého světla. [3]
2. Využití fosforescence luminoforů – tento způsob je podobný principu vzniku světla v klasických zářivkách. Luminofor je látka schopná pohlcovat energii a emitovat jí v podobě světla. Složení luminoforu ovlivňuje barvu emitovaného světla. Tento způsob emitování bílého světla je mnohem úspornější i méně prostorově náročný, avšak ve výsledném modro-žlutém zabarvení je potlačena zelená a červená složka, což se projevuje horším podáním barev osvětlovaných předmětů, viz obrázek 3. [2][3]



Obrázek 3 - Chip InGaN aktivující předřazený luminofor

Vyvinutím LED emitující bílé světlo se velmi rozšířila využitelnost LED, včetně klasického osvětlení prostoru. Kromě LED emitující viditelné světlo existují takové, které emitují ultrafialové nebo infračervené oblasti spektra. [3]
Nově využívané polovodičové materiály (na bázi fosfidů india, galia a hliníku), používané hlavními výrobci, se skládají z velmi složitých kombinací epitaxně vypěstovaných vrstev. Stále dokonalejší a přesnější technologické postupy při výrobě zajišťují vysokou čistotu výsledného produktu, čímž se neustále zvyšuje účinnost a teplotní odolnost LED. Díky zlepšování výrobních procesů lze stejnou technologií vyrábět LED emitující žluté, červené a oranžové světlo tak, že se pouze upraví velikost zakázaného pásu pomocí různých typů příměsí v daném polovodiči. [3]
Právě zmiňovaná teplota LED má velký vliv na životnost, účinnost a celkové barevné podání. Každý výrobce v datasheetu konkrétní LED uvádí i parametry při konkrétní teplotě. Například firma *Luxeon* u své výkonové LED *Altilon* uvádí graf závislosti proudu a světelného toku na teplotě, viz graf 1. [5]

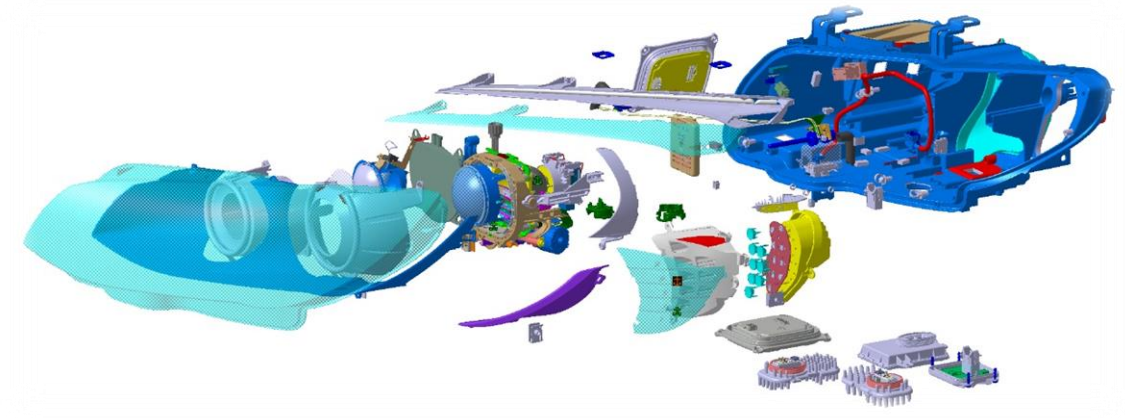


Graf 1 - Luxeon Altilon DS66 - závislost proudu a světelného toku na teplotě [5]

Jak je z grafu 1 patrné, při proudu 1300 mA a teplotách 55°C a 130°C je rozdíl světelného toku odhadem 20%. Při vyšších teplotách ovšem nedochází pouze ke snižování světelného toku, vlivem tepelného namáhání dochází i k degradaci okolních materiálů, zkracování životnosti LED i okolních komponent. Například u LED *Luxeon Rebel* v datasheetu je uveden pokles životnosti při 135°C a 150°C z 50000 hodin na 31000 hodin, vzniká tak téměř 48% pokles životnosti. Vytvoření efektivního chlazení LED je tedy velice důležitým aspektem při praktických aplikacích této technologie. [5][8]

2.2 Konstrukce světlometu

Polovodičový čip má velmi malé rozměry. První LED měly čip o velikosti plochy 0,05 mm^2, ovšem vývojem se velikost plochy zvětšovala až na jednotky milimetrů čtverečních. V důsledku toho se zvětšoval i příkon, proud a svítivost – od desetin wattů po desítky wattů, od jednotek miliampérů až po 1,5 ampéru a od jednotek lumenů až po stovky lumenů. V současné době se průmyslově vyrábějí diody s proudem řádově několika set miliampérů o příkonu desítek wattů, takže při hodnotách měrného výkonu až 100 lm/W dosahuje světelný tok diod až několika set lumenů. [2][3]



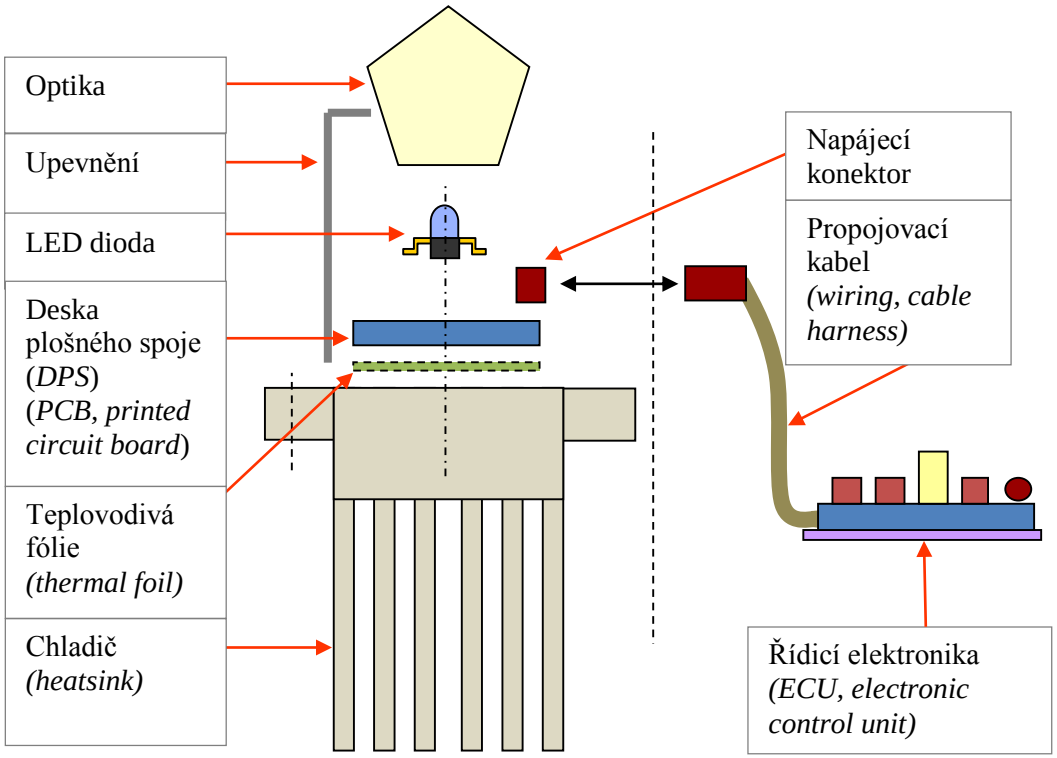
Obrázek 4 – Komponenty LED světlometu BMW M3 [2]

Z pohledu uživatele lze současné diody rozdělit do tří skupin:

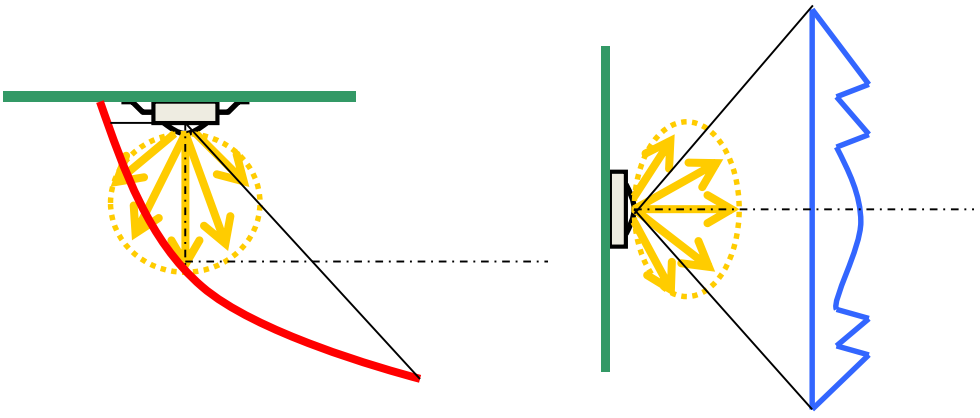
- LED o malém výkonu s proudem 1-2 mA
- Standartní LED s proudem větším než 20 mA
- Výkonové LED (high power) s proudem větším než 350 mA

LED světlomet je složen z LED modulů, které k získání většího světelného toku využívají několik LED zapojených do série nebo jedné LED se zvětšenou plochou čipu, avšak zde je nutné zajistit dostatečné chlazení čipu, jak již bylo uvedeno výše. Dostatečného chlazení se v současnosti dosahuje díky použití pasivního či aktivního chlazení LED modulů a aplikaci teplovodivé pasty mezi LED modul a chladič. Při využití zapojení LED do série jsou veškeré diody zapojeny v propustném směru a přes vhodný rezistor připojeny k napájecímu zdroji, nebo připojeny přímo na zdroj konstantního proudu (u LED s proudem větším než několik desítek miliampér). Velikost proudu, který má procházet diodami je uveden v katalogovém listu výrobce, jako jeho zdroj je použit budič (viz obrázek 7) s výstupním napětím přizpůsobeným použité kombinaci a množství LED, jedná se v podstatě o zdroj konstantního proudu, který zajišťuje optimální pracovní napětí a tím i zprostředkovaně optimální pracovní teplotu. [2][3]
Světlo emitované chipem má široký úhel rozptylu, je tedy nutné jej usměrnit pomocí vhodných optických prvků (reflektor, čočka), či u klasických LED pomocí krytu z epoxidové pryskyřice, jehož barva zpravidla odpovídá barvě emitovaného světla.

Nastavení reflektoru či čočky ovlivňuje úhel emitovaného světelného svazku, který se v současném sortimentu pohybuje v rozmezí od 8° po 120°, viz obrázek 6. [2][3]
Ačkoliv je při výrobě diod dodržen stejný postup, tak každá dioda je jinak „kvalitní“ a má mírně odlišné vlastnosti (rozsah napětí v propustném směru při jmenovitém proudu, barvu světla, světelný tok). Diody se proto rozdělují do skupin s podobnými/stejnými vlastnostmi (napětí (tzv. napěťový bin), světelného toku (světelný bin) a barvy (barevný bin)) a poté se společně usazují do série, kde může být pro identifikaci dané skupiny usazen i tzv. binovací rezistor. [2][3]

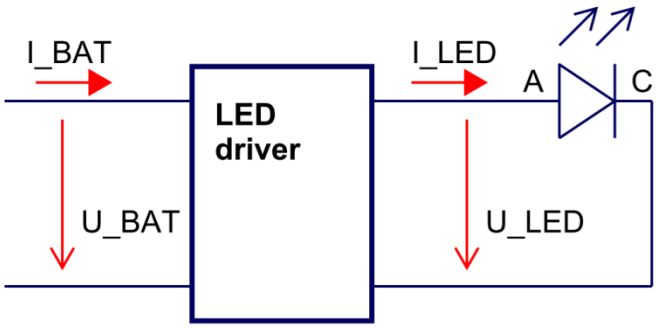


Obrázek 5 - Mechanická konstrukce LED modulu [2]



LED a reflektor LED a čočka

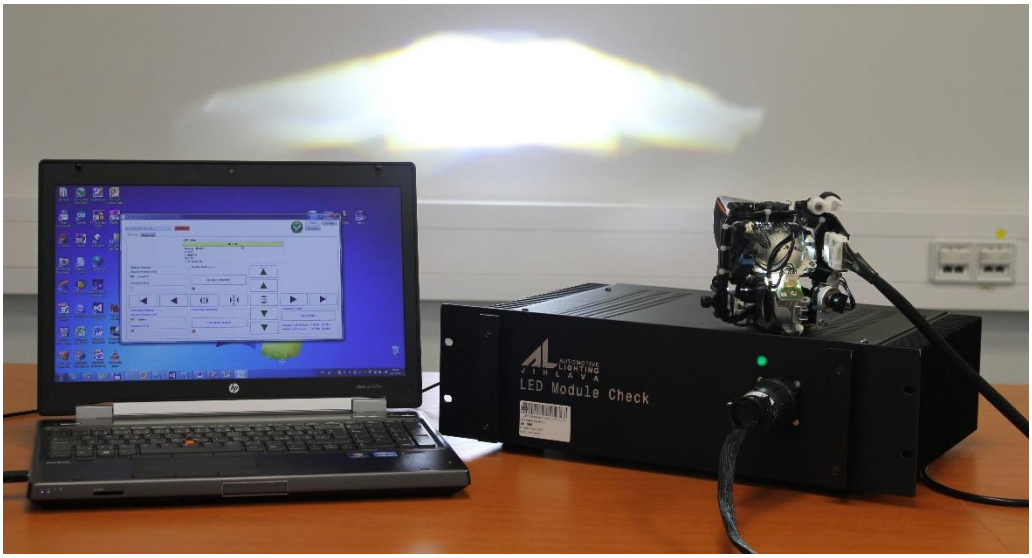
Obrázek 6 - Optická soustava LED modulu - využití reflektoru a čočky [2]



Obrázek 7 - Blokové schéma obecného budiče LED [2]

2.3 Měřené veličiny

Tak jako v kterémkoliv odvětví výroby je i v tomto případě důležitá výstupní kontrola výrobků. Jelikož je kontrola LED světlometů velice specifická a nebyl na trhu dostupný žádný specifický nástroj pro kontrolu LED modulů, byl firmou AL vyvinut *LED Module Check*, který je schopen pomocí SW (vytvořen v ASP.NET C#) a připojeného PC otestovat připojený LED modul. [2]

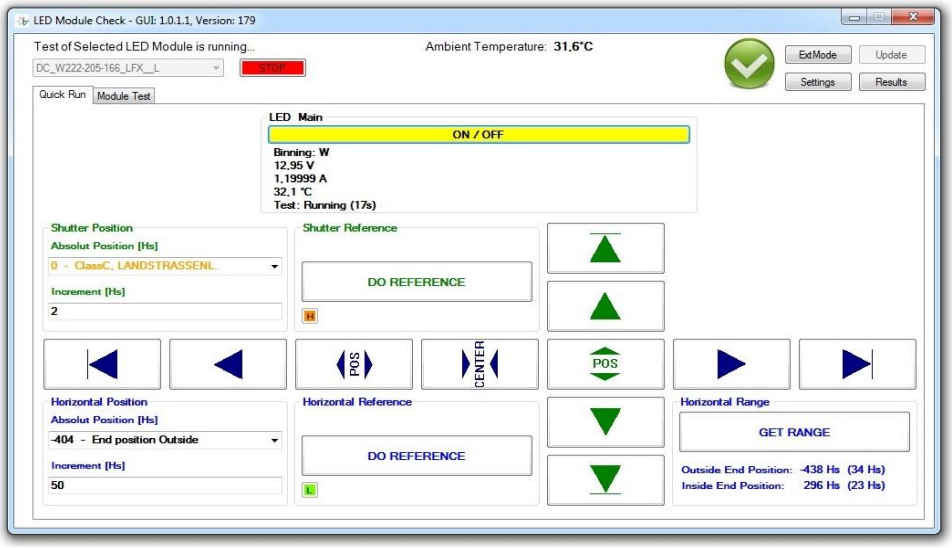


Obrázek 8 - LED Module Check [2]

LED Module Check obsahuje měřící základnu *NI cDAQ-9178*, připojené měřící karty *NI 9205, NI 9227, NI 9227, NI 9481, NI 9264*, dále 6 zdrojů proudů pro jednotlivé LED skupiny, mikrokontroler *Arduino* pro ovládání krokových motorů LED modulu a teploměr. Podrobnější popis *LED Module Check* nebyl firmou AL umožněn z důvodu ochrany duševního vlastnictví firmy.[2]

Po připojení LED modulu je nutné, aby uživatel nejprve z nabídky vybral, o jaký typ LED modulu se jedná, popřípadě binning LED, není-li v LED modulu obsažen binovací rezistor. *LED Module check* je sice schopen zjistit velikost binovacího rezistoru, tím

i určit o jaký binning (skupinu) LED se jedná a podle konfiguračního souboru určit i minimální a maximální napětí na diodách, ovšem mnohdy na modulu není binovací rezistor osazen, v takovém případě je uživatel nucen hodnotu binovacího rezistoru zadat ručně, či pomocí optické čtečky kódů naskenovat QR kód modulu, kde je požadovaná hodnota uvedena. Díky správnému výběru typu LED modulu je *LED Module Check* schopen plně ovládat veškeré prvky připojeného LED modulu, provádět a vyhodnocovat testy. [2]



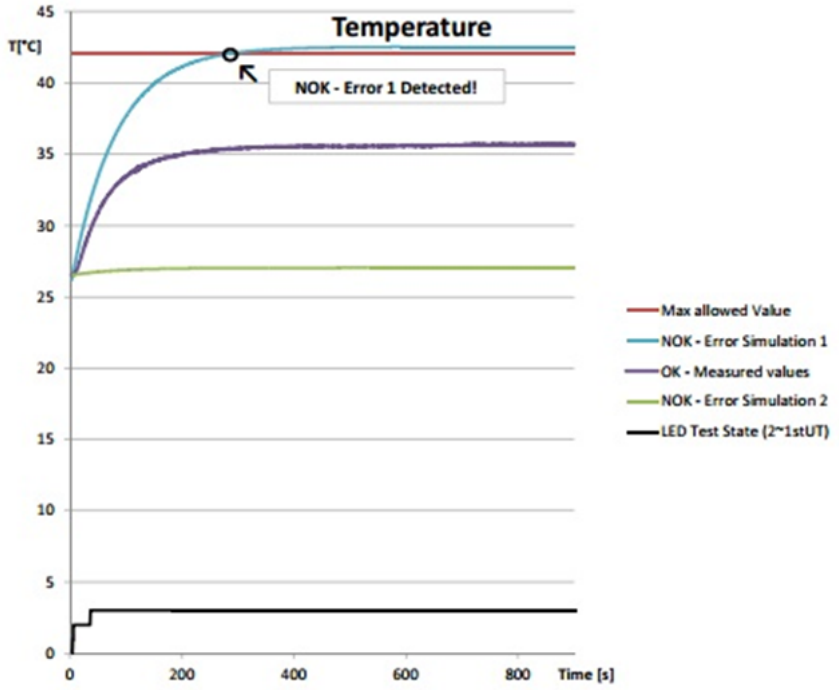
Obrázek 9 - Rozhraní SW pro LED Module Check [2]

Uživatel je schopen buď LED modul testovat manuálně (může ovládat veškeré prvky LED modulu), má-li podezření na konkrétní závadu nebo zvolit automatický test LED modulu, který je schopen pomocí teplotního testu odhalit hlavní závady modulu. Při nalezení závady je test přerušen, aby nedošlo k případnému dalšímu poškození LED modulu. [2]

LED Module Check provádí měření teploty, napětí, proudu, pozic krokového motoru, proudu procházející větrákem LED modulů a nakonec i autodiagnostiku sama sebe.

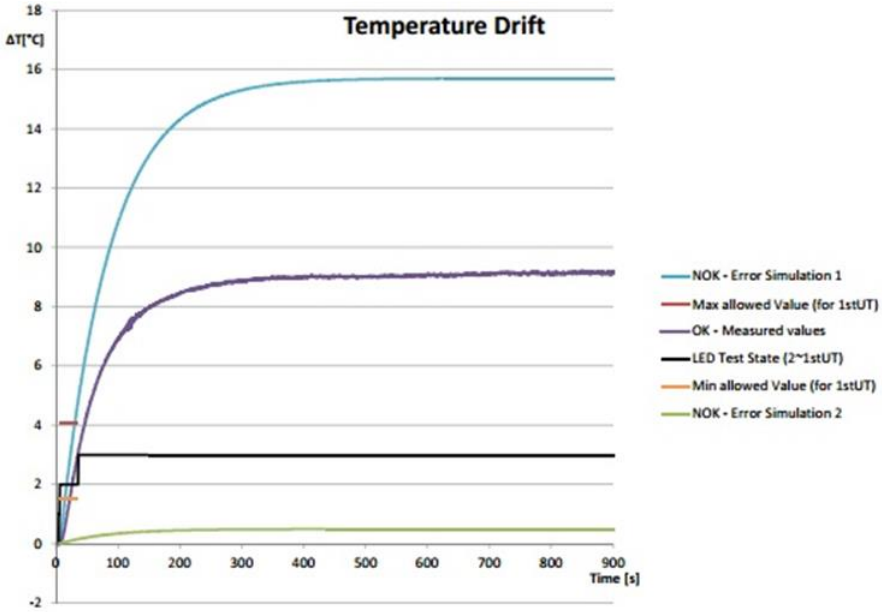
2.3.1 Teplota

Každá LED, jak již bylo uvedeno, je velmi závislá na teplotě. Teplotu diody není možné ve světlometu přímo měřit, ovšem každý LED modul má implementován NTC (termistor), jehož hodnota se považuje za teplotu diody či diod. *LED Module Check* navíc obsahuje vlastní teploměr, jenž měří teplotu prostředí. Pokud je teplota LED nižší, nežli teplota okolí – je zřejmě poškozen teploměr na driveru. Pokud je rychlost vzrůstu teploty LED příliš vysoká (viz graf 3) – pak je špatné chlazení, naopak je-li rychlost vzrůstu teploty příliš pomalá nebo neměnná, pak zřejmě došlo k poškození teploměru driveru. [2]



Graf 2 – EXCEL, Měření teploty LED modulu pomocí LED Module Check [2]

Na grafu 2 je vidět průběh měření teploty. Měření probíhalo za teploty okolí 27 °C po dobu 850 s, tudíž LED by neměla mít stejnou, nebo nižší teplotu jako okolí. Výrobce jednotlivých LED v katalogovém listu taktéž uvádí maximální hodnotu teploty, při které ještě nedochází ke snižování životnosti či poškození diody. Tato teplota (uložená v jednotlivých konfiguračních souborech LED modulů pro LED Module check) nesmí být překročena. S tímto faktem počítá konstrukce LED modulů a světlometů, proto dojde-li k tomuto překročení, pak je zřejmě špatné chlazení světlometu. [2]

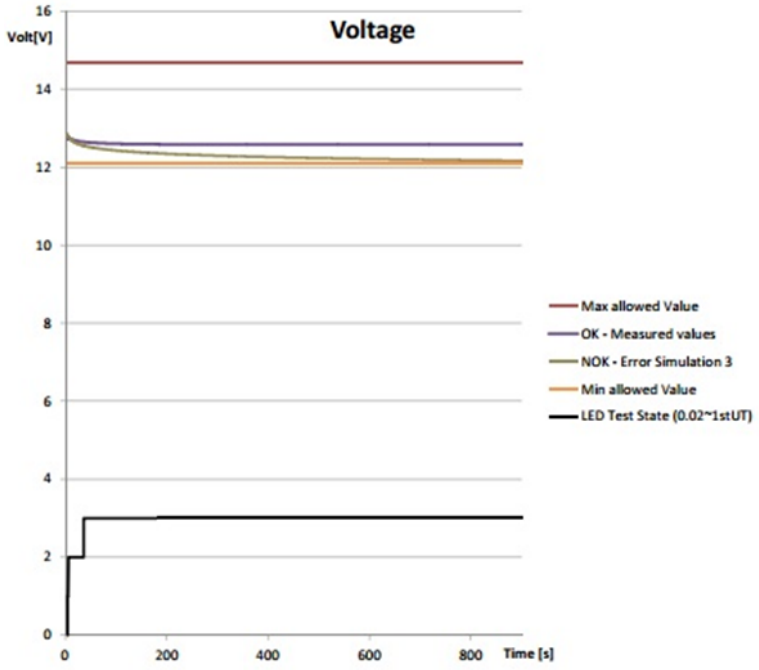


Graf 3 – EXCEL, Měření teplotního driftu LED modulu pomocí LED Module Check [2]

Při zahájení tzv. teplotního testu (viz graf 3) jsou na LED modulu rozsvíceny veškeré LED, aby byla zajištěna maximální zátěž modulu. Jako referenční teplota je následně brána teplota v čase 0 a každých 200ms po dobu 30s je snímán rozdíl teploty modulu oproti referenční teplotě. Po 30s se poslední hodnota teplotního driftu porovná s limity LED a vyhodnotí se test jako úspěšný nebo neúspěšný. V současné době jsou tyto limity teplotních driftů voleny na základě zkušeností, výsledkem této DP bude nástroj umožňující nastavení limitů na základě dlouhodobého sběru dat. [2]

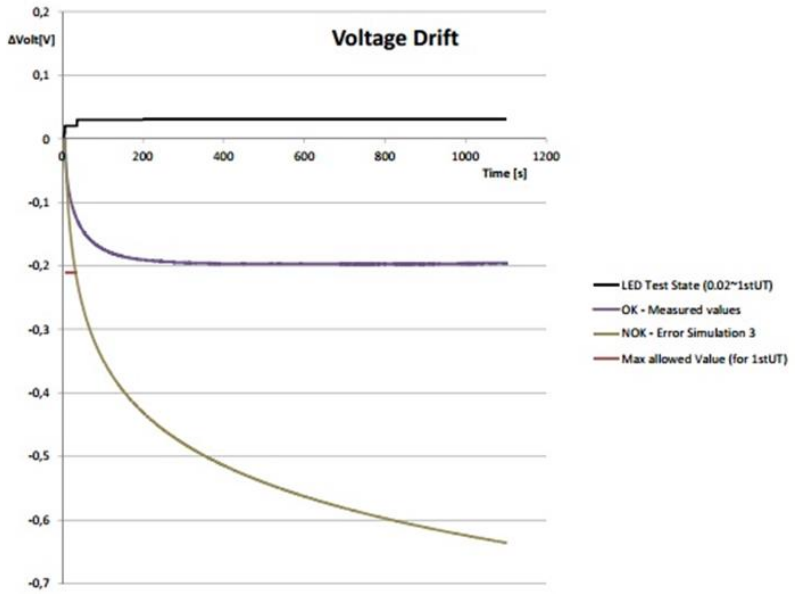
2.3.2 Napětí

Jelikož jsou jednotlivé skupiny LED napájeny zdrojem proudu, vzniká na nich napětí v určitém rozsahu. Sám výrobce v datasheetech jednotlivých LED uvádí minimální napětí, kdy LED přestává emitovat světlo a maximální napětí, kdy již hrozí nevratné poškození. LED Module check tak měří přímo napětí na jednotlivých LED skupinách obsažených v LED modulu.[2]



Graf 4 – EXCEL, Měření napětí LED modulu pomocí LED Module Check [2]

Stejně jako u teploty, tak i u napětí se během teplotního testu zaznamenávají hodnoty napětí (viz graf 5), kde se v čase spuštění teplotního testu zaznamená hodnota napětí v čase 0 s jako referenční hodnota a poté se každých 500ms měří rozdíl napětí modulu oproti referenční hodnotě. Během měření se modul zahřívá, čímž klesá elektrický odpor skupiny LED a napětí mírně klesá. Pokud během teplotního testu napětí klesne o více než maximální limit, pak je test neúspěšný. [2]



Graf 5 – EXCEL, Měření napěťového driftu LED modulu pomocí LED Module Check [2]

2.3.3 Proud

Proud poskytují zdroje proudu, maximální odchylka proudu procházející LED skupinou od proudu udávaným výrobcem je 1%, největší rozdíl je po dobu 3s po zapnutí napájení. Po těchto 3 s se hodnota proudu protékající LED modulem ustálí a snímá se. V případě překročení maximální odchylky proudu od požadované hodnoty je test okamžitě ukončen, jelikož při této chybě je buď vážně poškozen LED modul, nebo zdroj proudu. [2]

2.3.4 Pozice krokových motorů

Některé z LED modulů jsou vybaveny krokovými motory pro natáčení modulu ve světlometu, například při zatáčení vozu. Při měření pozic krokových motorů dochází v podstatě k testování pohyblivosti a referenci. *LED Module Check* umožňuje natočit krokové motory do referenční pozice (základní pozice rovnoběžná s osou automobilu), poté lze automaticky projet celý rozsah motorů (u každého modulu jsou zadány minimální rozsahy, které modul musí splnit), či pouze krokovat po zadaných intervalech nebo na konkrétní pozici. Pokud motor nedosáhne zadané pozice nebo jím začne protékat příliš vysoký proud (blokace motoru), dojde k chybě a test se ukončí. [2]

2.3.5 Otáčky větráku

Některé LED moduly obsahují aktivní chlazení pomocí větráku, díky kterému dochází k efektivnějšímu odvodu tepla, menším nárokům na objem chladiče, ovšem větrák je pohyblivé zařízení, které v tak náročných podmínkách automobilového provozu (velký rozsah teplot, vibrace, nečistoty) může být poškozeno. Testování probíhá tak, že se každých 400ms měří proud procházející větrákem. Pokud je proud nižší/vyšší než výrobcem předepsaný, dojde k chybě a test je přerušen. [2]

2.3.6 LED Module Check

Mezi poslední snímaná data patří i samotný stav *LED Module Check*. Zde se především kontroluje stav systému, který popisuje objekt *STATUS,* v němž se ukládá výsledek autodiagnostiky měřícího nástroje. V případě chyby je zde chyba zaznamenána. [2]

3 SYSTÉM PŘEDÁVÁNÍ DAT

Aby bylo možné vytvořit webovou aplikaci pracující s velkým množstvím dat, je nutné navrhnout a realizovat systém předávání dat z testovacích stanic do databáze. Dále je nutné navrhnout a realizovat samotnou databázi, která by byla schopna data uchovat v plném rozsahu (bez ztrát dat) a zároveň by zamezila redundanci dat.

Systém předávání dat je možné realizovat dvěma způsoby:

1. Přímé připojení SW ovládající *LED Module Check* na serverovou databázi. Zde by ovšem mohlo docházet k výpadkům připojení na databázi, čímž by vznikal i problém nekompletních dat. Navíc funkce pro vkládání dat do databáze by musela být obsažena i na straně klienta, čímž by se objevil problém při aktualizaci této funkce, neboť právě tyto aktualizace by se neprojevily ihned, ale postupně až po aktualizaci celého SW klienta. Z tohoto důvodu by v jeden okamžik mohly existovat dvě verze této funkce přistupující do databáze, čímž by mohlo docházet ke kolizím. [10][11]
2. Předávání souborů na souborový server. Tento způsob je založen na uložení veškerých naměřených dat do souborů v předem domluveném formátu a posléze odeslání na souborový server. Odtud by si webová aplikace byla schopna převzít soubory a dále je zpracovávat (ukládat do databáze). Výhoda tohoto řešení je v relativní jednoduchosti, přehlednosti, absenci nutnosti přístupových práv do databáze a především v možnosti archivace souborů pro případné pozdější využití či kontrolu. [10][11]

Pro realizaci systému předávání dat byl vybrán 2. způsob přímo firmou AL.

3.1 Třídy a serializace v C#

SW pro ovládání *LED Module Check* je psán v *ASP.NET* za využití programovacího jazyka C# (více v kapitole 5 – ASP.NET C#). Pro přenos výše zmíněných dat byl vybrán formát XML, jenž je v C# přímo generován pomocí tzv. „serializace tříd“. [2][6]
Programovací jazyk C# pro realizaci specifických náročnějších objektů využívá „tříd“. Jedná se v podstatě o popis dat a funkcí v daném objektu. Některé třídy mohou obsahovat pouze popis funkcí objektu, jiné pouze popis dat v objektu. Záleží na konkrétním využití objektu. [12]
Pro lepší představu uvedu příklad – budeme chtít v programu používat data týkající se osob. Jelikož v C# jsou v základu obsaženy pouze jednoduché datové objekty, jako jsou například čísla, znaky, proud znaků atd., tak je nutné si nejprve jasně definovat třídu, kterou budeme v aplikaci používat pro uchovávání dat ohledně osob. Vytvoříme si tedy třídu s názvem *Osoba*, ve které budeme uchovávat tři datové objekty – jméno, věk

a pohlaví. Jméno nám bude reprezentovat datový typ *string*, což je proud znaků, který se používá pro klasický text. Věk bude reprezentovat datový tip *int*, což je číslo a pro pohlaví použijeme datový typ *bool*, který nabývá pouze dvou hodnot – *true* a *false*. Dá se taktéž vyjádřit jako 1 a 0. Pro nás si můžeme definovat, že *true* bude značit, že se jedná o ženu, *false* bude pro muže.
Tímto bychom měli jasně definované datové objekty v naší vytvořené třídě *Osoba*.

```
public class Osoba
{
    public string Jmeno;
    public int Vek;
    public bool Pohlavi;
}
```

Abychom mohli s třídou *Osoba* plně pracovat, je nutné ji doplnit o funkce. Mezi základní funkce patří *konstruktor* a *destruktor*. *Konstruktor* je funkce, která se volá při vytváření instance objektu dle dané třídy (jinými slovy – pokud si vytvoříme datový objekt jménem Petr a Petr bude *Osoba*, tak se v podstatě vytvořila instance objektu jménem Petr, který je typu *Osoba* a bude tak s ním zacházeno). Tento *konstruktor* může být bezparametrický (volá se bez parametrů, pokud není definováno jinak, tak se volá automaticky) nebo parametrický (volá se s parametry, například již při vytváření můžeme určit jméno, věk a pohlaví), viz níže. *Destruktor*, jak již název napovídá, je funkce, která se volá při rušení daného datového objektu. To se například hodí pří výpisu činností programu nebo při požadavku na určitou činnost programu při rušení objektu. Jak *konstruktor*, tak *destruktor* jsou vytvářeny automaticky, pokud nejsou ve třídě definovány. Doplníme tedy naši třídu *Osoba* o oba typy konstruktorů a destruktor:

```
public class Osoba
{
    public string Jmeno;
    public int Vek;
    public bool Pohlavi;
    // BEZPARAMETRICKY KONSTRUKTOR
    public Osoba()
    {
        Jmeno = "NEUVEDENO";
        Vek = 0;
        Pohlavi = false;
    }
    // PARAMETRICKY KONSTRUKTOR
    public Osoba(string i_Jmeno, int i_Vek, bool i_Pohlavi)
    {
        Jmeno = i_Jmeno;
        Vek = i_Vek;
        Pohlavi = i_Pohlavi;
    }
    // DESTRUKTOR
    ~Osoba()
    {
        Console.WriteLine(Jmeno + " byl odstranen");
    }
}
```

Jak je patrné ze zdrojového kód, třída nyní obsahuje jeden bezparametrický *konstruktor*, který přidělí instanci objektu vytvořeného bez parametrů jméno „NEUVEDENO“, nastaví věk na 0 a jako pohlaví nastaví *false* (muž). Dále třída obsahuje parametrický *konstruktor*, který nastaví datové objekty, které obsahuje třída, podle parametrů se kterými je třída vytvořena. Nakonec třída obsahuje *destruktor*, který pouze na příkazovou řádku (pro kterou je tento zkušební program/třída psána) vypíše jméno osoby s dodatkem „byl odstranen“. Pro zajímavost ještě doplníme třídu o funkci *PridejDesetLet*, která věk osoby navýší o 10 let.

```
public class Osoba
{
    public string Jmeno;
    public int Vek;
    public bool Pohlavi;
    // BEZPARAMETRICKY KONSTRUKTOR
    public Osoba()
    {
        Jmeno = "NEUVEDENO";
        Vek = 0;
        Pohlavi = false;
    }
    // PARAMETRICKY KONSTRUKTOR
    public Osoba(string i_Jmeno, int i_Vek, bool i_Pohlavi)
    {
        Jmeno = i_Jmeno;
        Vek = i_Vek;
        Pohlavi = i_Pohlavi;
    }
    // FUNKCE NA PRIDANI 10 LET
    public void PridejDesetLet()
    {
        Vek = Vek + 10;
    }
    // DESTRUKTOR
    ~Osoba()
    {
        Console.WriteLine(Jmeno + " byl odstranen");
    }
}
```

Nyní již pro naše účely máme plně hotovou a dostačující třídu *Osoba*. Pro ukázku si vytvoříme v hlavní části programu jednu instanci objektu třídy *Osoba* za pomocí bezparametrického *konstruktoru* a jednu instanci za pomocí parametrického *konstruktoru*. Poté zavoláme funkci *PridejDesetLet*.

```
static void Main(string[] args)
{
    // VYTVORENI INSTANCI OBJEKTU
    Osoba Prvni = new Osoba(); // BEZ PARAMETRU
    Osoba Druhy = new Osoba("Anna", 25, true); // S PARAMETRY

    // ZAVOLANI FUNKCE PridejDesetLet
    Prvni.PridejDesetLet();
    Druhy.PridejDesetLet();
}
```

Ihned po vytvoření instancí objektů mají naše objekty tyto hodnoty:

Tabulka 1 - Hodnoty instancí objektů po vytvoření

Název objektu	Jmeno	Vek	Pohlavi
Prvni	NEUVEDENO	0	false
Druhy	Anna	25	true

Po zavolání funkce *PridejDesetLet* mají naše objekty tyto hodnoty:

Tabulka 2 - Hodnoty instancí objektů po zavolání funkce "PridejDesetLet"

Název objektu	Jmeno	Vek	Pohlavi
Prvni	NEUVEDENO	10	false
Druhy	Anna	35	true

Po ukončení programu se na konzoli, ze které byl program spuštěn, vypíší tyto řádky:

Petr byl odstranen
NEUVEDENO byl odstranen

Třída *Osoba* je velmi jednoduchá a je možné ji dále rozšiřovat o datové objekty, funkce, přepisovat u ní systémové funkce (v podstatě bezparametrický *konstruktor* je přepsáním systémové funkce) atd., dokud by neodpovídala přesně našim požadavkům, ovšem pro náš účel takto napsaná třída plně dostačuje.
Aby byla ukázka kompletní, je nutné ukázat ještě proces serializace a následné deserializace objektu. Pro náš příklad si vybereme objekt *Druhy* a provedeme serializaci. Serializace je v podstatě to, že vezmeme aktuální instanci objektu (včetně aktuálního stavu) a provedeme konverzi objektu na *stream* (proud bytů) a následně uložíme do XML souboru. Vytvoříme si tedy samostatnou funkci Serializace s parametrem typu *Osoba,* která zajistí serializaci do souboru. Naštěstí jsou pro serializaci a deserializaci již vytvořené systémové funkce, tudíž nám stačí pouze tyto funkce zavolat s určitými parametry. [12]

```
static void Serializace(Osoba OsobaKSerializaci)
{
    // VYTVORENI INSTANCE XML SERIALIZERU K SERIALIZACI OBJEKTU DO XML FORMATU
    XmlSerializer serializer = new XmlSerializer(OsobaKSerializaci.GetType());

    // VYTVORENI "ZAPISOVACE" DO SOUBORU
    TextWriter writer = new StreamWriter("Osoba.xml");

    // STREAM ZE SERIALIZERU ULOZIME DO SOUBORU
    serializer.Serialize(writer, OsobaKSerializaci);

    // UKONCIME ZAPISOVANI DO SOUBORU
    writer.Close();
}
```

Zavolání této funkce v hlavní části programu pomocí příkazu:

```
Serializace(Druhy);
```

Po spuštění programu, doplněného o tuto funkci a volání funkce, se ve složce s programem vytvoří soubor *Osoba.xml*, který obsahuje tato data:

```
<?xml version="1.0" encoding="utf-8"?>
<Osoba>
  <Jmeno>Anna</Jmeno>
  <Vek>35</Vek>
  <Pohlavi>true</Pohlavi>
</Osoba>
```

Jak je vidět, uložený soubor doopravdy obsahuje data, která obsahoval objekt *Druhy*, který jsme předali jako parametr funkce, v momentě zavolání funkce. Proces deserializace je velice podobný, jen spočívá v opačném postupu. Nejprve se načte *stream* (proud bytů) ze souboru, podle třídy se provede deserializace a načtená data se uloží do objektu. Vytvořme si tedy bezparametrickou funkci *DeSerializace*, která nám bude vracet objekt typu *Osoba*:

```
static Osoba DeSerializace()
{
        // VYTVORENI NOVEHO OBJEKTU, KTERY FUNKCE VRATI
        Osoba NovaOsoba = new Osoba();

        // XML SERIALIZER K DESERIALIZACI OBJEKTU Z XML
        XmlSerializer serializer = new XmlSerializer(NovaOsoba.GetType());

        // NACTENI STREAMU ZE SOUBORU
        StreamReader reader = new StreamReader("Osoba.xml");

        // NAHRANI ZISKANYCH DAT Z DESERIALIZACE XML DO OBJEKTU
        NovaOsoba = (Osoba)serializer.Deserialize(reader);

        // UKONCIME CTENI ZE SOUBORU
        reader.Close();

        // VRATIME NACTENY OBJEKT
        return NovaOsoba;
}
```

Poté si v hlavní části programu vytvoříme nový objekt typu *Osoba* s názvem *Treti* a její hodnoty nastavíme podle výsledku deserializace již dříve vytvořeného souboru *Osoba.xml*, který obsahuje hodnoty z objektu *Druhy*, taktéž typu *Osoba*.

```
Osoba Treti = DeSerializace();
```

Po této operaci bude mít objekt *Treti* tyto hodnoty:

Tabulka 3 - Hodnoty instance objektu Treti po deserializaci

Název objektu	Jmeno	Vek	Pohlavi
Treti	Anna	35	true

Jak je vidět, objekt *Treti* obsahuje naprosto stejné hodnoty, které obsahoval objekt *Druhy* v okamžiku, kdy se provedla serializace jeho dat do souboru Osoba.xml. Proces serializace a deserializace byl tedy úspěšný.
Cílem této ukázky bylo názorně ukázat, jak se v programovacím jazyku C# navrhují třídy, jak se následně pracuje s instancemi objektů dané třídy a jak se provádí serializace/deserializace objektů, protože přesně tak, jak zde bylo zjednodušeně ukázáno, bylo postupováno při návrhu třídy, která slouží k přenosu dat a následné serializaci/deserializaci dat obsažených v navržené třídě.

3.2 Návrh třídy TestResult

Při návrhu třídy pro přenos dat výsledků měření (dále *TestResult*) jsme spolupracovali s pracovníky firmy AL včetně pana Petra Poláka, jehož bakalářská práce se týká právě doplnění SW ovládající *LED Module Check* o funkci exportu dat (serializace do XML souborů) na souborový server. Aby bylo možno zajistit funkční serializaci a deserializaci dat, je nutné, aby jak moje webová aplikace, tak funkce vytvořená panem Polákem, obsahovaly identickou třídu *TestResult*, podle které se tyto procesy vykonávají. Jakákoliv sebemenší odlišnost této třídy na straně klienta nebo severu může mít za následek nečitelnost dat uložených v XML.
Mezi hlavní požadavky na tuto třídu patřilo zachování veškerých naměřených dat (bezztrátovost), jasné odlišení jednotlivých fází a typů měření, přehlednost a zamezení redundancí dat. Poslední zmiňované ovšem u serializace do XML nelze zajistit, protože systémová serializace se provádí tak, jak data „leží a běží“, jinými slovy – pokud se v objektu budou některá data neustále opakovat, budou se tak i zapisovat do XML souboru, není zde žádná bezztrátová nebo ztrátová komprese. Tato vlastnost na jednu stranu způsobuje to, že XML soubory jsou zbytečně veliké a zabírají tak větší místo na disku, ale na druhou stranu právě díky nulové kompresi a proudovému zápisu dat jsou data uložená v XML souboru pro uživatele velmi čitelná, když si XML soubor otevře přímo v poznámkovém bloku, jak je uvedeno výše.

Základní rozvržení hlavní třídy **TestResult**:

Tato třída obsahuje základní data o použitém *LED Module Check*, testovaném LED modulu, názvu uživatele a PC, na kterém byl test prováděn, verzi SW, celkový STATUS výsledku měření (zda test dopadl v pořádku nebo s chybou) a výrobní čísla. To by ovšem k obsáhnutí veškerých dat, která byla během měření nasbírána, nestačilo. Proto třída *TestResult* obsahuje ještě další vnořené třídy. Označení následujících tříd symbolem **<L>** značí, že se jedná o *list*, což v je C# seznam objektů. Takto označené níže uvedené třídy se tedy v *TestResult* mohou objevit vícekrát, ne jen jednou. Toho se využívá, pokud testovaný modul má více skupin LED (například LED modul obsahující LED pro tlumené světlo, dálkové světlo, rohové světlo, blinkr…) nebo více NTC termistorů atd.

- **AmbTemperature** – třída obsahující pouze údaj o okolní teplotě během měření a o přesném čase změření okolní teploty. Teplota je zaznamenána na začátku testu.
- **HWResult** – třída obsahující výsledek autodiagnostiky měřícího nástroje *LED Module Check*, jeho konfiguraci a výrobní číslo.
- **<L>LedResult** – třída obsahující název testované série LED, dále celkový STATUS tohoto testu, případné chyby, binning dané série LED a další limity platné pro danou sérii LED. Třída obsahuje ještě samostatný list tříd *LedValue*, kde jsou uloženy hodnoty napětí a proudu v daný časový okamžik.
- **<L>TempResult** – třída obsahující informace o měřící NTC teploměru (název, limity, výsledky jednotlivých teplotních testů). Naměřené hodnoty teploty daného NTC teploměru obsahuje list tříd *TempValue*, kde jsou uchovány hodnoty teploty v čase.
- **FanResult** – třída obsahující záznam o proudu procházející větráčkem v daný čas a maximální/minimální proud, který může větráčkem procházet. Do této třídy se uloží buď první dobrá hodnota nebo poslední špatná.
- **<L>MotorResult** – třída obsahující data z měření a informace o krokovém motůrku. Naměřená data se ukládají do listu tříd *MotorPos*. V podstatě se jedná pouze o pozice motůrků a proud procházející motůrkem v daný časový okamžik.
- **<L>LightDistridutionTest** – tato třída byla zatím definována pouze formálně, v současnosti není nijak využívána, je připravena na budoucí využití. Bude se jednat o výsledky vizuální kontroly světelného toku LED modulu.
- Dále byl definován list třídy **TechnoTeamResult**, ovšem v požadavku není uvedena povinnost evidovat tento list tříd v databázi, proto jsem s ním dále již nepracoval.

Pro konkrétnější představu o výše nastíněné třídě *TestResult* přikládám její definici:

```
public class TestResult
{
    public string moduleName = string.Empty;
    public string moduleDescription = string.Empty;
    public string moduleTTNr = string.Empty;
    public string moduleSN = string.Empty;
    public string moduleNewTTNr = string.Empty;
    public string moduleNewSN = string.Empty;
    public bool newDmcLabelVerificationFlag = false;
    public string userName = string.Empty;
    public string computerName = string.Empty;
    public string swVersion = string.Empty;
    public DateTime Timestamp = DateTime.MinValue

    public STATUS result = new STATUS();

    public AmbTemperature ambientTemperature = new AmbTemperature();
    public HWResult hwResult = new HWResult();
    public List<LedResult> ledResults = new List<LedResult>();
    public List<TempResult> teperatureResults = new List<TempResult>();
    public FanResult fanResult = null;
    public List<MotorResult> motorResults = new List<MotorResult>();
    public List<LightDistridutionTest> LightDistridution = null;
    public VisualInspection VisualInspection = null;
    public List<TechnoTeamResult> technoTeamResults = new
                                                List<TechnoTeamResult>();
}
```

Určitě stojí za povšimnutí, že třída *TestResult* neobsahuje žádné funkce, *konstruktory* či *destruktor*. Důvod této absence je ten, že u této třídy je prostě nepotřebujeme. Jak SW na straně klienta, tak na straně serveru se přistupuje k jednotlivým položkám objektu přímo a nastavuje/čte si je dle potřeby. Ani žádné náročnější operace nad objekty této třídy nejsou prováděny, na straně klienta se provádí pouze zápis dat do objektů této třídy a na straně severu se provádí čtení dat z objektů této třídy. Jedná se tak pouze o transportní typ třídy, proto není potřeba žádných funkcí.
Pro přehlednost přikládám deklarace všech tříd obsažených v *TestResult*, se kterými dále pracuji, v příloze č.1 a v příloze č.2. Struktura těchto tříd sloužila k přímému návrhu databáze, jakákoliv odchylka může znamenat ztrátu dat a následné chybné zobrazení dat či vyhodnocení jednotlivých měření.

4 DATABÁZE

Typ databáze, ve které budou data uložena, byl již dopředu zadán – *Microsoft SQL* (taktéž označován jako MSSQL). Jedná se o relační databázový systém založený na architektuře klient – server.

4.1 SQL a ER diagram

SQL je standardizovaný strukturovaný dotazovací jazyk, který je využíván právě v relačních databázových systémech pro práci s daty. Jeho historie sahá až do 70. let minulého století a v současnosti se využívá ve většině databázových systémů. [4]
Pro ukázku, stejně jako v C#, si uvedeme jednoduché příklady. Použijme opět problém tykající se dat osob (jméno, věk, pohlaví). Abychom tato data mohli v databázi uchovávat, musíme nejprve vytvořit tabulku, jejíž datové položky budou odpovídat požadovaným parametrům. Takovou tabulku vytvoříme tímto příkazem:

```
CREATE TABLE [dbo].[Osoby]
(
	[Jmeno] VARCHAR(50) NOT NULL,
	[Vek] INT NULL,
	[Pohlavi] BIT NULL
)
```

Tento příkaz provede to, že v databázi *dbo* vytvoří tabulku jménem *Osoby* a v ní budou datové položky *Jmeno* (maximálně 50 znaků), *Vek* (číslo) a *Pohlavi* (bit – true, false). Pokud do této tabulky začneme nahrávat data, může dojít k tomu, že hodnoty dvou záznamů se budou shodovat, čímž nebudeme schopni naprosto jasně identifikovat daný záznam. Ukázka takové situace:

Tabulka 4 - Data tabulky Osoby bez Id

Jmeno	Vek	Pohlavi
David	25	False
Jana	26	True
David	25	False

V tento okamžik nejsme schopni přesně identifikovat, zda [David, 25, False] je první záznam, nebo třetí. Proto se v databázích používá tzv. identifikační klíč (také identifikátor), jenž je značen *Id* a je reprezentován jako číslo. Většinou reprezentuje pořadí záznamu v tabulce a díky němu jsme schopni naprosto jasně identifikovat záznam v tabulce. SQL příkaz včetně *Id* by vypadal následovně:

```
CREATE TABLE [dbo].[Osoby]
(
	[Id] INT NOT NULL PRIMARY KEY,
	[Jmeno] VARCHAR(50) NULL,
	[Vek] INT NULL,
	[Pohlavi] BIT NULL
)
```

Označením „`[Id] INT NOT NULL PRIMARY KEY`“ jsme určili, že pro tuto tabulku je primární klíč (taktéž identifikační klíč) položka jménem *Id*. Tato položka nesmí být prázdná, nesmí se opakovat a je využívána k identifikaci zápisu v databázi. Podívejme se opět na předchozí případ zápisu stejných dat, nyní ovšem s *Id*.

Tabulka 5 - Data tabulky Osoby s ID

Id	Jmeno	Vek	Pohlavi
0	David	25	False
1	Jana	26	True
2	David	25	False

Nyní již jsme schopni naprosto jasně určit, že data [0, David, 25, False] patří prvnímu zápisu v tabulce a data [2, David, 25, False] patří třetímu zápisu v tabulce.
Do tabulky nemusíme zapisovat pouze konkrétní data, ale můžeme využít tzv. cizích klíčů (taktéž referenčních klíčů), které budou ukazovat na další zápis, z pravidla v jiné tabulce. Pro náš příklad si řekněme, že budeme evidovat město, ve kterém daná osoba bydlí. Protože se město může velmi často opakovat a my chceme zabránit duplicitě dat, využijeme cizího klíče. Nejdříve si znova vytvoříme tabulku *Osoby*, tentokrát již s cizím klíčem ukazující na zápis v tabulce *Mesta*, kterou si následně vytvoříme. *Osoby*:

```
CREATE TABLE [dbo].[Osoby]
(
	[Id] INT NOT NULL PRIMARY KEY,
	[Jmeno] VARCHAR(50) NULL,
	[Vek] INT NULL,
	[Pohlavi] BIT NULL,
	[Id_Mesto] INT NULL,
CONSTRAINT [FK_Osoby_Mesta] FOREIGN KEY ([Id_Mesto]) REFERENCES [Mesta]([Id])
)
```

Nyní máme vytvořenou tabulku *Osoby*, která má navíc položku *Id_Mesto*, která nám bude jednoznačně ukazovat na zápis v tabulce *Mesta* podle *Id*. Ještě je nutné si vytvořit tabulku *Mesta*, která pro zjednodušení bude obsahovat pouze položky *Id* a *Mesto* (maximálně 50 znaků):

```
CREATE TABLE [dbo].[Mesta]
(
	[Id] INT NOT NULL PRIMARY KEY,
    [Mesto] VARCHAR(50) NULL
)
```

Tabulky bychom tímto měli vytvořené. Nahrajme si do nich data a ukažme si, jak funguje reference z jedné tabulky na druhou:

Tabulka 6 - Data tabulky Osoby s ID a s referemcí

Id	Jmeno	Vek	Pohlavi	Id_Mesto
0	David	25	False	0
1	Jana	26	True	0
2	David	25	False	1

Tabulka 7 - Data tabulky Mesta

Id	Mesto
0	Jihlava
1	Brno

Jak můžeme vidět, název města se nám neopakuje, ačkoliv dvě osoby žijí ve stejném městě.
Tímto bychom měli hotové základní vytváření tabulek pomocí SQL. Nyní je na řadě samotné dotazování na data. Dotaz, jako takový, se velmi podobá lingvistické formě. Například pokud budeme chtít vybrat všechna data z tabulky Osoby, řekli bychom *„Vyber vše z Osoby“*, v angličtině *„Select all from Osoby“* a takový příkaz v SQL vypadá následovně: [4][9]

```
SELECT * FROM Osoby
```

Tento dotaz nám vrátí v podstatě to, co vidíme v Tabulce 6. Pokud budeme chtít konkrétnější údaje, pak náš dotaz můžeme podmínit. Například pokud budeme chtít pouze ty osoby, které jsou starší více než 25 let:

```
SELECT * FROM Osoby WHERE (Vek > 25)
```

Po tomto dotazu by se nám vrátil pouze záznam s *Id=1*. Podmínky lze samozřejmě různě řetězit a můžeme i specifikovat, jaké položky ze záznamů odpovídajících dotazu požadujeme. Například pokud budeme chtít všechna jména mužů z databáze, dotážeme se tímto SQL dotazem:

```
SELECT Jmeno FROM Osoby WHERE (Pohlavi = False)
```

Navrácené hodnoty databází:

Tabulka 8 - Vrácené hodny po SQL dotazu

Jmeno
David
David

Námi vytvořená databáze je velmi jednoduchá a přehledná, ovšem v praxi tento případ málokdy nastává. Proto se využívá ER diagramu, který slouží ke znázornění, jak je celá databáze koncipována – jaké vztahy a mezi jakými tabulkami existují a jaké položky jsou v jednotlivých tabulkách obsaženy. V praxi zavedený postup je ten, že se nejdříve vytvoří základní ER diagram (tabulky, vztahy), po validaci se jednotlivé parametry konkretizují a nakonec dochází k samotnému generování SQL příkazu na vytvoření databáze přesně podle ER diagramu. Nyní si takový postup ukážeme na našem příkladu. [4][9]

Pro vytvoření ER diagramu použijeme SW nástroj *MySQL Workbench* (dostupný z https://www.mysql.com/products/workbench/), který umožňuje po vytvoření diagramu i generování SQL příkazů pro databázi, která odpovídá diagramu. Bohužel notace těchto SQL příkazů se mírně liší od té, kterou používá MSSQL, ovšem po mírné úpravě lze SQL příkazy přepsat, aby vyhovovaly MSSQL. Nejprve si tedy určíme, jaká data chceme v databázi ukládat. Rozhodneme se pro základní data osob a měst, ve kterých žijí. Umístíme tedy do ER diagramu dvě tabulky a pojmenujeme je:

Schéma 1 - Tvorba ER diagramu, krok 1

Tabulky jsou nyní vytvořeny, avšak jsou prázdné, proto do začátku do obou políček doplníme identifikační klíč (*Id*):

Schéma 2 - Tvorba ER diagramu, krok 2

Jak lze vidět, tabulky již obsahují datové položky. To, že se jedná o identifikační klíč, poznáme z ikonky zlatého klíče. Tímto bychom měli základní datové struktury hotové. Nyní si musíme určit, jaké vztahy mezi těmito tabulkami (entitami) budou. Víme, že v tabulce *Osoby* určitě chceme odkazovat na město, ve kterém daná osoba žije. Opačný případ (že bychom v tabulce *Mesta* odkazovali na nějakou osobu) není potřeba, tudíž budeme mít pouze jeden vztah. Jeho typ ale musíme konkretizovat – jestli se jedná o povinný vztah a jestli je typu 1:1, 1:N, N:1 nebo N:M. [4]
Povinnost vztahu se určuje tak, že si řekneme, jestli je doopravdy nutné, aby daný vztah existoval. V našem případě záleží čistě na nás, jestli tento vztah vyžadujeme, jestli nám nevadí osoba bez uvedeného města, ve kterém bydlí, nebo naopak vyžadujeme uvádět toto město. Co se týče grafického značení, tak povinný vztah se značí nepřerušovanou čarou a nepovinný vztah přerušovanou. Určeme si tedy, že náš vztah bude nepovinný. [4]

Druhý parametr vztahu záleží na logice dat, která takto chceme ukládat. Musíme tedy určit, zdali se jedná o typ: [4]

- **1:1** – jedná se o jakýsi exkluzivní typ vazby – záznam v tabulce 1 může odkazovat pouze na jeden záznam v tabulce 2, a na tento záznam v tabulce 2 může být odkazováno pouze z jednoho záznamu z tabulky 1. V praxi lze uvést manželství – manžel může mít pouze jednu manželku a manželka může mít pouze jednoho manžela. [4]
- **1:N** – v tomto typu vztahu jeden záznam z tabulky 1 může odkazovat na více záznamů v tabulce 2, ale na konkrétní záznam v tabulce 2 může být odkazováno pouze z jednoho záznamu z tabulky 1. Příklad z praxe – vlastnictví vozidla. Člověk může vlastnit několik vozidel, ale jedno vozidlo může být vlastněno pouze jedním člověkem. [4]
- **N:1** – velmi podobný případ předchozímu, pouze jsou strany prohozené. Zápis z tabulky 1 může odkazovat pouze na jeden zápis z tabulky 2, ale na zápis v tabulce 2 může odkazovat vícero zápisů z tabulky 1. Příklad z praxe – parkoviště. Automobil může parkovat pouze na jednom parkovišti, ale na parkovišti může být vícero automobilů. [4]
- **N:M** – typ, kdy zápis z tabulky 1 odkazuje na více zápisů z tabulky 2 a na zápis z tabulky 2 může odkazovat více zápisů z tabulky 1. Příklad z praxe – kino a film. Kino může hrát více filmů a film může být hrán ve více kinech.

Pokud si tedy řekneme, že jedna naše osoba může žít pouze v jednom městě, ale v jednom městě žije více osob, pak je typ našeho vztahu jasný – N:1. Nyní tedy náš ER diagram doplníme o nepovinný vztah typu N:1 a tím se nám v tabulce Osoby objeví nepovinný cizí klíč *Id_mesta*:

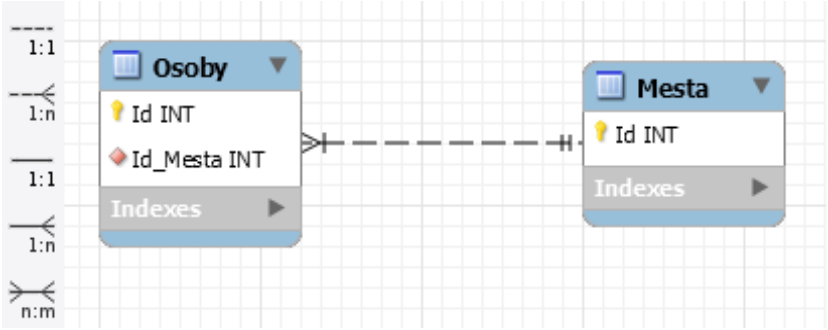


Schéma 3 - Tvorba ER diagramu, krok 3 + grafické značení vztahů

To, zdali se jedná o vztah 1:1, 1:N, N:1 nebo N:M, poznáme podle zakončení grafického značení vztahu. Pokud je u dané tabulky jednoduché zakončení, značí to vztah typu 1. Pokud je u tabulky jakýsi „*zobáček*“, jedná se o vztah typu N. Z ER diagramu můžeme tedy vyčíst, že se doopravdy jedná o vztah N:1. [4]

Po této fázi by měla přijít na řadu validace, zda tato struktura (model) vyhovuje našim podmínkám. V praxi se používá postup, při kterém se navozují různé možné situace a ověřuje se, zda by je byla databáze schopna pokrýt. V našem případě je návrh databáze v pořádku. Po této validaci již jen zbývá náš ER diagram doplnit o datové položky:

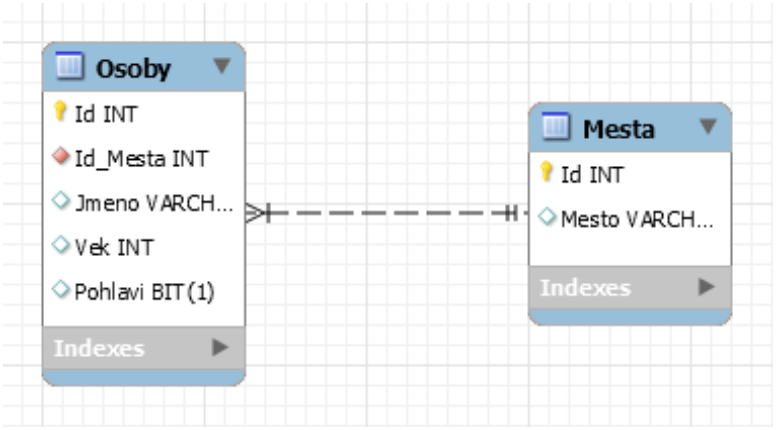


Schéma 4 - Tvorba ER diagramu, krok 4

Takto tedy vypadá konečný ER diagram naší databáze. Pokud si necháme vygenerovat SQL příkaz, získáme tento kód:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Mesta` (
  `Id` INT NOT NULL,
  `Mesto` VARCHAR(45) NULL,
  PRIMARY KEY (`Id`))

CREATE TABLE IF NOT EXISTS `mydb`.`Osoby` (
  `Id` INT NOT NULL,
  `Id_Mesta` INT NOT NULL,
  `Jmeno` VARCHAR(45) NULL,
  `Vek` INT NULL,
  `Pohlavi` BIT(1) NULL,
  PRIMARY KEY (`Id`),
  INDEX `fk_Osoby_Mesta_idx` (`Id_Mesta` ASC),
  CONSTRAINT `fk_Osoby_Mesta`
    FOREIGN KEY (`Id_Mesta`)
    REFERENCES `mydb`.`Mesta` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Jak si lze povšimnout, kód se mírně liší od námi vytvořeného. Místo hranatých závorek používá apostrofy a jinak značí identifikační (primární) klíč. To lze ovšem pomocí funkce „*najít a nahradit*“ opravit a zbylé úpravy udělat ručně. Jinak se ovšem vygenerovaný kód je principiálně stejný, jako námi vytvořený. Tímto jsme si ukázali základy navrhování databáze pomocí SQL a relačního modelu.

4.2 Návrh ER diagramu

Při návrhu ER diagramu celé databáze, která slouží k ukládání a čtení naměřených dat, jsem vycházel z již zmiňované třídy *TestResult* (příloha č. 1), tříd které tato třída využívá a třídy *STATUS* (příloha č. 2). Samotnou třídu *TestResult* jsem si určil jako výchozí bod (tabulku) a od ní jsem přidával postupně další tabulky a vztahy.
U každé třídy jsem postupoval tak, že jsem si nejprve určil, na jaké další třídy obsahuje reference a jak je tato reference deklarována. Pokud byla reference deklarována jako samostatný ukazatel na objekt dané třídy, pak jsem vztah mezi aktuální tabulkou a odkazovanou tabulkou určil jako 1:1. Pokud byla reference deklarována v listu (může zde být uloženo vícero referencí), pak jsem typ vztahu určil jako 1:N.
Bohužel se typ vztahu N:1 či M:N z definice tříd nedá určit, ale při pohledu na definice tříd jsem si uvědomil, že všechny třídy kromě *HWResult, MotorResult, Limits* a *STATUS* obsahují položku typu *DateTime*, jedná se v podstatě o časový otisk přesný na milisekundy. Z praktického hlediska tak je vyloučena možnost, že jeden zápis v tabulce s tímto otiskem by byl shodný s jakýmkoliv jiným zápisem. Nevzniká tak možnost vzniku redundantních dat a tím ani potřeba vztahu N:1 či M:N. U výše uvedených tříd, bez časového otisku, lze rovnou vyloučit tabulku *MotorResult*, která sice nemá časový otisk, ale tabulky, na které dále odkazuje, tento otisk mají a navíc se jedná o vztah 1:N, tudíž je možnost shody opět vyloučena. Zůstávají pouze tabulky *HWResult, Limits* a *STATUS*, zde by vztah N:1 nebo M:N mohl nastat.
Z tohoto pohledu je ale nejdůležitější třída *STATUS*, neboť se na ní odkazuje téměř ve všech ostatních třídách. Jedná se v podstatě o třídu, která je schopna popsat stav instance objektu. Kvůli tomuto důvodu hrozí největší možnost vzniku redundantních dat a je nutné vytvořit mechanismy k ošetření této možnosti.
Co se týče povinnosti či nepovinnosti vztahů mezi entitami, tak jsem všechny vztahy již dopředu určil jako nepovinné, protože mohou nastat případy, kdy například daný objekt třídy *TestResult* nebude obsahovat referenci na objekt třídy *LedResult*, ačkoliv se jedná o stěžejní část naměřených hodnot. Tato situace například může nastat, pokud diagnostika vyhodnotí používaný *LED Module Check* jako chybový a nespustí tak test připojeného LED modulu. Tímto opatřením (nepovinnosti vztahů) se tak do databáze zapíší i „nekompletní údaje“ z měření a v případě potřeby je lze dohledat.
Po vytvoření základních definic entit a vztahů jsem musel doplnit entity o datové položky. V podstatě se jednalo o kopírování datových položek z definic tříd do jednotlivých tabulek. Zde ovšem vznikly dva problémy – MSSQL neobsahuje datové typy *bool* a *float*. *Bool* jsem nahradil typem *bit*, který taktéž, jako *bool*, nabývá pouze hodnot 0/1 (*false, true*). Datový typu *float* jsem musel nahradit datovým typem *double*, protože má větší rozsah a přesnost než *float*, nebude tak docházet ke zkreslování dat, datová náročnost je ovšem mnohem větší. Po konzultaci s panem Ing. Radovanem Holkem, CSc. jsem navíc veškeré tabulky doplnil o *Id*, ačkoliv *Id* nemusely obsahovat, zvýší se tím možnost dodatečných úprav nad daty v databázi.

Návrh kompletního ER diagramu databáze, kterou bude serverová aplikace využívat, vypadá následovně:

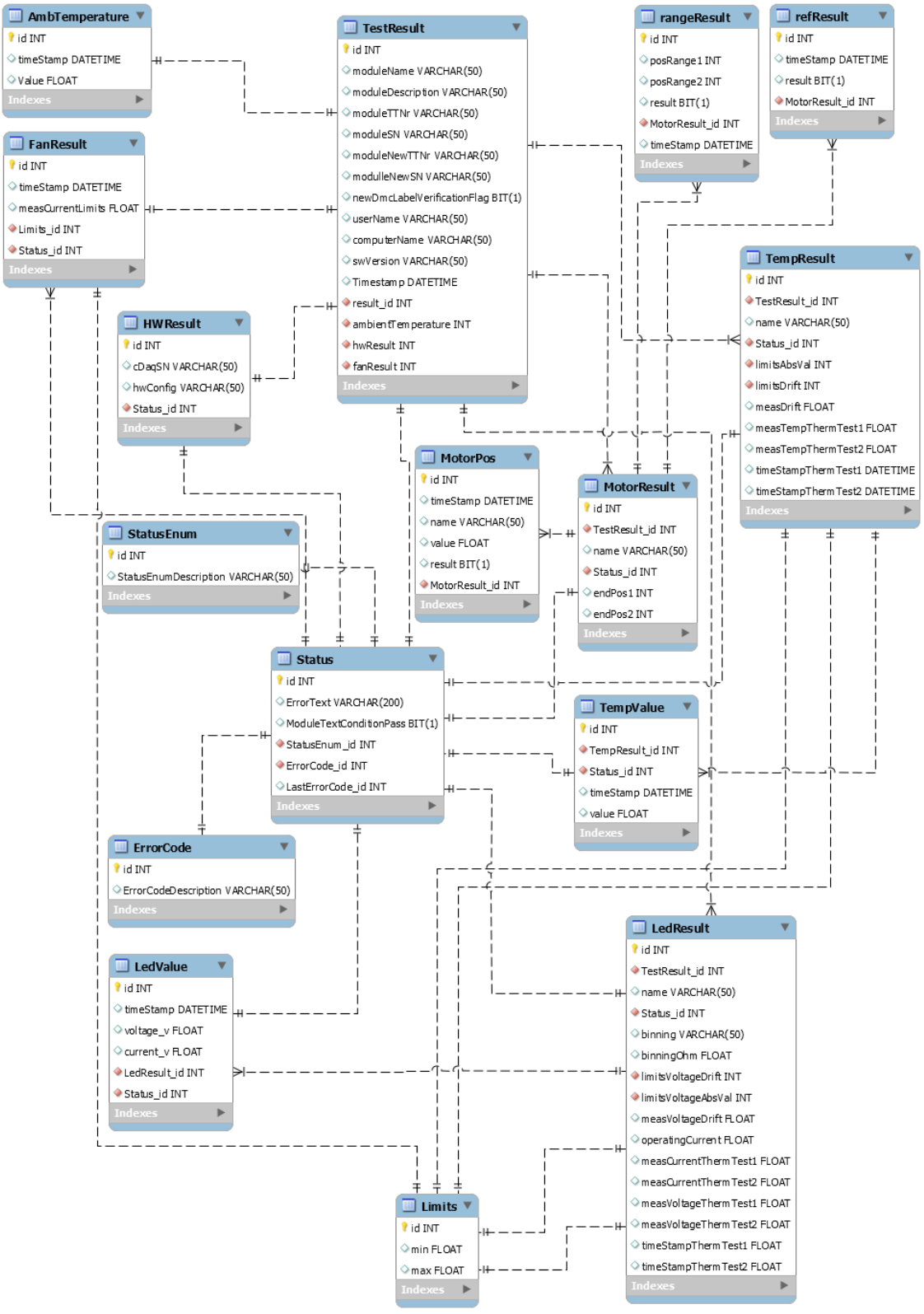


Schéma 5 - ER diagram databáze, verze 1

Uvedený ER diagram možná na první pohled působí chaoticky, k čemuž nejspíše přispěl i fakt, že v programu *MySQL Workbench* není možné kreslit cestu vykresleného vztahu, pouze již vykreslené cesty lze mírně upravovat. Pokud ovšem budeme postupovat od tabulky *TestResult*, tak zjistíme, že se jedná o velmi jednoduchou strukturu databáze.

- **TestResult** – tabulka se základními informacemi o měření (kdy bylo provedeno, výrobní číslo LED modulu, typ modulu…). Tato tabulka je dále spojena vztahem s dalšími sedmi tabulkami, každá z těchto tabulek pokrývá každou část měření (viz kapitola 2.3 - Měřené veličiny).
- **STATUS** – tato tabulka uchovává celkový výsledek (STATUS) měření, či hodnoty. Odkazuje dále na:
  - **StatusEnum –** tabulka se slovním určením stavu statusu
  - **ErrorCode –** tabulka se slovním vyhodnocením chyby
- **AmbTemperature** – tabulka obsahující informace o okolní teplotě a časový otisk, kdy byla tato okolní teplota změřena.
- **FanResult** – tabulka obsahující záznam změřeného proudu procházející větráčkem aktivního chlazení LED modulu, limit tohoto proudu, časový otisk a STATUS tohoto měření proudu. Dále odkazuje na:
  - **STATUS**
  - **Limits -** Tabulka obsahující maximální a minimální hodnoty
- **HWResult** – v této tabulce se uchovávají informace o použitém *LED Module Check*, jedná se o jeho výrobní číslo, nastavení HW a STATUS, což je v podstatě výsledek autodiagnostiky. Dále odkazuje na:
  - **STATUS**
- **LedResult** – v této tabulce se uchovává jméno (označení) testované skupiny led, binning LED, výsledek termálního testu atd. Dále je ve vztahu s:
  - **STATUS**
  - **LedValue –** Tabulka k uchování hodnoty naměřeného napětí, proudu a časový otisk, kdy přesně byly tyto hodnoty naměřeny.
  - **Limits**
- **MotorResult** – tato tabulka uchovává informace o provedených testech motůrků LED modulů. Dále je ve vztahu s:
  - **rangeResult –** tabulka obsahující výsledek rozsahového testu motůrků.
  - **refResult –** v této tabulce je výsledek referenčního testu motůrků.
  - **MotorPos –** tabulka s uloženými pozicemi motůrků s časovým otiskem.
- **TempResult –** v této tabulky se ukládají základní informace a výsledky z měření teploty LED modulu (název NTC které snímalo teplotu, výsledek teplotního driftu…). Dále ve vztahu s:
  - **Limits**
  - **STATUS**
  - **TempValue –** tabulka sloužící k ukládání hodnoty naměřené teploty a časového otisku.

Při pozdějším programování aplikace jsem zjistil, že tabulky *StatusEnum* a *ErrorCode*, které jsou u tabulky *STATUS*, mohu odstranit, protože jejich hodnoty obsahuje již třída *STATUS* sama o sobě v definici, proto jsou jejich hodnoty redundantní. Upravený ER diagram vypadá takto:

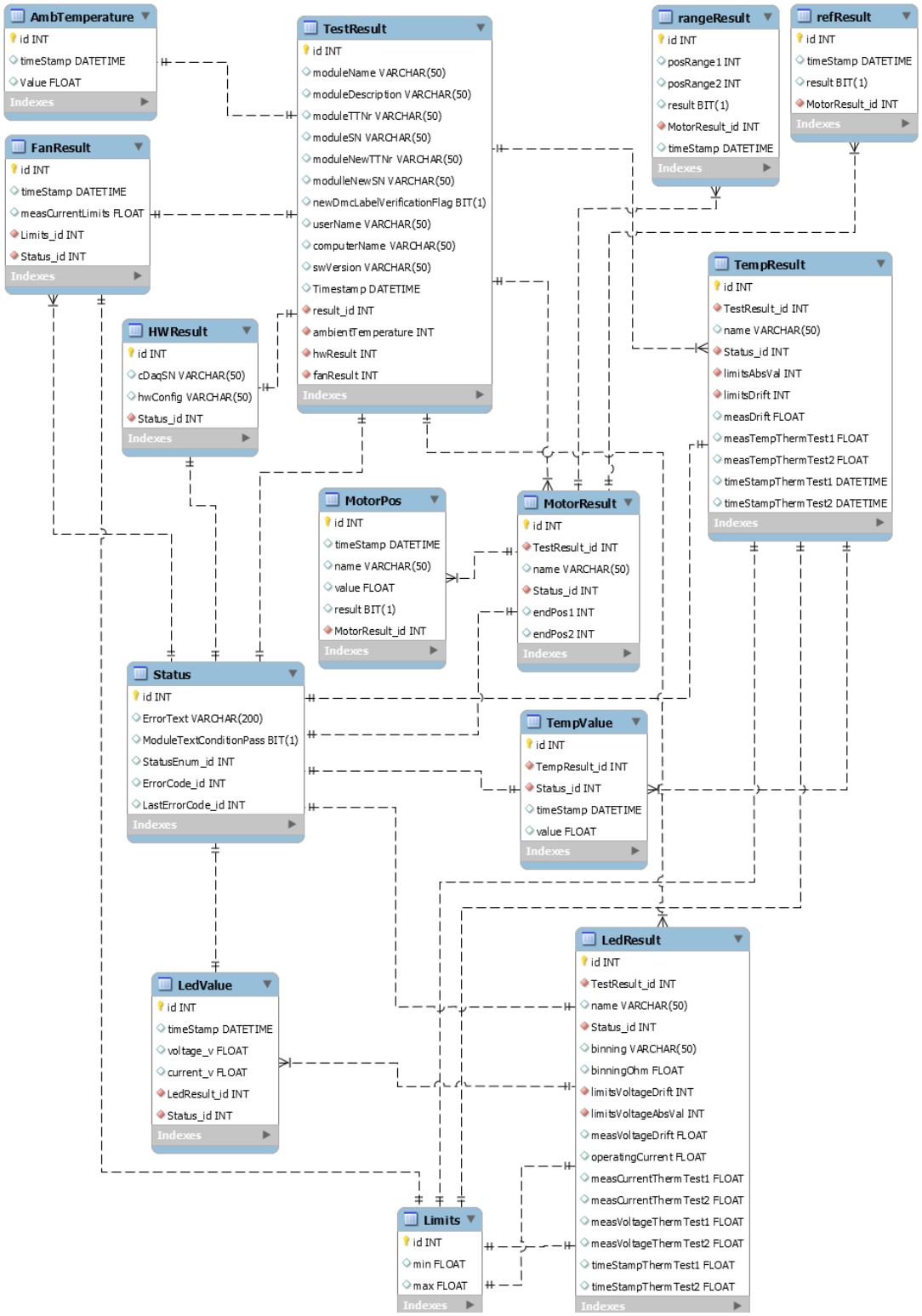


Schéma 6 - ER diagram databáze, verze 2

V tento okamžik jsem měl ER diagram databáze, která přesně odpovídala mým požadavkům na pokrytí všech dat vyskytujících se ve třídě *TestResult* a tříd, na které odkazuje. Dalším krokem bylo vytvoření SQL příkazů. Jak jsem již zmínil *MySQL Workbench* bohužel vytváří SQL příkazy, kde je mírně jiná notace. Například tabulku *MotorPos* mi vygeneroval nástroj následovně:

```
CREATE TABLE `TestResults`.`MotorPos` (
  `id`              INT          NOT NULL,
  `timeStamp`       DATETIME     NULL,
  `name`            VARCHAR(50)  NULL,
  `value`           FLOAT        NULL,
  `result`          BIT(1)       NULL,
  `MotorResult_id`  INT          NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_MotorPos_MotorResult1` FOREIGN KEY (`MotorResult_id`)
REFERENCES `TestResults`.`MotorResult` (`id`)
)
```

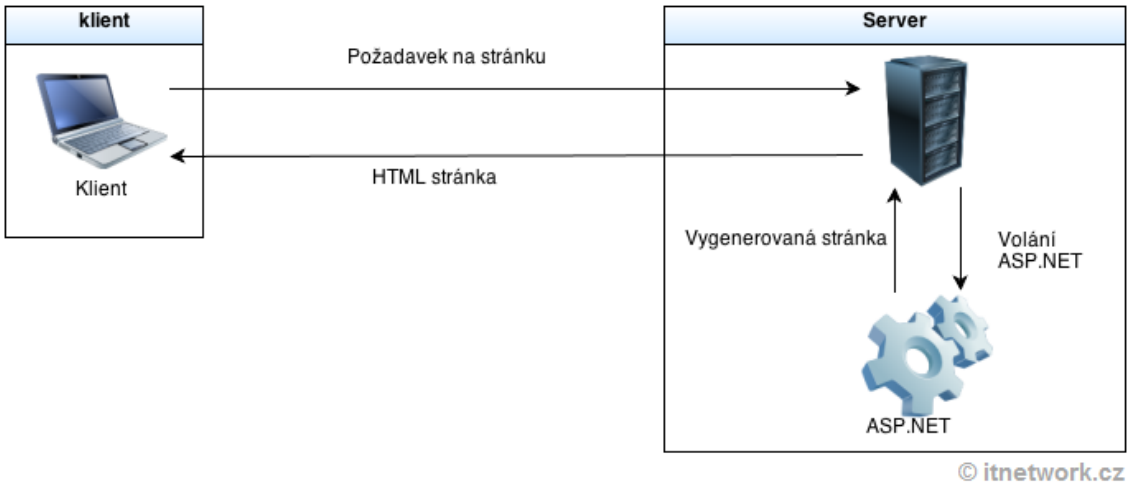
Po úpravách vypadá SQL pro MSSQL příkaz takto:

```
CREATE TABLE [dbo].[MotorPos] (
    [id]             INT          IDENTITY NOT NULL,
    [timeStamp]      DATETIME     NULL,
    [name]           VARCHAR (50) NULL,
    [value]          FLOAT        NULL,
    [result]         BIT          NULL,
    [MotorResult_id] INT          NOT NULL,
    PRIMARY KEY CLUSTERED ([id] ASC),
    CONSTRAINT [fk_MotorPos_MotorResult1] FOREIGN KEY ([MotorResult_id])
REFERENCES [dbo].[MotorResult] ([id])
)
```

Po této úpravě jsem tedy získal SQL příkaz pro kompletní vytvoření databáze v MSSQL. V tomto okamžiku jsem měl kompletně vše k návrhu databáze připravené a mohl jsem začít vytvářet samotnou serverovou aplikaci.
Při dalším postupu během programování dále vznikl požadavek na vytvoření „roletky“ pro rychlejší vyhledávání v databázi. Roletka je webový prvek, který se po kliknutí „rozbalí“ a umožní výběr z již definovaných hodnot. Abych mohl pro „roletku“ připravit vhodná data, vytvořil jsem dále tabulku *ModuleNameEnum*, která má pouze dvě datové položky – *id* a *moduleName*. Tato tabulka slouží pouze pro roletku, do hlavního ER diagramu databáze nijak nezasahuje. Této tabulce se budu později více věnovat.

5 ASP.NET C#

V požadavku pro vytvoření aplikace bylo uvedeno, že aplikace musí být napsána v *ASP.NET* s využitím programovacího jazyka C#. *ASP.NET* je webový framework, jinak řečeno – jedná se o sadu knihoven umožňující tvorbu webových aplikací v jazyce C# nebo Visual Basicu od Microsoftu. Tato technologie je založena na principu klient – server. Programová část probíhá na straně serveru a ke klientovi se již dostává jen výstup v podobě HTML (značkovací jazyk, díky kterému webový prohlížeč je schopen sestavit grafickou podobu webové stránky), jedná se tedy o dynamický web (viz Obrázek 10). [7]



Obrázek 10 - Komunikace klient/server ASP.NET [7]

Aby bylo možné lépe porozumět principu, tak uvedu příklad – uživatel si chce zjistit na webové stránce například informace o automobilu Škoda Octavia a klikne na tlačítko s odkazem na tyto informace. Tento odkaz má například podobu *www.auta.cz/Skoda-Octavia-I.* Kliknutím na toto tlačítko se na server odešle požadavek na zobrazení této webové stránky. Server po obdržení požadavku zavolá *ASP.NET*, které se nejprve podívá, co klient požaduje (v našem případě požaduje informace o Škodě Octavii I), posléze se připojí k databázi, načte požadovaná data, vygeneruje webovou HTML stránku a prostřednictvím serveru odešle webovou stránku klientovi. Ten vidí již jen statickou stránku, která ovšem byla dynamicky vytvořena. [7]
K programování je tedy nutná znalost jak programovacího jazyka C#, tak i alespoň základní znalost hypertextového značkovacího jazyka HTML. V *ASP.NET* se nabízí 2 způsoby, jak vytvářet webové stránky: [7]

- **WebForms –** jedná se o jakousi snahu přenést WinForms (standardní aplikace pro desktop s platformou Windows) na webové rozhraní. Při využívání tohoto způsobu se v designeru stránky poskládají veškeré prvky stránky z *controls* (tlačítka, popisky, tabulky, textová pole…), které nabízí toolbox. Aplikace se tak pro uživatele jeví jako desktopová, ovšem na jejím pozadí je složitější logika. Úkol WebForms, simulovat desktopovou aplikaci na webu, je velmi náročný,

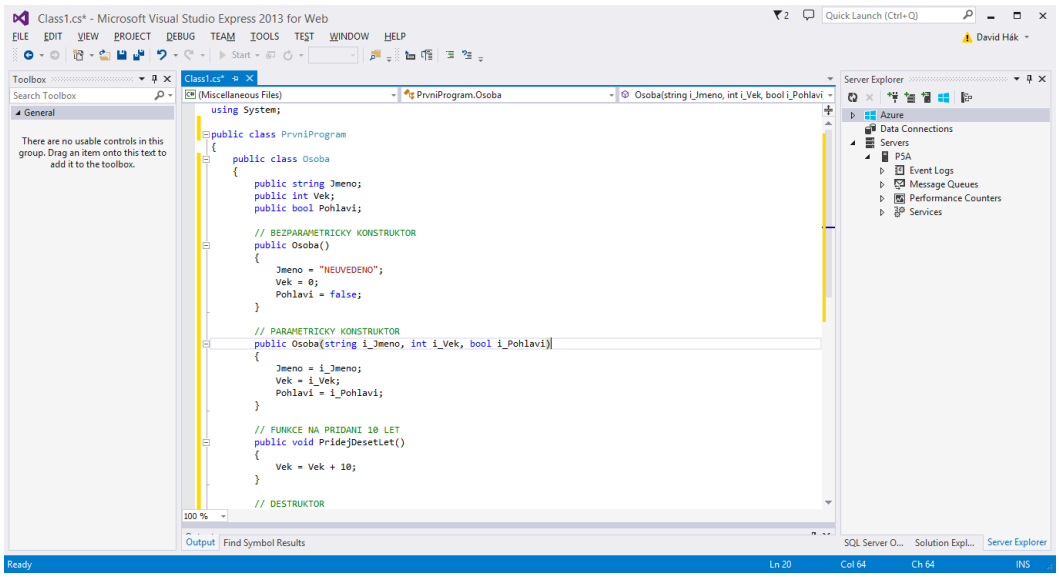
neboť protokol HTTP, který se využívá pro přenos obsahu webu, je bezstavový (nepřenáší se hodnoty, které uživatel zadal do textových políček, jaké vybral datum atd.). Server tedy vůbec neví, jaké hodnoty uživatel, jako klient, zadal. Veškeré stavy zde musí udržovat samotné *ASP.NET* pomocí ViewState – jedná se o skryté formulářové pole, které na výsledné HTML stránce uchovává stav aplikace a tento stav odesílá jej na server při dalším dotazu. Server podle ViewState je schopen si určit, v jakém stavu by měla aplikace být (pokud by běžela na straně klienta), provede požadované akce, vygeneruje výsledné HTML, nový ViewState a odesílá klientovi. Výhodou tohoto postupu je, že je velmi bezpečný (data jsou šifrována a digitálně podepsána) a drží se filozofie HTTP. Hlavní nevýhodou je to, že zvyšuje objem přenášených dat. Někdy až mnohonásobně. [7]

- **MVC** – novější způsob pohlížející na celou problematiku jinak. Rozděluje celou logiku problému generování webového obsahu na tři části: [7]
  - **Kontroler** – řídící komponenta, která přijme od uživatele data a komunikuje s modelem. [7]
  - **Model** – část, která obsahuje veškerou logiku (program) [7]
  - **Pohled** – jedná se o šablonu, do které model vloží data. Tím se vytvoří výsledná HTML stránka a ta se následně odesílá uživateli. [7]

Pro mojí webovou aplikaci jsem si vybral WebForms, neboť mám zkušenosti s jejím využíváním.

5.1 Microsoft Visual Studio

Microsoft Visual Studio je ucelená kolekce nástrojů a služeb (vývojové prostředí, databázové nástroje, virtuální zařízení…), ve kterém je programátor schopen vytvářet aplikace pro různé technologie nejen pro platformu Windows (iOS, Android, Linux). [6]



Obrázek 11 - Microsoft Visual Studio

5.1.1 Licence

Microsoft Visual Studio (dále jen VS) je vydáváno v různých verzích a s různými přívlastky. Většina z těchto verzí je určena pro komerční oblast, kde si firmy mohou zaplatit licenci ať již v podobě paušálních poplatků, tak zakoupení konkrétní verze VS. Vždy se ovšem vydávají i verze, které nejsou zpoplatněné – *Community* a *Express*. [7]

- **Community –** srovnatelná s verzí *Professional*, avšak licenční podmínky nedovolují tuto verzi využít ve firemních sítích s více než 250 počítači, nebo pokud se jedná o firmu s vyšším ročním výnosem než 1 milion dolarů. [7]
- **Express –** jedná se v podstatě o jednotlivé podčásti VS *Professional* (proto přívlastky jako *VS 2013 Express for Web*), mají menší sadu nástrojů, nemají podporu vzdálené databáze, nemají x64 překladače. Míří především na amatéry, či studenty. [7]

Ačkoliv mám k dispozici, jako student, verzi *Ultimate* od *DreamSpark*, rozhodl jsem se využít verze *VS 2013 Express for Web*, pro splnění cílů diplomové práce plně dostačí. Navíc díky využití této verze nemusím řešit, zda v případě vytváření webové aplikace pro AL se školní licencí VS neporušuji licenci *DreamSpark*.

5.1.2 Instalace

Instalace VS je velice jednoduchá. Po zakoupení licence, či využití nezpoplatněné licence VS, je možné si ze stránek VS (www.visualstudio.com) stáhnout instalátor. Po zadání několika vstupních parametrů (umístění instalace, komponenty k instalaci) začne instalace VS. Jedná se ovšem o velmi velký objem dat, VS potřebuje pro běh mnoho komponent a proto instalace trvá v řádu hodin. Ačkoliv se s každou novou verzí VS zkracuje doba instalace, tak instalace poslední verze stále trvá okolo 2 hodin. [7]

5.1.3 Možnosti vývoje

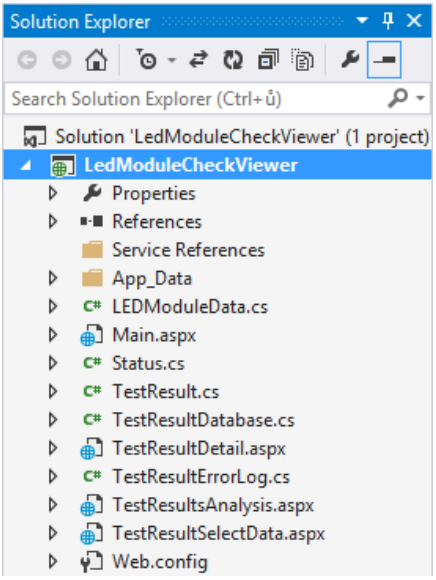
VS má ve znaku symbol nekonečna a jeho možnosti pouze potvrzují, proč se Microsoft rozhodl pro tento znak. Tento nástroj umožňuje tvorbu aplikací pro všechny Microsoftem vyvinuté technologie a platformy, které mají své pevné místo ve firemní sféře. Navíc v právě vydávané verze VS 2015 jsou uvolněny i pro platformy OS X (Apple) a Linux. Dále umožňuje tvorbu programů i pro mobilní platformy Windows, iOS a Android. Vyhlídky VS tedy vypadají nadějně a z VS se tak začíná stávat velmi univerzální a multiplatformní vývojový nástroj, který se díky velmi vstřícné licenční politice a široké podpoře akademického prostředí jistě udrží na trhu a svojí pozici bude nadále posilovat. [7]

6 WEBOVÁ APLIKACE

Mluvíme-li o vývoji aplikací ve VS, mluvíme v podstatě o projektech. Pod tímto názvem si můžeme představit souhrn informací o naší aplikaci (název, využité technologie, programovací jazyk, cílovou platformu), dále umístění, kde jsou zdrojové komponenty aplikace (složka projektu), přístup k databázi a další. [7]

6.1 Založení projektu

První věc, kterou jsem si musel určit před založením projektu, byl název. Jelikož veškeré syntaxe, komentáře a celkově i uživatelské rozhraní v aplikacích, které jsou v AL vyvíjeny, musí být dle interních směrnic AL v anglickém jazyce, tak již samotný název webové aplikace musí být taktéž v anglickém jazyce. Jako pracovní název jsem si tedy zvolil *LED Module Check Viewer*.
Po spuštění Microsoft Visual Studia jsem si založil nový projekt pomocí FILE -> NEW PROJECT. Poté jsem v okně s konfigurací nového projektu zvolil typ projektu *ASP.NET Web Application, Visual C#*, následně název projektu jsem vyplnil jako *LedModuleCheckViewer*, dále požadované umístění a Solution name jsem nechal stejné, jako název projektu.
V dalším kroku jsem si musel zvolit šablonu vytvářeného projektu. VS obsahuje již několik dopředu vytvořených šablon projektů, kde je již například vytvořen základní styl webové aplikace s grafickým vzhledem, systémem uživatelů či konkrétním typem způsobu vytváření webových stránek (WebForms, MVC). Vzhledem k povaze a specifičnosti webové aplikace jsem se rozhodl nevyužít žádné předpřipravené šablony. Tímto krokem jsem měl vytvořený nový projekt. Nyní jsem musel vytvořit databázi, ke které bude aplikace přistupovat.



Obrázek 12 - Solution Explorer

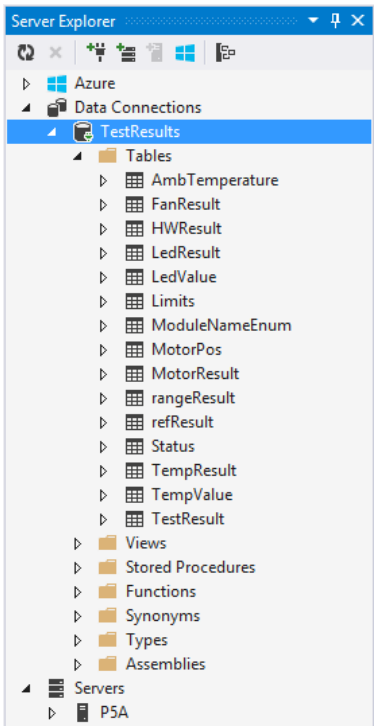
6.2 Vytvoření databáze

Dalším krokem bylo vytvoření databáze. Tu jsem vytvořil tak, že ve VS v projektu *LedModuleCheckViewer* jsem v podokně *Solution Explorer* klikl pravým tlačítkem myši na název projektu a dále v kontextové nabídce na ADD -> New Item. Poté jsem v okně s nabídkou nových objektů do projektu vybral skupinu Visual C# -> Data a v této skupině jsem vybral objekt *SQL Server Database*. Název zvoleného objektu (databáze) jsem zadal jako *TestResults*.

Po kliknutí na tlačítko ADD se v *Solution Explorer* objevila složka *App_Data* a v ní soubor *TestResults.mdf*, do kterého MSSQL zapisuje uložená data. S tímto souborem, jako takovým, nelze provádět žádné operace a nelze přímo přistupovat k datům, která obsahuje. Pro tento účel se využívá *Server Explorer*, kde se právě vytvořená databáze objevila v záložce *Data Connections*. Právě přes *Server Explorer* se ve VS přímo přistupuje k databázím. Zde je možné definovat parametry databáze, prováděl SQL příkazy, vytvářet pohledy, ukládat procedury atd.

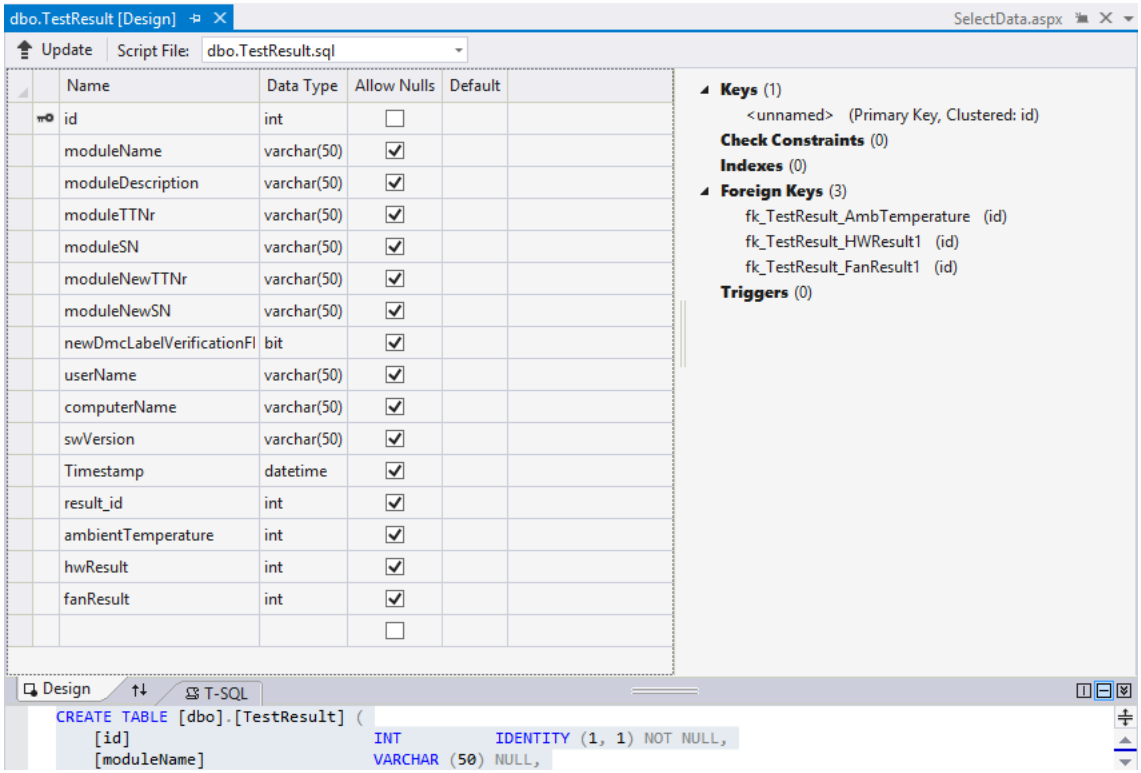
Pro provedení SQL, které jsem si nechal vygenerovat podle navrhnutého ER diagramu (viz 4.2 – Návrh ER diagramu), jsem tedy v *Server Exlorer -> Data Connections* klikl pravým tlačítkem myši na vytvořenou databázi a z kontextové nabídky vybral možnost *New Query*.

Do nové otevřeného okna jsem pak vložil vygenerované SQL příkazy na základě ER diagramu (viz 4.2 – Návrh ER diagramu). Po kliknutí na spuštění dotazu se úspěšně provedl SQL příkaz a struktura databáze v *Server Explorer* pak vypadala následovně:



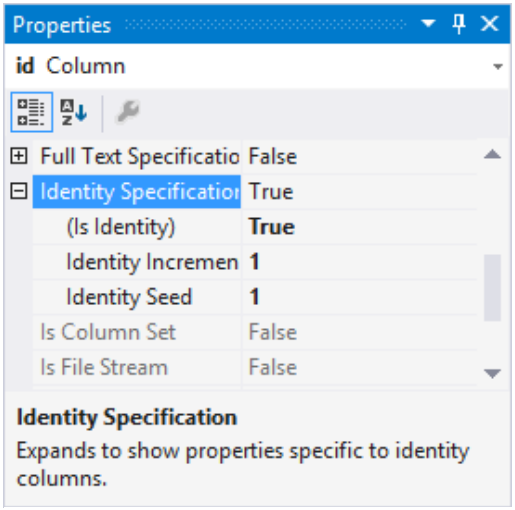
Obrázek 13 - VS, Server Explorer

Při dvojkliku na libovolnou tabulku v *Server Explorer* se otevře nové okno se strukturou a informacemi o dané tabulce. Například při dvojkliku na *TestResult* se otevře *Database Table Design* (viz Obrázek 14).



Obrázek 14 - VS, Database Table Design

Jak je vidět na Obrázku 14, tabulka *TestResult* má 16 datových položek, položka id je primární klíč, dále má tabulka 3 cizí klíče a krom id jsou u všech zbývajících položek povoleny *null* (prázdné) hodnoty. Vytvoření tabulky tedy proběhlo v pořádku a přesně tak, jak jsem definoval při vytváření ER diagramu.
Stejným postupem jsem ověřil všechny tabulky v databázi, jejich vytvoření taktéž proběhlo v pořádku. Dále jsem u všech id tabulek nastavil autoinkrementaci id, čímž se zajistí, že databáze si hodnotu id nově vzniklého záznamu určí sama a nedojde tak ke kolizi hodnot id.



Obrázek 15 - VS, nastavení auto inkrementace id

Poslední věcí, kterou jsem musel udělat, bylo zjištění tzv. *Connection String* jedná se o připojovací řetězec k databázi. Obsahuje název databáze, verzi databáze, umístění databáze, přihlašování atd. Pro jeho zjištění stačí v *Server Explorer* stisknout pravé tlačítko myši nad databází a z kontextové nabídky vybrat *Properties*. Zde je hned první položkou *Connection String*, v mém případě má hodnotu:

```
Data Source =
(LocalDB)\v11.0;AttachDbFilename=c:\David\Diplomka\Projekt\LedModuleCheckViewer\
LedModuleCheckViewer\App_Data\TestResults.mdf;Integrated Security=True
```

Jak je patrné, cesta k databázi je dána absolutní adresou. Proto jsem cestu upravil tak, aby cesta byla relativní a odkazovala se vůči složce projektu. Poté *Connection String* vypadal následovně:
V mém případě databáze neeviduje žádné přihlašovací údaje, pokud by tomu tak bylo,

```
Data Source =
(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|TestResults.mdf;Integrated
Security=True
```

bylo by nutné *Connection String* doplnit o tyto údaje.
Tímto krokem jsem měl vytvořenou databázi a mohl jsem začít vytvářet funkční část webové aplikace.

6.3 Vykreslování grafů

K vykreslování všech typů grafů použitých ve webové aplikaci jsem využil objektů třídy *Chart* definované v *System.Web.UI.DataVisualization.Charting*. Jedná se o standardní ovládací prvek obsažený v *ASP.NET* umožňující generovat grafy různých typů s širokou podporou grafických úprav, aby výsledný graf co možná nejlépe odpovídal požadované podobě.
Po vytvoření instance objektu třídy *Chart* je nutné do tohoto objektu přidat *ChartArea*, jedná se v podstatě o oblast grafu sloužící k vykreslování. Zde je možné nastavit osy X a Y, jejich kótování, rozsah, dále styl mřížky grafu atd. Poté je nutné vytvořit *Series*, jedná se o třídu, která v sobě uchovává samotné hodnoty, které se vykreslují (X a Y hodnota). Dále je možné opět nastavit typ vykreslování (čára, sloupce, výseč…) styl atd.
Poslední částí, kterou jsem využíval, je vytvoření legendy grafu k popisku pod grafem.
Pro ukázku, jak jsem využíval této třídy, uvádím ukázku s komentáři z generování grafu pro pozice motorů LED modulu:

```
// VYTVORENI INSTANCE OBJEKTU GRAF, NASTAVENI NAZVU A
// PRIDANI OBLASTI PRO VYKRESLOVANI HODNOT
Chart Graf = new Chart();
Graf.Titles.Add(element.name + " - Position");
Graf.ChartAreas.Add(new ChartArea("Position" + index.ToString()));

// DEKLARACE NOVYCH "SERII" SLOUZICI PRO UKLADANI HODNOT
Series serieValue = Graf.Series.Add("Position");
Series serieMaxValue = Graf.Series.Add("Position MAX value");
Series serieMinValue = Graf.Series.Add("Position MIN value");

// NASTAVENI BAREV A TYPU CAR JEDNOTLIVYCH SERII DAT
serieValue.ChartType = SeriesChartType.StepLine;
serieValue.Color = System.Drawing.Color.Blue;
serieMaxValue.ChartType = SeriesChartType.Line;
serieMaxValue.Color = System.Drawing.Color.Red;
serieMinValue.ChartType = SeriesChartType.Line;
serieMinValue.Color = System.Drawing.Color.Red;

// NASTAVENI TLOUSTKY CAR
Graf.Series["Position"].BorderWidth = 3;
Graf.Series["Position MAX value"].BorderWidth = 2;
Graf.Series["Position MIN value"].BorderWidth = 2;

// NASTAVENI LEGENDY GRAFU, VZHLED OSY X A Y, MRIZKY GRAFU
Graf.Legends.Add(new Legend("Legend"));
Graf.Legends["Legend"].Alignment = System.Drawing.StringAlignment.Near;
Graf.Legends["Legend"].Docking = Docking.Bottom;
Graf.Series["Position"].Legend = "Legend";
Graf.Series["Position"].IsVisibleInLegend = true;
Graf.Series["Position"].IsValueShownAsLabel = true;
Graf.ChartAreas.Last().AxisX.Minimum = 0;
Graf.ChartAreas.Last().AxisX.Title = "Time (seconds)";
Graf.ChartAreas.Last().AxisX.LabelStyle.Format = "F1";
Graf.ChartAreas.Last().AxisX.MajorGrid.Enabled = false;
Graf.ChartAreas.Last().AxisX.IsStartedFromZero = true;
Graf.ChartAreas.Last().AxisY.Title = "Position (coordinates)";
Graf.ChartAreas.Last().AxisY.MajorGrid.LineDashStyle = ChartDashStyle.Dot;

// PRIPRAVENI VSTUPNICH DAT (SERAZENI A URCENI PRVNIHO CASOVEHO ZAZNAMU
// JAKO REFERENCNI HODNOTY
element.positions.OrderBy(Date => Date.timeStamp);
FirstTime = element.positions.ElementAt(0).timeStamp;

// ZAPIS DAT DO SERIE, HODNOTA X SE URCUJE PODLE ROZDILU AKTUALNI HODNOTY CASU
// A REFERENCNI HODNOTY CASU. HODNOTA Y SE VKLADA PRIMO
foreach (LED_Module_Check_GUI.Source.MotorPos element_value in
element.positions)
{
    DifferenceTime = element_value.timeStamp - FirstTime;
    serieValue.Points.AddXY(DifferenceTime.TotalSeconds, element_value.value);
    serieMaxValue.Points.AddXY(DifferenceTime.TotalSeconds, element.endPos1);
    serieMinValue.Points.AddXY(DifferenceTime.TotalSeconds, element.endPos2);
}
```

Tímto kódem je generován Graf 10 v kapitole 6.5.4 - TestResultDetail

6.4 Tisk do PDF

Bohužel *ASP.NET* standardně neobsahuje žádný nástroj, který by byl schopen tisknout do PDF. Vzhledem k této absenci a neuvěřitelné složitosti takového nástroje, rozhodl jsem se nevytvářet tento nástroj, ale použít již hotových nástrojů třetích stran.
Během hledání jsem zjistil, že nástroje pro tisk PDF do *ASP.NET* přistupují k ovládání a vytváření PDF dvojím způsobem:

- Psaní dokumentu – tento způsob pracuje na principu jakéhosi přímého psaní do PDF. Ve většině případů se ovládaly tyto nástroje tak, že se pomocí nich vytvořil nový dokument, a pak se pomocí souřadnic do dokumentu umisťoval kurzor a následně se psal text, popřípadě vytvářely tabulky nebo vkládal obrázek. Posléze se dokument uložil jako PDF.
- Konverze HTML kódu – tento způsob pracuje na principu konvertoru, kterému se zadá odkaz na HTML webovou stránku, nastaví se další parametry (název PDF, šířka okrajů, stupeň komprese obrázků atd.) a následně se provede konverze. Do PDF se tak v podstatě uloží odkazovaná webová stránka.

Pro mojí webovou aplikaci se přímo vybízel druhý způsob, díky kterému jsem nemusel vytvářet další strukturu zobrazení dat, pouze jsem funkce starající se o generování dynamických tabulek doplnil o mechanismy, které si zjistí, zda se stránka tiskne do PDF a pokud ano, pak nastaví například větší font u popisků grafů, nastaví, že se zobrazují pouze dva grafy vedle sebe nebo že se skryjí tlačítka (která by v PDF neměla žádnou funkci).
Po dlouhém hledání a zkoušení všech nalezených nástrojů jsem si zvolil *SelectPDF* (dostupný z http://selectpdf.com/), který nabízí oba dva způsoby vytváření PDF, ovládání ze všech zkoušených nástrojů mi přišlo nejjednodušší a logické, navíc ve své free verzi umožňuje použití i v komerční sféře pouze s omezením, že maximální počet stránek v PDF je 5. V tomto případě se neporuší licenční podmínky a k počtu stránek – webová aplikace díky tomuto nástroji generuje na první stránku PDF celkem 4 grafy a následně na každou stránku 6 grafů. Maximální počet grafů v PDF je tedy 28 a po konzultaci s pracovníky AL jsem byl ujištěn, že nikdy nebude potřeba z jednoho měření vytvářet takový počet grafů.
Výsledné vygenerované PDF přikládám v příloze č. 3 a v příloze č. 4.

6.5 Třídy aplikace

Dalším krokem bylo vytvoření funkčních tříd, do kterých jsem umisťoval jednotlivé vytvořené funkce dle logického rozdělení (viz Obrázek 12).

6.5.1 TestResultErrorLog

První třída, kterou jsem vytvořil. Třída nemá webový obsah, má pouze jedinou funkci:

```
static public void ErrorLog(string Message)
```

Tuto funkci volám při vyvolání výjimky během běhu programu. Pro lepší představu uvedu příklad volání výjimky:

```
try
{
        //PRIKAZ1
}

catch (Exception err)
{
        TestResultErrorLog.ErrorLog(err.Message);
}

finally
{
        // PRIKAZ2
}
```

Pro zachycení výjimky se používá výše uvedená konstrukce, která se dělí na tři části:

- **TRY –** do této části se umisťuje ta část programu (většinou volání funkcí), u které není zaručeno, že vždy proběhne. Například se může jednat o funkci zajišťující připojení k databázi. Pokud například není dostupné připojení k serveru s databází, tak se funkce neprovede a vyvolá výjimku.
- **CATCH –** tato část slouží právě k zachycení případné výjimky. Slouží k určení, jak se má program zachovat v případě vyvolání výjimky. Obvykle se jedná o vyskakovací okno s informací o výjimce (chybě) programu. V uvedené příkladu se provede zavolání práce funkce ErrorLog, která je ve třídě TestResultErrorLog. Tato funkce se zavolá s parametrem err.Message, což je chybová hláška.
- **FINALLY –** tato část se provede vždy. Ať už část TRY proběhne bez problému, nebo se nedokončí a vyvolá výjimku. Do FINALLY se většinou umisťují funkce uzavírající připojení k databázi, nebo zavírající soubor, atd.

Důvod volání funkce *ErrorLog* je tedy jasný. Funkce samotná slouží k tomu, že vezme chybovou hlášku z parametru a uloží ji do souboru ErrorLog_#ČAS#.txt a ten uloží do složky *ErrorLog* v adresáři programu. Takto se podchytí jakákoliv výjimka vzniklá za běhu aplikace a je možné podle ní později upravovat aplikaci.

6.5.2 TestResultDatabase

Tuto třídu jsem vytvořil jako styčný bod s databází, třída nemá webový obsah, jedná se o funkční třídu. Sice jsem veškeré SQL dotazy na databázi mohl uložit do *Stored Procedures* přímo v MSSQL a pak je volat pouze jako funkce, ale vzhledem k tomu, že s databází v podstatě aplikace komunikuje jen prostřednictvím této třídy a nejedná se o jednoduché příkazy (navíc většinu volá cyklicky), tak jsem se rozhodl, že s databází budu komunikovat prostřednictvím této třídy a možnosti *Stored Procedures* nevyužiji.
V této třídě se nacházejí následující funkce:

```
static public void ReadFolder(string folder_path)
```

Tato funkce slouží k hledání XML souborů v zadané složce (pomocí parametru *folder_path*), následné provedení deserializace XML pomocí funkce *ReadXML*, kterou volá s odkazem na nalezený XML soubor a s referencí na instanci objektu *new_testresult* třídy *TestResult*. Poté funkce zavolá funkci *SaveTestResultToDatabase* s parametrem reference objektu *new_testresult*, který v okamžik volání této funkce již obsahuje načtené hodnoty z nalezeného XML souboru. Nakonec tato funkce přesune úspěšně načtený soubor do složky *archive* k pozdější kompresi veškerých použitých XML souborů a archivaci. Obě použité funkce funkcí *ReadFolder* popíši později.

```
static public void ReadXML(string path, ref TestResult new_testresult)
```

Tato funkce provádí deserializaci XML souboru (cesta k němu je parametr *path*) a výsledek deserializace ukládá do instance objektu *new_testresult*, na kterou má funkce referenci v parametru.

```
static public int SearchForStatus(
System.Data.SqlClient.SqlConnection con,
string cmdString,
System.Data.SqlClient.SqlCommand cmd,
string ErrorText,
Int64 ModuleTextConditionPass,
Int64 enValue,
int ErrorCode_id,
int LastErrorCode_id)
```

Vzhledem k tomu, že v tabulce *STATUS* je největší riziko vzniku redundantních dat, vytvořil jsem funkci *SearchForStatus*. Tato funkce v parametrech má aktuální připojení na databázi (aby se nemuselo vytvářet nové a ukončovat staré) a hodnoty nového zápisu do tabulky *STATUS*. První co funkce udělá je, že provede SQL dotaz na databázi s požadavkem na vrácení všech dat z tabulky *STATUS* se všemi parametry, které má nově vytvářený *STATUS*. V případě, že databáze nevrátí žádná data, tak funkce vrátí hodnotu *int.MinValue* (což je -2 147 483 648). Pokud ale databáze navrátí data z tabulky *STATUS* s požadovanými daty, tak funkce vrátí hodnotu identifikačního klíče (id).
Práce s touto funkcí je velmi jednoduchá. Před zápisem nového záznamu do tabulky *STATUS* se zavolá tato funkce. Pokud funkce vrátí hodnotu *int.MinValue*, pak žádný záznam v tabulce *STATUS* s těmito parametry, jako má připravovaný nový záznam, není

a je možné tedy vytvořit nový záznam bez rizika redundance záznamu. Pokud ovšem funkce vrátí jinou hodnotu (obecně lze určit hodnotu větší nebo rovnu nule), pak v databázi v tabulce *STATUS* existuje odpovídající záznam a jeho id je hodnota navrácena funkcí. Pak se tedy vezme tato hodnota a uloží se jako cizí klíč do právě řešené tabulky a stále nedojde k redundanci dat.
Za zmínku určitě stojí, že pokud tuto funkci vypnu, tak při nahrávání 74 souborů XML do databáze z různých měření vznikne 51208 záznamů v tabulce *STATUS*. Po smazání všech dat z databáze, zapnutí funkce *SearchForStatus* a provedení stejné operace s načítání dat ze 74 souborů, tak vznikne pouze 50 záznamů v tabulce *STATUS*. Důležitost této funkce při zamezení výskytu redundantních dat je tak jasně prokázána.

```
static public int SearchForModuleNameEnum(
System.Data.SqlClient.SqlConnection con,
string cmdString,
System.Data.SqlClient.SqlCommand cmd,
string moduleName)
```

Funkce *SearchForModuleNameEnum* má v podstatě jedinou funkci – připravovat data pro již zmiňovanou „roletku“ (viz 4.2 – Návrh ER diagramu). Jak jsem již zmiňoval, tato roletka má sloužit k rychlejšímu vyhledávání v datech. Uživatel pouze rozklikne „roletku“ a vybere si ze seznamu názvů modulů. Nemusí nic zadávat, nevzniká ani možnost chyby při zadávání. Pro hodnoty této tabulky jsem vytvořil již zmíněnou tabulku *ModuleNameEnum* a tato funkce nahrává data do této tabulky.
Po zavolání se funkce pomocí SQL dotazu dotáže databáze, zdali tabulka *ModuleNameEnum* neobsahuje záznam, kde položka *moduleName* se nerovná parametru funkce *moduleName*. Pokud tabulka tento záznam neobsahuje, pak funkce vrátí hodnotu *int.MinValue*, pokud tabulka tento záznam obsahuje, pak funkce vrátí hodnotu identifikačního klíče záznamu.
Funkce se využívá podobně jako funkce *SearchForStatus*. Pokud funkce po zavolání vrátí hodnotu *int.MinValue*, pak se provede zápis záznamu do tabulky *ModuleNameEnum*. Pokud vrátí jinou hodnotu (opět větší nebo rovnu nule), pak tabulka již záznam obsahuje a není potřeba vytvářet nový záznam.

```
static public void SaveTestResultToDatabase(ref TestResult actual_testresult)
```

Jedná se o jednu z nejdůležitějších funkcí. Tato funkce provádí zápis instance objektu *actual_testresult* třídy *TestResult* do databáze. Popisovat postupně veškeré části funkce by bylo velmi zdlouhavé, proto uvedu spíše principiální popis funkce a uvedu zajímavé problémy, na které jsem během programování narazil.
Princip funkce spočívá v tom, že funkce postupně prochází veškeré instance objektů, které obsahuje základní instance objektu třídy *TestResult*, a provádí jejích zápis do databáze a kompletně řeší problém zápisu cizích klíčů v databázi, aby bylo možné později nalézt veškeré části tohoto komplexního záznamu. Pokud bych to měl popsat jinak – jedná se o to, že tato funkce vezme instanci objektu třídy TestResult, rozdělí ho do odpovídajících tabulek a při postupném zápisu těchto tabulek do databáze i zajišťuje, aby veškeré cizí klíče byly správně zapsány.

Při psaní této funkce jsem narazil na celkem velmi zajímavé problémy:

- MSSQL nezná datový typ *double*. Pro uchování hodnot datového typu *double* jsem vybral datový typ *float*, protože je přesnější a má větší rozsah, nebude docházet ke ztrátám dat.
- Při vkládání položek datového typu *double* do databáze docházelo k neplatným zápisům a vracely se výjimky funkce. Po návštěvě odborných fór byl problém jasný – v české lokalizaci systému má *double* podobu například „12,432“. Funkce tuto hodnotu přesně tak, jak je uvedena, převedla na text (*string*) a snažila se jí vložit do databáze. Bohužel MSSQL tuto notaci nedovoluje, navíc už samotný SQL dotaz si s touto notací neporadil, neboť čárku bral jako oddělovací znak. Náprava tohoto problému je velmi jednoduchá – při volání funkce ToString() nad touto položkou stačí využít přetížené funkce ToString() s parametrem *CultureInfo.InvariantCulture*, který zajistí to, že jako desetinný oddělovač bude místo čárky použita tečka, pak se tato hodnota převede na text jako „12.432“ a tak již nevzniká uvedený problém.
- Při vkládání položek datového typu *DateTime* vznikal velmi podobný problém, jako při vkládání položek datového typu double. Zde ovšem nebyl problém jen v notaci, ale i v celkové skladbě časového otisku. Konkrétně se jednalo o datum, kde v české mutaci se využívá formátu „den.měsíc.rok“, zatímco databáze požaduje formát „měsíc/den/rok“. Při použití stejného řešení, jako v případě double sice problém zanikl, ale objevil se ihned další – do databáze, jako nejmenší časová jednotka, se zapisovaly vteřiny. V normálních aplikacích by to problém nebyl, avšak zde probíhá zápis hodnot, které jsou měřeny periodicky po 200ms, čímž již vzniká problém, neboť by tak v databázi bylo najednou 5 hodnot se stejným časovým otiskem, ačkoliv jsou různé. Nehledě na zkreslení dat. Proto jsem volanou funkci *ToString()* musel využít s parametrem „MM/dd/yyyy HH:mm:ss.fff“, který mi zajistil, že se do databáze zapsal formát datového otisku tak, jak databáze požaduje, navíc ovšem i s milisekundami, které jsou pro zobrazování naměřených dat velice důležité.
- Další problém se také týká časového otisku (položka datového typu *DateTime*). Tato chyba vznikla tak, že SW Led Module Check při inicializacích objektů, které obsahují časový otisk, nastaví hodnotu tohoto časového otisku na *DateTime.MinValue*, což je 1.1.0001. Určité objekty v *TestResult* se ovšem inicializují, aniž by se pak naplnily, takže hodnota časového otisku zůstane na 1.1.0001. V tom by problém ani až tak nebyl, avšak MSSQL jako minimální hodnotu *DateTime* má určeno datum 1.1.1900, tím pádem při pokusu o zápis hodnoty 1.1.0001 vznikala výjimka a zápis se neprovedl. Tento problém jsem vyřešil tak, že před každým zápisem časového otisku program provede kontrolu, zda datum časového otisku není menší, než 1.1.1900. Pokud je menší, pak program přenastaví datum na 1.1.1900. Pokud není menší, pak problém nevzniká a program zapíše požadovanou hodnotu.

```
static public void LoadTestResultFromDatabaseStatus(
int id,
ref STATUS element,
System.Data.SqlClient.SqlConnection con,
string cmdString,
System.Data.SqlClient.SqlCommand cmd)
```

Tato funkce slouží k načtení dat z tabulky *STATUS* do reference na instanci objektu *element* třídy *STATUS*. Jelikož jsou v pozdější funkci *LoadTestResultFromDatabase* mnohokrát načítána data z tabulky *STATUS*, vytvořil jsem tuto funkci, kterou stačí zavolat s parametry připojení, požadovaným záznamem s id a referenci na instanci objektu, kam se mají načtená data uložit. Pokud funkce požadovaný záznam z tabulky *STATUS* nenalezne, uloží do objektu defaultní hodnoty.

```
static public TestResult LoadTestResultFromDatabase(
int TestResult_ID,
bool LoadAll)
```

Jedná se o funkci, která v podstatě dělá opak funkce *SaveTestResultToDatabase*. Tato funkce načítá *TestResult* z databáze a vrací jej programu. Podle jednotlivých cizích klíčů tato funkce poskládá objekt třídy *TestResult* podle požadovaného id (*TestResult_ID*) v parametru funkce. Dalším parametrem je *LoadAll*. Pokud tento parametr má hodnotu *true*, pak funkce načte veškerá data. Pokud má parametr hodnotu *false*, pak tato funkce načtě pouze „hlavičku“ *TestResultu*. To se hodí především při výpisu všech záznamů měření v databázi odpovídajících parametrům hledání. Protože plný záznam měření obsahuje řádově tísíce záznamů (instancí objektů), trvalo by načítání seznamu při načítání úplných záznamů neúměrně dlouho. Proto se v této situaci využívá pouze „hlaviček“, které jsou mnohem menší a nejsou tak náročné.

```
static public string returnFloatString(double input)
```

Tato funkce řeší problém při přetypování datového typu *double* na *string*, který následně databáze přetypovává na *float*. V SW *LED Module Check* je totiž jako defaultní hodnota položek datového typu *double* nastavena jako *double.MaxValue* nebo *double.MinValue*. Při přetypování na *string* standardní funkcí *ToString()* vznikají problémy. Proto tato funkce nejdříve určí, zda parametr funkce input má hodnotu *double.MaxValue* nebo *double.MinValue*. Podle výsledku funkce vrátí hodnotu *float.MinValue* nebo *floatMaxValue* přetypovanou na *string*.

```
static public void LoadModuleNames(
ref System.Web.UI.WebControls.DropDownList DropDownListModuleNames)
```

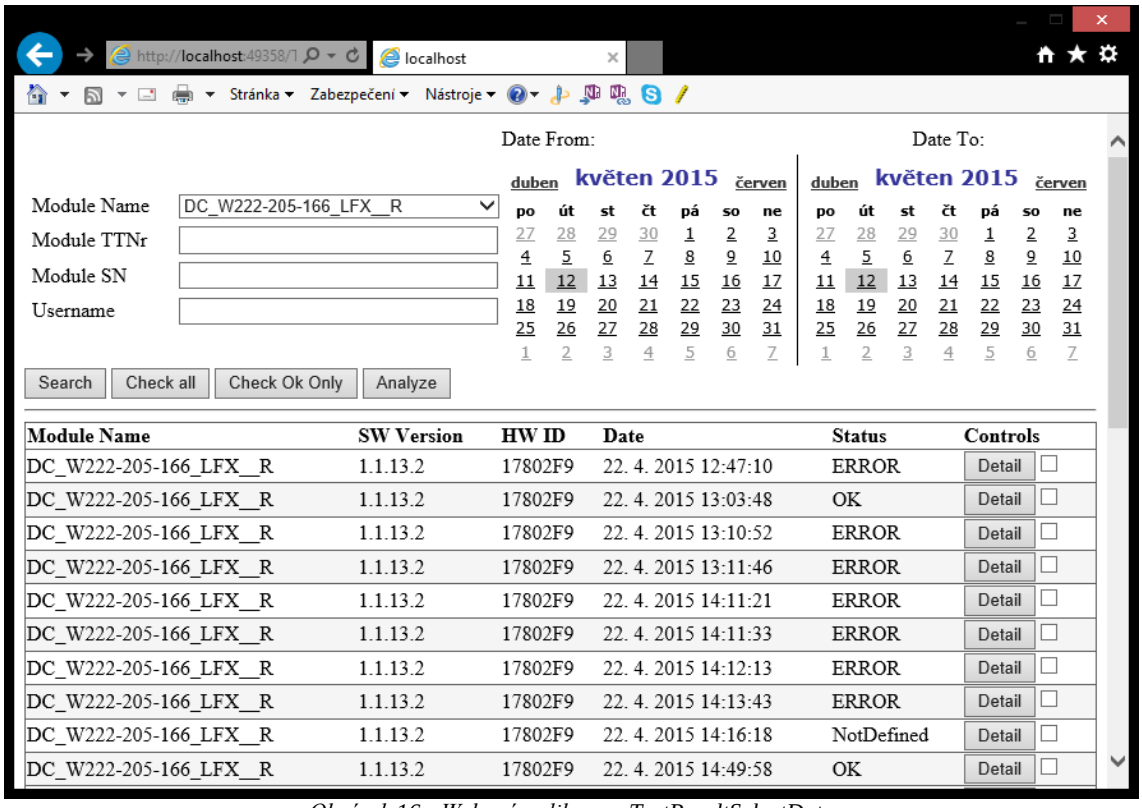
Funkce, která z databáze z tabulky *ModuleNameEnum* vyčte veškeré hodnoty a uloží je do referenčního objektu třídy *DropDownList* v parametru funkce.

```
static public void SearchForTestResultsHeaders(
ref List<LEDModuleCheckViewer.SelectData.TestResultItem> TestResultItemsList,
string DropDownListModuleName,
string TextBoxModuleTTNr,
string TextBoxModuleSN,
string TextBoxUserName,
DateTime CalendarFrom,
DateTime CalendarTo)
```

Jedná se o funkci, která na základě parametrů vyhledá a přidá do referenčního listu objektů tříd *TestResultItem* nalezené záznamy. Tato funkce volá funkci *LoadTestResultFromDatabase* s parametrem *LoadAll = false*, čili nedochází k načítání celého objektu třídy *TestResult*, ale pouze jeho hlavičky.

6.5.3 TestResultSelectData

Jedná se o první třídu, která má webový obsah. Tato funkce slouží k vyhledání a výběru dat k zobrazení dle různých parametrů.



Obrázek 16 - Webová aplikace – TestResultSelectData

Na rozdíl od přechozích tříd, tato obsahuje již i webový obsah. Nejprve jsem začal právě návrhem tohoto webového rozhraní. První snahou bylo, aby byl vyhledávací nástroj graficky jednoduchý a přehledný. Navíc předpokládám, že než bude tato webová aplikace

nasazena do ostrého provozu, projde její grafická část velkou grafickou úpravou, aby odpovídala vnitřním normám AL.
Webový obsah se dá rozdělit do tří částí:

- **Parametrická část** – slouží k nastavení parametrů pro vyhledávání v uložených záznamech. Tato část je v horní části webového obsahu a spadá do ní roletka *Module Name*, textboxy *Module TTNr, Module SN, Username* a dva kalendáře určující, jaké časové období měření uživatele zajímá. Pro roletku *Module Name* se při načítání webového obsahu provede dotaz na databázi a ta vrátí veškerá jména modulů obsažených v databázi. Samozřejmě bez duplicitních záznamů. Zbylé textboxy slouží k danému účelu – filtrování výsledků podle TTNr, SN či uživatele, který provedl testy.
- **Ovládací část –** v této části jsou umístěna tlačítka pro vykonání požadovaných funkci. Názvy tlačítek odpovídají jejich funkci. Pro upřesnění uvedu stručný popis:
  - **Search –** provede vyhledání v databázi podle nastavených parametrů.
  - **Check All –** označí všechny nalezené výsledky pro pozdější analýzu nad daty. Po opětovném stisknutí naopak odznačí všechny výsledky.
  - **Check Ok Only –** označí pouze ty nalezené výsledky, kde celkový výsledek testu je Ok, jinými slovy – LED Modul prošel testem bez problémů.
  - **Analyze –** toto tlačítko odešle všechny nalezené a označené výsledky k analýze.
- **Tabulka s výsledky –** tabulka, do které se zapisují nalezené výsledky. Tabulka dále kromě jednotlivého označování výsledků umožňuje i zobrazit detail *TestResultu* (měření).

Tímto by byl popsán webový obsah třídy, nyní popis funkční části:
Nejprve je nutné zmínit, že třída *TestResult* nemá datovou položku uchovávající identifikační klíč (*Id*). Tudíž jako první věc, kterou jsem musel udělat, bylo vytvoření nové třídy, kterou budu využívat pro ukládání nalezených instancí objektů třídy *TestResult* a jejich id. Tuto třídu jsem mohl vytvořit dvěma způsoby – buď vytvořit potomka třídy (dědění) *TestResult*, který by tuto třídu rozšiřoval právě o datovou položku pokrývající id, nebo pomocí vnoření třídy *TestResult* do právě vytvořené třídy. Já jsem si zvolil druhou možnost, konkrétní realizace vypadá následovně:

```
public class TestResultItem
{
	public int id;
	public TestResult item;

	public TestResultItem(int i_id, TestResult i_item)
	{
		id = i_id; item = i_item;
	}}
```

Díky této třídě je program schopen evidovat jak třídu *TestResult*, tak i *id* odpovídající záznamu v databázi. Dále třída *TestResultSelectData* využívá následující funkce:

```
protected void Page_Load(object sender, EventArgs e)
```

Toto je funkce která se automaticky volá při načtení funkce. Jedná se o jakýsi „spouštěč“ automatických činností. Do této části jsem umístil volání funkce pro načtení názvů modulů v databázi (*LoadModuleNames*), následně volání funkce pro načtení záznamů odpovídajících parametrů hledání (*SearchForTestResultsHeaders*) a nakonec funkci, která pomocí načtených dat vyplní dynamickou tabulku pro výpis výsledků (*LoadTableTestResultItems*).

```
public void LoadTableTestResultItems()
```

Zmíněná funkce pro vyplnění dynamické tabulky. Jelikož nikdy nelze dopředu určit, kolik záznamů odpovídajících hledání se v databázi nalezne, využívá třída *TestResultSelectData* dynamické tabulky. Tato funkce nejprve vytvoří hlavičku dynamické tabulky a následně začne nahrávat data včetně ovládacích prvků (*checkbox* pro označení daného záznamu k analýze a tlačítka *Detail* pro zobrazení detailu daného záznamu).

```
protected void ButtonSearch_Click(object sender, EventArgs e)
```

Tato funkce slouží pouze k obnovení celé stránky, nevolají se v ní žádné další funkce, ani se nemění proměnné. Funkce je volána po kliknutí na tlačítko *Search*.

```
protected void ButtonDetail_Click(object sender, EventArgs e)
```

Funkce je volána všemi tlačítky *Detail* u jednotlivých záznamů. Každé tlačítko má své id, které v mém případě odpovídá id daného záznamu *TestResult*. Po kliknutí na takové tlačítko tato funkce otevře nové okno v prohlížeči s odkazem na *TestResultDetail.aspx* a přidá k odkazu parametr id s hodnotou id tlačítka, což je hodnota *id* daného *TestResult*.

```
protected void ButtonCheckAll_Click(object sender, EventArgs e)
```

Tato funkce se volá po stisknutí tlačítka *Check All* a veškeré checkboxy na stránce označí jako *true* (zaškrtnuto). Pokud je tlačítko voláno opětovně, udělá pravý opak – všechny checkboxy označí jako *false*. Takto neustále periodicky mění označení.

```
protected void ButtonAnalyze_Click(object sender, EventArgs e)
```

Tuto funkci volá tlačítko *Analyze*. Po zavolání tato funkce zkontroluje veškeré checkboxy a u těch, které mají hodnotu *true* (jsou zaškrtlé) si poznamená jejich *id*. Následně otevře nové okno prohlížeče s odkazem na *TestResultsAnalysis.aspx* a parametrem IDs, který je složen ze všech *id TestResultů*, u kterých byl zaškrtnutý checkbox.

```
protected void CalendarFrom_SelectionChanged(object sender, EventArgs e)
```

Při změně vybraného data kalendáře *Calendar From* se zavolá tato funkce, která pouze na webové stránce vypíše, jaké datum bylo zvoleno.

```
protected void CalendarTo_SelectionChanged(object sender, EventArgs e)
```

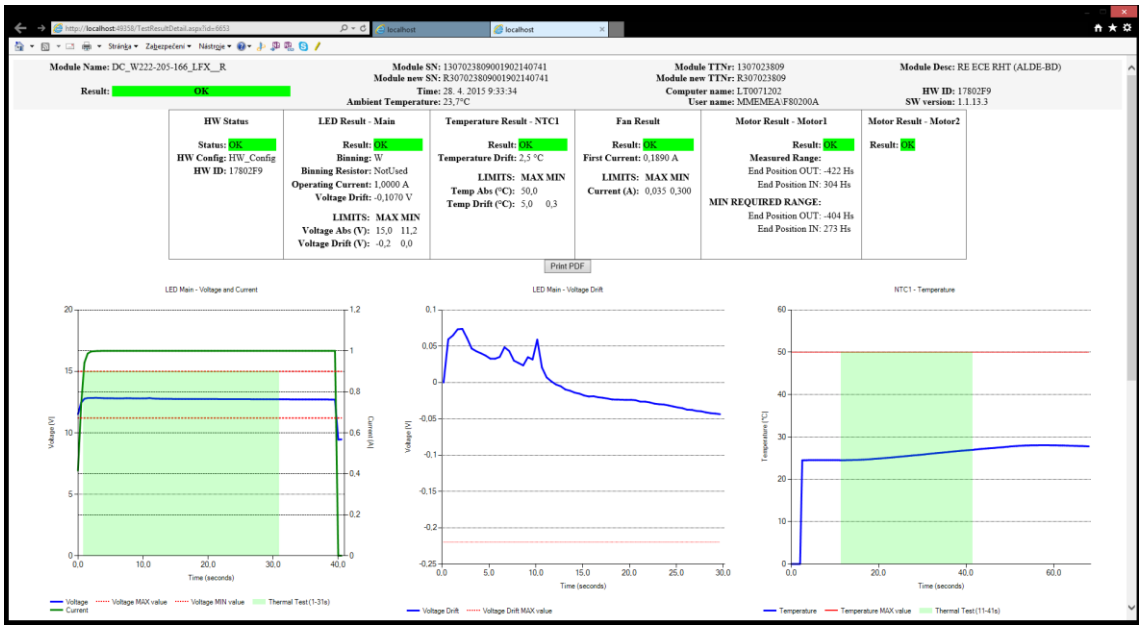
Naprosto identická funkce, jako předchozí, jen platí pro změnu data u kalendáře *Calendar To*.

```
protected void ButtonOkOnly_Click(object sender, EventArgs e)
```

Tato funkce je velmi podobná funkci *ButtonCheckAll_Click*. Také kontroluje všechny checkboxy na stránce, ale nastavuje hodnotu *true* (zaškrtnuto) pouze u těch, které jsou u záznamu *TestResult*, jehož celkový výsledek dopadl *Ok* (v pořádku). Toto se hodí v případě, kdy je nutné dělat analýzu pouze nad bezproblémovými měřeními.

6.5.4 TestResultDetail

Třída s webovým obsahem sloužící k náhledu na celkový průběh testu.



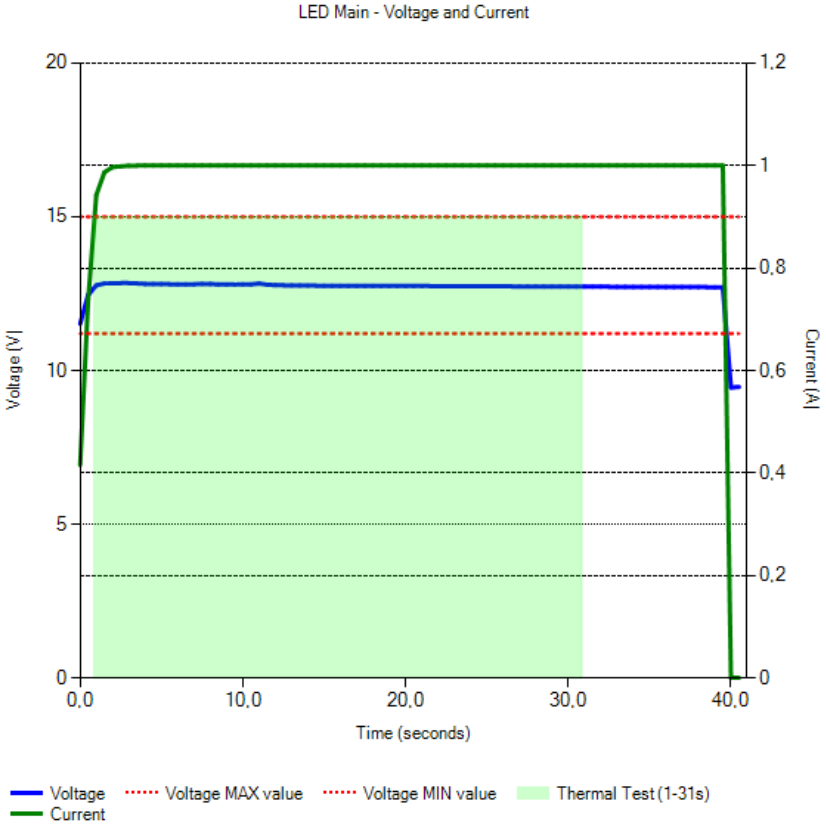
Obrázek 17 - Webová aplikace – TestResultDetail

Webový obsah lze rozdělit do čtyř skupin:

- **Hlavička testu** – nachází se v horní části webové stránky, jsou v ní zahrnuty naprosto základní informace o proběhlém testu (čas, název modulu, okolní teplota atd.) a celkový výsledek testu. Slouží k jednoznačnému určení, o jaký test se jedná a o jaký LED modul.

- **Výsledky jednotlivých měření** – nachází se pod hlavičkou testu a obsahuje výsledky a informace z jednotlivých měření (*STATUS* měřícího nástroje, LED hodnoty (napětí, proud), teplotní test, test motoru a větráčku).
- **Ovládací panel** – v současnosti se zde nachází pouze tlačítko generující PDF report (více později). Do budoucna počítám s dalšími funkcemi.
- **Grafy** – v této oblasti se vykreslují grafy jednotlivých měření.

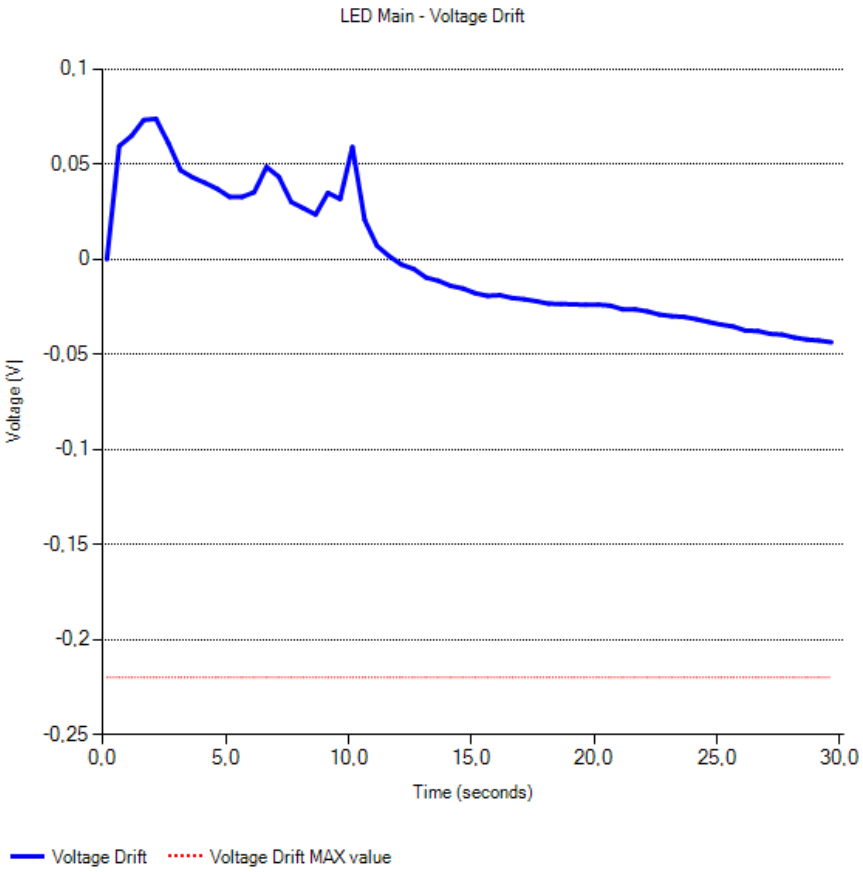
Při návrhu webového obsahu jsem se snažil o co největší přehlednost. Proto jsem informace umisťoval do jednoduchých tabulek a výsledky jsem navíc značil barevně, aby uživatel již na první pohled, aniž by studoval výsledky, věděl, jak test dopadl. Nejzajímavější část webového obsahu jsou jednoznačně grafy, rád bych se jim chvíli věnoval a ukázal detailněji grafy pro jednotlivá měření.
Tato třída generuje celkem 5 typů grafů. První typ grafů (viz Graf 6) slouží ke znázornění průběhu absolutní hodnoty napětí a proudu na dané skupině LED v čase. Grafy tohoto typu tedy mají dvě osy Y (napětí, proud) a jednu osu X (čas). Jako počáteční hodnota času (0s) se uvažuje čas první naměřené hodnoty.



Graf 6 - Webová aplikace - Graf absolutní hodnoty napětí a proudu

Za povšimnutí určitě stojí světle zelená oblast grafu. Ta graficky znázorňuje časový úsek, kdy byl vykonáván termální test (viz 2.3 – Měřené veličiny). V našem případě si můžeme povšimnout, že v čase cca 11s je u napětí mírný nárůst a následný pokles napětí. Pokud bychom se plně nesoustředili na tento graf, ani bychom si tohoto výkyvu nevšimli.

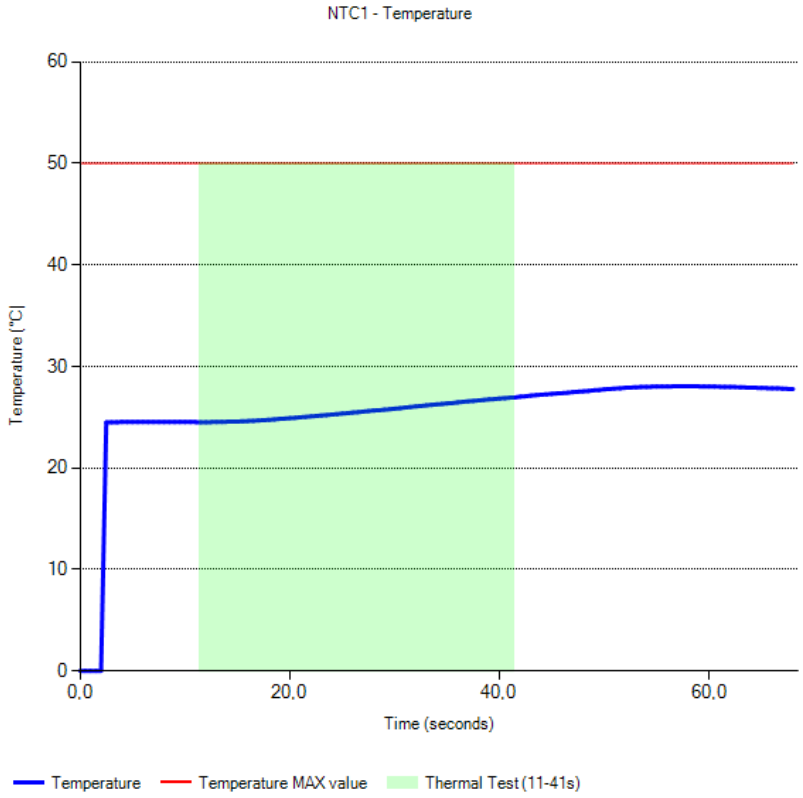
Z tohoto důvodu tato třída vykresluje i druhý typ grafu – napěťový drift. Princip je jednoduchý, třída *TestResult* obsahuje záznam, kdy započal a skončil termální test. Třída *TestResultDetail* si tedy najde tento čas a podle něj si najde podle času nejbližší pozdější záznam naměřeného napětí. Časový otisk a hodnotu napětí tohoto záznamu si určí jako počáteční bod a všechny následné hodnoty vykresluje do grafu ve vztahu vůči tomuto počátečnímu body (vykresluje tedy časový a napěťový rozdíl oproti počátečnímu bodu) do doby, nežli termální test skončil (30s). Napěťový drift Grafu 6 pak vypadá následovně.



Graf 7 - Webová aplikace – Graf napěťového driftu

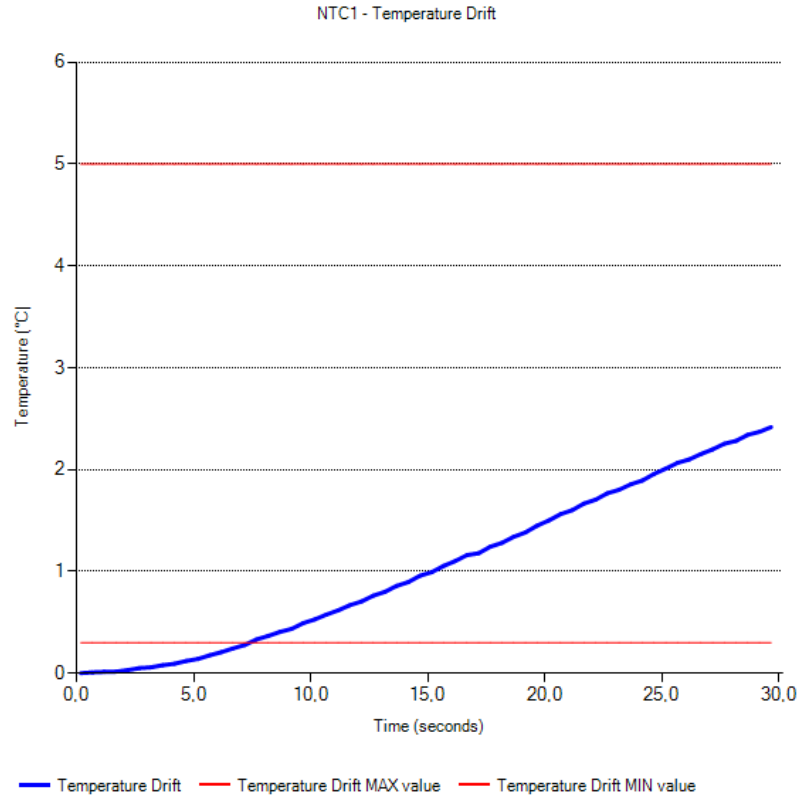
Jak si můžeme všimnout, napěťový výkyv (nyní v čase 10s až 11s) je nyní mnohem výrazněji vidět. Konkrétně tento případ pracovníci AL vyhodnocují jako anomálii, správný napěťový drift by mohl mít první 1-2s mírný nárůst napětí oproti počátečnímu bodu, ale následně by měl postupně klesat. Tuto anomálii vzhledem k hodnotám rozdílového napětí je možné vysvětlit například rušením měření, či manipulací s měřící soustavou.

Dalším typem grafů je absolutní hodnota teploty. Jedná se o jednoduchý graf, kde je vykreslována absolutní hodnota teploty a stejně, jako v případě typu grafu absolutní hodnoty napětí a proudu, je zde vyznačena oblast, kdy byl provádět termální test (viz Graf 8).



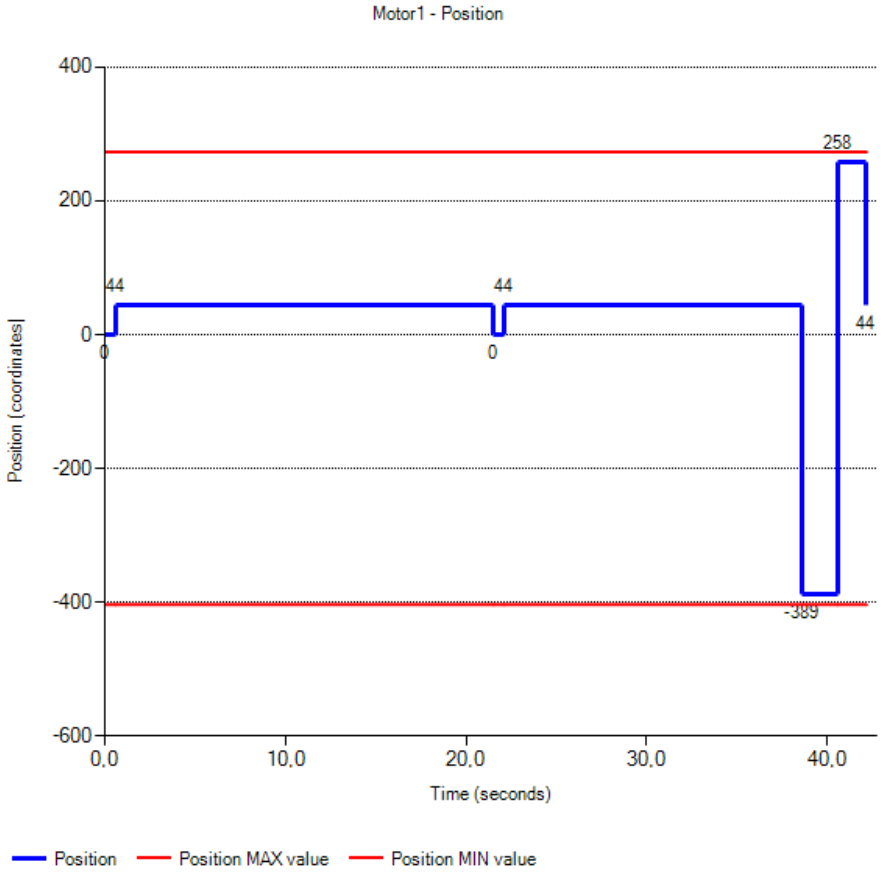
Graf 8 - Webová aplikace - Graf absolutní hodnoty teploty

Následuje typ grafu znázorňující teplotní drift. Princip je naprosto stejný, jako v případě napěťové driftu, pouze se vykreslují místo hodnot napětí hodnoty teploty, viz Graf 9.



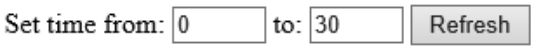
Graf 9 - Webová aplikace - Graf teplotního driftu

Posledním typem grafů je ten, který zobrazuje pozici motorů LED modulu v čase. SW LED Module Check totiž může provádět referenční či rozsahové testy těchto motorů, nebo je uživatel může přímo ovládat, viz Graf 10.



Graf 10 - Webová aplikace - Graf pozic motorů

Nutno ještě dodat, že každý graf u sebe obsahuje malé kontrolní centrum, které slouží k vykreslení požadovaného časového úseku průběhu testu. To se zejména hodí, pokud test trval abnormálně dlouho, například 300s a nejzajímavější část testu (termální test) se v grafech s absolutními hodnotami ztrácí.



Obrázek 18 - Webová aplikace - kontrolní centrum grafu

Tímto je popis grafů dokončen a můžeme si popsat jednotlivé funkce, objekty a třídy ve třídě *TestResultDetail*.

```
public TestResult DetailTestResult;
```

Prvním objektem je deklarace objektu *DetailTestResult*, třídy *TestResult*. Tento objekt slouží k načtení a uchování dat zvoleného záznamu měření, třída *TestResultDetail* tento objekt inicializuje později.

```
public List<ControlPanel> ControlPanels = new List<ControlPanel>();
```

Tento list objektů slouží ke generování kontrolních center u každého grafu. Definice třídy *ControlPanel* je uvedena níže.

```
public bool PrintPDF;
```

Tato deklarace globální proměnné *PrintPDF* slouží k tomu, aby třída dokázala určit, zda se aktuální webový obsah vykresluje pro zobrazení uživateli, nebo pro tisk do PDF, více o tisku do PDF a úpravách je uvedeno níže.

```
public class ControlPanel
```

Definice třídy *ControlPanel*, která obsahuje pouze dva objekty typu *TextBox* pro textové znázornění a případného určení časového úseku, který má být u daného grafu zobrazen. Pro pozdější jednoznačné určení, jaký kontrolní panel patří ke kterému grafu, obsahuje tato třída i deklaraci proměnné *name*, typu *string*, kam se ukládá jméno grafu.

```
List<Chart> Charts = new List<Chart>();
```

Tento list objektů tříd *Chart* slouží k samotnému ukládání grafů.

```
protected void Page_Load(object sender, EventArgs e)
```

Jak jsem již dříve uvedl, tato funkce je automaticky volána před vygenerováním webové stránky. V tomto konkrétním případě plní několik funkcí – v první řadě ověří, zda je aktuální webová stránka volána s parametrem *id* určující jaký záznam měření (*TestResult*) se má zobrazit. Pokud tento parametr chybí, stránka uživatele automaticky přesměruje na webový obsah třídy *TestResultSelectData*, kde může provést výběr. Pokud ale parametr id existuje, tak tato funkce provede inicializaci již zmíněného objektu *DetailTestResult* pomocí funkce *LoadTestResultFromDatabase* s parametry id a *LoadAll = true* pro úplné načtení objektu. Dále si funkce pomocí globální proměnné *PrintPDF* zjistí, zda se stránka zobrazuje pro uživatele nebo PDF a podle toho upraví parametry zobrazování. Následně si tato funkce zavolá funkce pro zobrazení/vykreslení hlavičky, výsledků měření a grafů daného záznamu měření.

```
public void TestResultHeader()
```

Funkce pro výpis hodnot záznamu měření do hlavičky webového obsahu.

```
public void DataCharts()
```

Tato funkce se stará o přidávání (generování) instancí objektů třídy *Chart* do listu *Charts*. Její princip spočívá v tom, že cyklicky prochází veškeré objekty tříd *LedResult*, *TempResult* a *MotorResult* obsažené v aktuálním *TestResult* objektu a jejich naměřené hodnoty zapisuje do instancí objektů třídy *Chart*. Krom těchto hodnot do tohoto objektu zapíše i základní informace o daném měření.

Po tomto vytvoření a naplnění grafů se funkce postará i o vykreslení grafů do webové stránky. Zde se již kontroluje, zda se stránka zobrazuje pro uživatele nebo PDF. V prvním případě se grafy vykreslují do tří sloupců se standardním popisem a kontrolním centrem. V případě PDF se grafy vykreslují pouze do dvou sloupců (aby se na list A4 vykreslily v dostatečné velikosti), zvětší font popisků u grafů a nezobrazí kontrolní centrum.

```
public double returnDoubleFromTextbox(string text)
```

Funkce, která z parametru text získá hodnotu typu double a tuto hodnotu vrátí. Využívá se pro získání číselných hodnot z textboxů v kontrolních centrech pod grafy.

```
public string returnColor(STATUS.StatusEnum StatusEnum)
```

Funkce, která vrátí hexadecimální hodnotu barvy podle parametru *StatusEnum*. Používá se pro lepší grafické znázornění výsledku textu. Pokud je například hodnota *StatusEnum* = Ok, pak funkce vrátí zelenou barvu. Pokud je hodnota rovna *Error*, pak funkce vrátí červenou barvu. V ostatních případech vrací oranžovou barvu.

```
public void TestResutltInfo()
```

Tato funkce vypisuje výsledky testů na webovou stránku. Cyklicky prochází objekty tříd *HWResult*, *LedResult*, *TempResult*, *FanResult* a *MotorResult* v *DetailTestResult* a vypisuje jejich hlavní hodnoty, výsledky a limity do tabulky.

```
public string returnStatusString(STATUS input)
public string returnBinningOhmString(double input)
```

Tyto třídy slouží pouze k formálním výpisům hodnot. V případě funkce *returnStatusString* se nám vrátí vždy textová hodnota parametru input, ovšem pokud je *input.value* = *Error*, tak nám funkce vrátí *Error* včetně chybového kódu.

```
protected void ButtonRefresh_Click(object sender, EventArgs e)
```

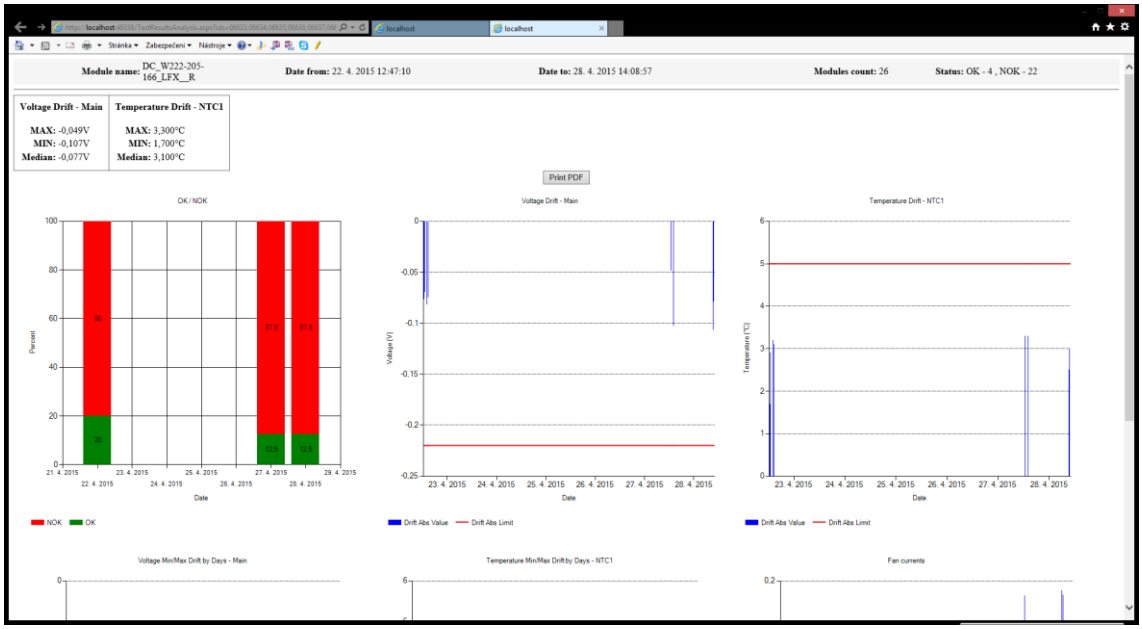
Funkce volaná tlačítky *Refresh*. Slouží k opětovnému vykreslení grafů s požadovanými rozsahy časové osy.

```
protected void ButtonPdf_Click(object sender, EventArgs e)
```

Tato funkce se volá po stisknutí tlačítka PDF. Její funkce spočívá v tom, že používá knihovnu *HtmlToPdf* k vygenerování PDF reportu. Nejprve nadefinuje potřebné parametry (formát, orientaci, šířku/výšku v pixelech, odkaz atd.) a poté zavolá funkci pro konverzi odkazované stránky a výsledný PDF soubor uloží.

6.5.5 TestResultAnalysis

Třída s webovým obsahem sloužící k zobrazení statistik nad vybranými záznamy měření.

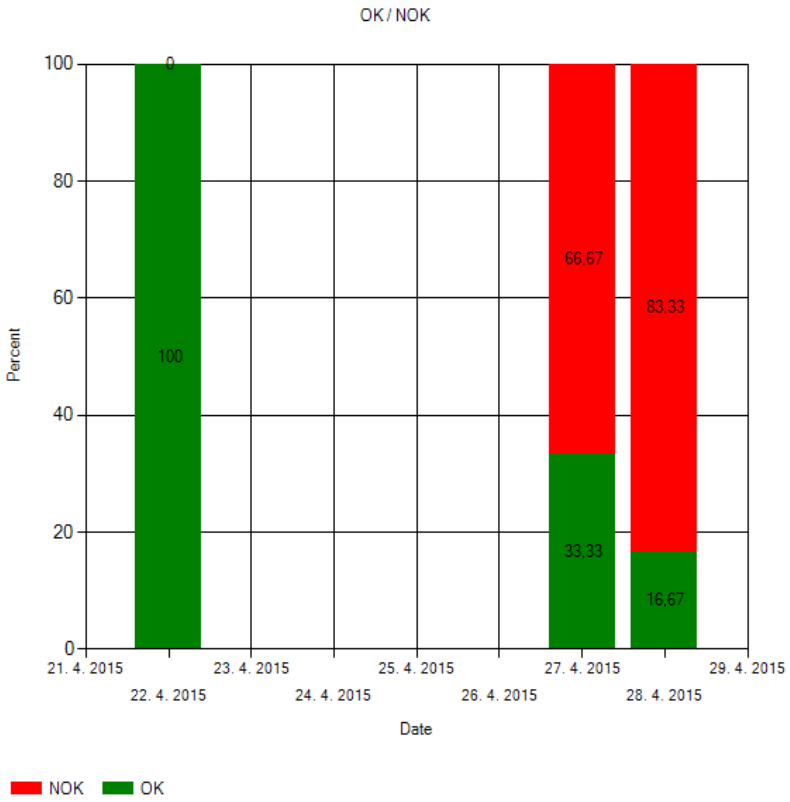


Obrázek 19 - Webová aplikace – TestResultAnalysis

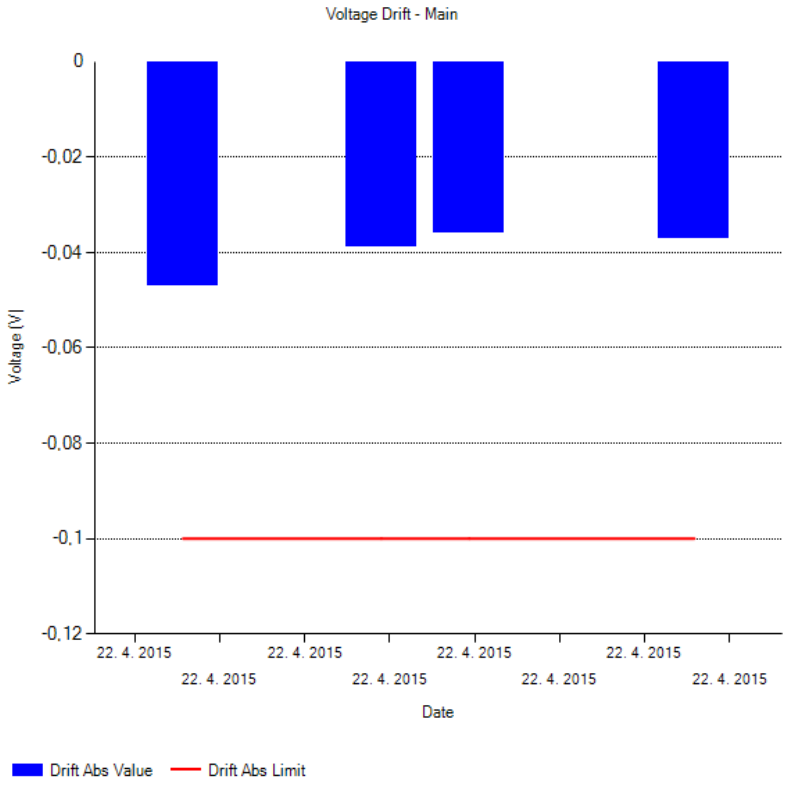
Koncept webového obsahu v podstatě kopíruje ten ze třídy *TestResultDetail*. Opět je zde hlavička, ve které jsou uvedené základní informace o analyzovaných měření, následuje tabulka s naměřenými a vyhodnocenými parametry testů (zejména napěťových a teplotních driftů), tlačítko tisku trendů do PDF a nakonec vygenerované grafy.

Grafy lze rozdělit do tří typů. Prvním je typ zobrazující procentuální zastoupení provedených měření podle jejich celkových výsledků (viz Graf 11). Díky tomuto grafu lze rychle získat přehled, jaká je celková úspěšnost testů. Předpoklad pro využití tohoto grafu je takový, že podle něj bude možné určit souvislosti mezi výsledky měření a okolními vlivy. Například pokud se při výrobě LED modulů začne používat jiný druh teplovodivé pasty s nižší účinností, která přesáhne tolerovanou hranici, zvýší se tak chybovost testů. Uživatel tak bude schopen přesně zjistit okamžik, kdy se tato chybovost zvýšila a podle tohoto okamžiku začít hledat souvislosti. V našem případě by dříve nebo později zjistil, že v daný okamžik se přešlo na jiného dodavatele teplo vodivé pasty.

Dalším typem grafů je zobrazující výslednou hodnotu driftů. Jedná se o sloupcový graf této hodnoty v závislosti na čase (viz Graf 12). Uživatel je tak schopen sledovat jak se vyvíjí tato hodnota a opět podle těchto hodnot určovat různé souvislosti.

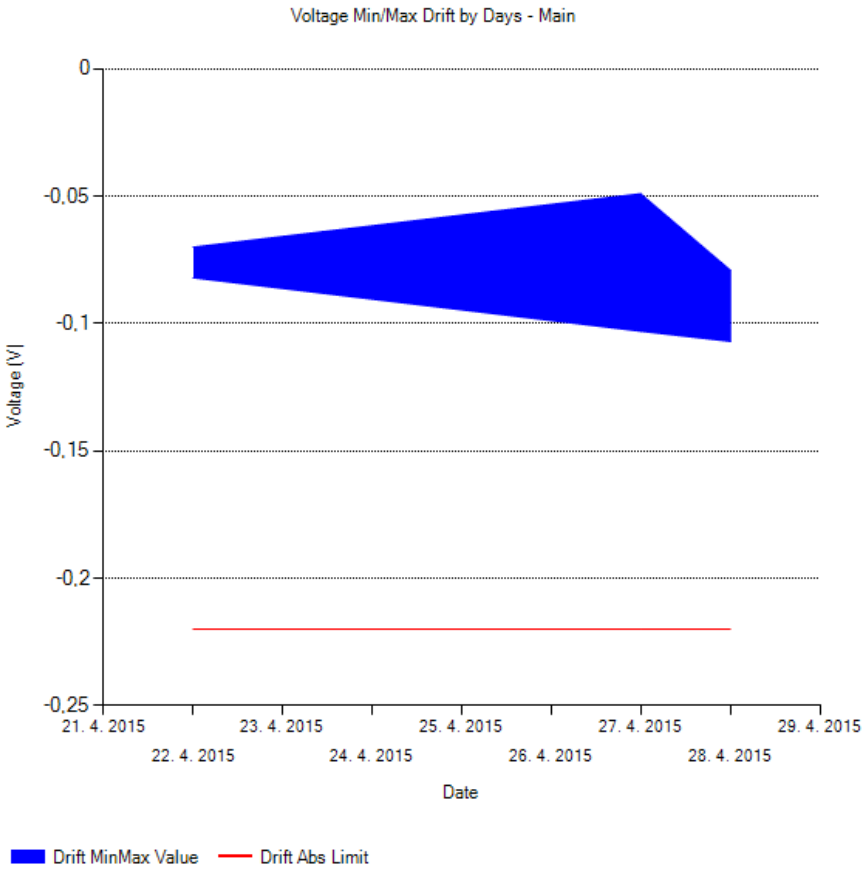


Graf 11 - Webová aplikace - Zobrazení výsledků měření



Graf 12 - Webová aplikace - Výsledky driftů

Posledním typem je graf zobrazující rozsah, ve kterém se pohybovaly výsledky driftů v závislosti na čase, konkrétně po dnech (viz Graf 13).



Graf 13 - Webová aplikace - Rozsahy výsledků driftů

Nyní k objektům, funkcím a třídám, které třída *TestResultAnalysis* obsahuje.

```
public List<TestResult> TestResults = new List<TestResult>();
public List<DriftValue> DriftValues = new List<DriftValue>();
public bool PrintPDF;
```

List *TestResults* slouží k uchovávání všech instancí objektů třídy *TestResult*. List *DriftValues* slouží k vykreslování všech výsledných hodnot driftů (rozsahy v závislosti na dnech). Proměnná *PrintPDF* je zde opět kvůli tisku do PDF.

```
public class OkNokPercentage
```

Třída pro vykreslení grafu výsledků měření. Obsahuje číselné vyjádření počtu OK a NOK výsledků testů a časový otisk, do kterého se ukládají pouze dny, měsíce a roky. Hodnoty hodin, minut, vteřin atd. jsou nastaveny na 0. Třída navíc definuje funkci Update s parametrem *result*. Tato funkce na základě hodnoty parametru *result* buď zvýší hodnotu položky OK nebo NOK.

```
public class DriftValue
```

Jedná se o univerzální třídu pro ukládání výsledných hodnot driftů jak pro napětí, tak pro teplotu. Třída umožňuje záznam typu driftu (napěťový/teplotní), název skupiny LED či NTC, časový otisk, výslednou hodnotu driftu, limit driftu, značení jednotky (V, °C) a hodnotu, zda daný záznam byl použit pro vykreslení do grafu. Dále třída obsahuje pouze parametrický konstruktor.

```
public class ValueTime
```

Třída využívaná při sběru dat z objektů třídy *DriftValue* při vykreslování grafů. Třída *DriftValue* totiž slouží v podstatě k uložení výsledných hodnot driftů tak, aby je bylo možno později rozlišit (podle typu, skupiny LED/NTC) a tato třída *ValueTime* slouží k uložení roztříděných hodnot z *DriftValue* a následně při vykreslování do grafů. Více uvedeno ve funkci *DataAnalysisCharts*.

```
public class MinMaxValueTime
```

Třída pro uchování rozsahu výsledků driftu v daný den. Obsahuje definici minimální a maximální hodnoty driftů a časový otisk. Dále parametrický konstruktor s časovým otiskem, který zaznamená pouze den, měsíc a rok. Následně je zde funkce *UpdateMinMax*, která má parametr *i_Value*. Tato funkce po zavolání zkontroluje, zda parametr je menší/větší než minimální/maximální hodnota v objektu a podle toho provede přiřazení.

```
protected void Page_Load(object sender, EventArgs e)
```

Funkce volaná při načítání stránky. Stejně jako v případě *TestResultDetail* i tato funkce nejprve provede kontrolu, zda je stránka volána s parametry identifikačních klíčů požadovaných výsledků měření. Pokud ne, stránka uživatele přesměruje na webové stránky *TestResultSelectData*. Pokud existují identifikační klíče, tak funkce provede kompletní nahrání dat těchto měření z databáze. Dále se kontroluje, zdali se stránka má vykreslit pro uživatele nebo PDF. Následně se vypíše hlavička testů, zavolají se funkce *DataAnalysisCharts* a *DriftInfo*.

```
public void DataAnalysisCharts()
```

Tato funkce slouží k vygenerování grafů. Prvním typem grafu, který se generuje, je již zmíněný graf výsledků testu (OK/NOK). Funkce si tedy deklaruje list objektů třídy *OkNokPercentage* a v logické smyčce tento list naplňuje. Prochází všechny záznamy z měření a podle data buď vytváří nový objekt, nebo do již vytvořeného zapíše výsledek. Po tomto procesu již probíhá generování grafu OK / NOK výsledků.
Dále funkce v dalších dvou logických smyčkách naplňuje list objektů typu *DriftValue*, do kterého přidává objekty cyklickým procházením objektů *LedResult* a *TempResult* každého záznamu měření.

Následuje generování dalších dvou typů grafů. Toto generování, podle mého názoru, si zaslouží pozornost, neboť je univerzální jak pro teplotní/napěťové drifty. Pro přehlednost uvádím algoritmus tohoto procházení:

```
foreach (DriftValue element in DriftValues)
{
  if (element.Used == false)
  {
    List<ValueTime> ValuesTimes = new List<ValueTime>();

    foreach (DriftValue element_search in DriftValues)
    {
       if ((element_search.DriftType == element.DriftType) &&
           (element_search.ModuleName == element.ModuleName) &&
           (element_search.Used == false))
       {
         ValuesTimes.Add(
              newValueTime(element_search.Value,element_search.TimeStamp));
         element_search.Used = true;
       }
    }
. . .
```

Výše uvedený algoritmus provádí to, že prochází každý objekt v listu *DriftValues*. U aktuálního objektu nejprve zkontroluje, zda tento objekt již nebyl použit při generování grafu (položka *Used*). Pokud je hodnota proměnné *Used* = *true*, znamená to, že objekt již byl použit, a proto se s objektem již dál nepracuje a přechází se k dalšímu objektu v listu. Pokud ale proměnná *Used* = *false*, pak objekt ještě nebyl použit. Pokud je tomu tak, algoritmus si následně vytvoří list objektů *ValuesTimes* třídy *ValueTime* a posléze se do tohoto listu snaží přidávat objekty tak, že prochází celý list *DriftValues* a hledá objekt, který má stejné hodnoty typu driftu, názvu skupiny LED/NTC jako aktuální objekt. Pokud takový objekt najde (prvním je logicky aktuální objekt), tak jeho hodnoty časového otisku a hodnoty výsledného driftu uloží do listu *ValuesTimes*. Po skončení se již generuje graf na základě tohoto listu a aktuálního objektu, který nese navíc informační hodnoty o typu driftu, názvu atd.
To samé se v podstatě děje u generování grafu rozsahu výsledků driftů, pouze se před vyhledáváním nejprve nastaví hodnoty *Used* na *false* všech objektu v listu *DriftValues* a nalezené hodnoty se ukládají do listu objektů třídy *OkNokPercentage*.
Nakonec se generuje graf s hodnotami proudů protékajících větráčkem. Zde se již procházejí přímo objekty *TestResult* a zaznamenává se tato hodnota s časovým otiskem.
Poslední částí této funkce je generování dynamické tabulky s těmito grafy, toto generování je naprosto shodné s tím, které bylo popsáno ve třídě *TestResultDetail*.

```
public void DriftInfo()
```

Funkce generující základní informace o driftech (celková MIN/MAX hodnota dané skupiny LED/NTC a modus). V současnosti neobsahuje další údaje, neboť na konkrétnější analýzy není dostatek naměřených dat.

```
public void FindMinMax(List<double> input, ref double Min, ref double Max)
```

Funkce, která z listu hodnot input uloží do parametrů Min, Max dané hodnoty.

```
public double returnMedian(List<double> input)
```

Funkce vracející hodnotu medianu z listu hodnot input.

```
public string returnFormalString(double Value, int accuracy)
```

Funkce vracející formální podobu hodnoty parametru *Value* s přesností zadanou parametrem *accuracy*.

```
protected void ButtonPdf_Click(object sender, EventArgs e)
```

Totožná funkce s funkcí *ButtonPdf_Click* ze třídy *TestResultDetail*

6.6 Ověření zobrazovaných dat

Ověřování dat, která výsledná webová aplikace zobrazovala ať již v textové, nebo grafické podobě, jsem prováděl těmito způsoby:

- Porovnání výsledných grafů s grafy, které byly již dříve ručně vytvořeny pracovníky AL v MS Excel dle naměřených hodnot.
- Předložení výsledných grafů pracovníkům AL, kteří díky zkušenosti potvrdili, že výsledná charakteristika grafů je správná.
- Přímým porovnáním hodnot v XML souboru a zobrazovaných hodnot. Zde jsem si ve webové aplikaci zobrazil detailní výsledek měření a vyhledal jsem si XML soubor, ze kterého bylo měření nahráno do databáze. Porovnáním všech číselně vyjádřených hodnot a následně několika vybraných hodnot z grafů jsem určil, že se zobrazovaná data a data v XML shodují.
- Vytvořením dvou objektů třídy *TestResult*. Zatímco data do prvního objektu jsem nahrál pomocí deserializace XML, data do druhého objektu jsem nahrál z databáze. Jednalo se o záznam ze stejného měření a porovnáním všech datových položek v objektu jsem shledal, že jsou objekty identické a tudíž se při ukládání dat do databáze a čtení z databáze neztratila žádná data.

Veškeré ověřování proběhlo úspěšně a tak lze tvrdit, že databáze, program i výsledný grafický výstup programu jsou správně vytvořeny a nedochází ke ztrátám dat.

7 ZÁVĚR

Cílem mé diplomové práce bylo vytvořit funkční webovou aplikaci pro sběr a vyhodnocování dat z testovacích stanic LED modulů, které jsou obsaženy v předních automobilových LED světlometech.

V úvodu jsem se zabýval principem LED, jejich vlastnostmi a způsoby využití, následně konstrukcí LED světlometů a samotným procesem měření LED modulů. Popis měřícího nástroje jsem zkrátil pouze na velmi obecný základ, neboť tento nástroj byl vyvinut přímo Jihlavským zastoupením firmy AL a z důvodu ochrany duševního vlastnictví firmy mi nebylo dovoleno tento nástroj detailněji popsat. Rád bych zde ale dodal, že jsem sestavoval jeden měřící nástroj a mohu tak potvrdit, že právě jihlavské zastoupení firmy AL bylo právem vybráno jako jedno z hlavních center vývoje celé skupiny AL.

V další části jsem vysvětlil průběh návrhu a realizace tříd v programovacím jazyce C# a proces tvorby databáze prostřednictvím SQL, abych tak následně mohl lépe popsat veškeré činnosti spojené s aspekty tvorby webové aplikace, které jsem učinil.

V poslední části mé diplomové práce jsem již popisoval tvorbu webové aplikace, její funkční části, webový obsah a i velmi zajímavé problémy, na které jsem při tvorbě narazil, včetně odůvodnění vzniku a následného řešení.

Výsledná webová aplikace je plně funkční, veškeré požadavky na její vlastnosti jsou splněny. Dalším krokem bude nasazení webové aplikace na virtuální server, na kterém bude probíhat testovací fáze, při které budou vznikat další požadavky z řad zaměstnanců AL na další úpravy a nové vlastnosti.

Již v současnosti ale tato webová aplikace odhalila několik neobvyklých výsledků měření, kdy například teplota na NTC v LED modulu kolísala až o 10°C v řádu milisekund nebo například průběh napěťového driftu neodpovídal předpokládanému. První případ byl vysvětlen vadným NTC a druhý případ se přikládá za vinu okolnímu rušení například v podobě blízkosti mobilního telefonu nebo manipulací v okamžik termálního testu.

Jako další možnost rozšíření funkční části webové aplikace by bylo možné uvažovat přidání funkce pro přímého porovnání dvou a více měření, konkrétně naměřených hodnot v čase měření, čímž by bylo možné naprosto přesně určit vliv různých změn (změna teplovodivé pasty, tvaru chladiče, aktivního chlazení atd.) na vlastnosti LED modulu.

Závěrem bych rád dodal, že všechny cíle diplomové práce byly splněny a že celou diplomovou práci považuji za velmi přínosnou jak pro mě osobně, tak i pro firmu AL. Díky tomuto nástroji jsou zaměstnanci AL již schopni sbírat a analyzovat data z testovacích stanic a dle prvních reakcí si troufám usuzovat, že díky této webové aplikaci se celý proces měření LED modulů posunul o stupeň výše. Pro mě osobně byl přínos hlavně v tom, že jsem získal cenné zkušenosti, jak probíhá celý proces tvorby programu v nadnárodní společnosti – od prvního kroku v podobě upřesnění zadání, přes týmovou práci na návrhu společné třídy, až po závěrečné ladění a drobné úpravy webové aplikace.

8 LITERATURA

[1] WEB Automotive Lighting (12/2014).
Dostupné na URL: http://www.al-lighting.cz/

[2] Automotive Lighting (2014). Interní firemní literatura.

[3] Ing. Vladimír Dvořáček, S. L. (2009). Světelné zdroje - světelné diody. SVĚTLO

[4] Ing. Zbyněk Bureš, P. (2014). Databázové systémy. Vysoká škola polytechnická, Jihlava.

[5] Lumileds (2015). LUXEON Altilon Datasheet.
Dostupné na URL: http://www.lumileds.com/uploads/39/DS66-pdf

[6] Microsoft (2015). Developer Network.
Dostupné na URL: https://msdn.microsoft.com/

[7] ITnework.cz (2015). Sociální síť pro IT profesionály.
Dostupné na URL: http://www.itnetwork.cz/

[8] Lumileds (2014). LUXEON Rebel Datasheet.
Dostupné na URL: http://www.lumileds.com/uploads/20/DS63-pdf

[9] BÍLÁ, J., KRÁL,F (1999). Databázové a znalostní systémy, Skriptum ČVUT FS, Praha

[10] EVJEN, B., HANSELMAN S., RADER D. (2009). ASP.NET 3.5 v jazycích C# a Visual Basic [1] Computer Press. ISBN 978-80-251-2069-9.

[11] EVJEN, B., HANSELMAN S., RADER D. (2009). ASP.NET 3.5 v jazycích C# a Visual Basic [2] Computer Press. ISBN 978-80-251-2069-9

[12] VIRIUS, M. (2005). C# Hotová řešení, Computer Press. ISBN 80-251-1084-2

9 SEZNAM OBRÁZKŮ

10 SEZNAM GRAFŮ

11 SEZNAM TABULEK

12 SEZNAM SCHÉMAT

13 SEZNAM ZKRATEK

AL - Automotive Lighting
ASP - Active Server Pages
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
HW - HardWare
ID - IDentification
LED - Light Emitting Diode
MSSQL - Microsoft SQL
NTC - Termistor
SQL - Structured Query Language
SW - SoftWare
XML - eXtensible Markup Language

14 SEZNAM PŘÍLOH

Příloha 1 - Zdrojové kódy třídy TestResult

```
public class TestResult
{
   public string moduleName = string.Empty;              //type of the tested module
   public string moduleDescription = string.Empty;       //additional information
   public string moduleTTNr = string.Empty;              //TTNr number Scanned
   public string moduleSN = string.Empty;                //serial number
   public string moduleNewTTNr = string.Empty;           //new TTNr number (printed at new DMC label)
   public string moduleNewSN = string.Empty;             //new serial number (printed at new DMC label)
   public bool newDmcLabelVerificationFlag = false;      //new (printed) DMC label verification flag
   public string userName = string.Empty;                //user name
   public string computerName = string.Empty;            //computer name
   public string swVersion = string.Empty;               //SW version
   public DateTime Timestamp = DateTime.MinValue; //time of the module test beginning

   public STATUS result = new STATUS();                  //final test result

   public AmbTemperature ambientTemperature = new AmbTemperature();        //ambient temperature
   public HWResult hwResult = new HWResult();                              //HW result

   public List<LedResult> ledResults = new List<LedResult>();             //list of LED(s) results
   public List<TempResult> teperatureResults = new List<TempResult>(); //list of NTC(s) Temperature results
   public FanResult fanResult = null;                                      //FAN result
   public List<MotorResult> motorResults = new List<MotorResult>();       //list of Motor(s) results
   public List<LightDistridutionTest> LightDistridution = null;           //list of Light Distridution
   public VisualInspection VisualInspection = null;                        //Visual Inspection test results
   public List<TechnoTeamResult> technoTeamResults = new List<TechnoTeamResult>(); //list TechnoTeam
}

public class AmbTemperature
{
   public DateTime timeStamp = DateTime.MinValue;        //time measurement
   public float Value = float.MinValue;                  //measured ambient temperature
}

public class HWResult
{
   public string cDaqSN = string.Empty;                  //cDaq serial number
   public string hwConfig = string.Empty;                //hardware configuration file name
   public STATUS LastHwStatus = new STATUS();            //last HW STATUS
}

public class MotorResult
{
   public string name;                          //motor name
   public STATUS STATUS = new STATUS();         //STATUS
   public Int32 endPos1 = 0;                    //end position in dirrection1
   public Int32 endPos2 = 0;                    //end position in dirrection2
   public List<refResult> refResults = new List<refResult>();          //list of motor reference results
   public List<rangeResult> rangeResults = new List<rangeResult>();    //list of motor range results
   public List<MotorPos> positions = new List<MotorPos>();             //list of motor req positions results
}

public class refResult
{
   public DateTime timeStamp = DateTime.MinValue;        //time of the motor reference execution
   public bool result = false;                           //motor reference result
}
```

```
public class rangeResult
{
   public DateTime timeStamp = DateTime.MinValue;      //time of the motor range measurement
   public Int32 posRange1 = 0;                  //measured motor range end position value in dirrection1
   public Int32 posRange2 = 0;                  //measured motor range end position value in dirrection2
   public bool result = false;                  //result of the motor range measurement
}

public class MotorPos
{
   public DateTime timeStamp = DateTime.MinValue;  //time of the motor position request execution
   public string name = String.Empty;              //name of the requested motor position
   public double value = 0;                        //value in HS of the requested motor position
   public bool result = false;                     //result of the motor position request execution
}

public class FanResult
{
   public STATUS LatestStatus = new STATUS();      //last FAN STATUS
   public DateTime timeStamp = DateTime.MinValue;  //time of FAN current measurement
   public Limits currentLimits = new Limits();     //limits for FAN current measured value
   public double measCurrentFirst;                 //measured FAN current value
}

public class LedResult
{
   public string name = String.Empty;              //LED name
   public STATUS STATUS = new STATUS();                    //LED STATUS

   //limits - min, max, drift voltage, drift measured
   public string binning = string.Empty;           //LED binning name
   public double binningOhm;                       //LED binning resistor measured value
   public double operatingCurrent;                 //LED required operation current value

   public Limits limitsVoltageAbsVal = new Limits();   //limits for LED voltage (if LED is switched on)
   public Limits limitsVoltageDrift = new Limits();    //limits for LED voltage drift durring Thermal Test

   public double measCurrentThermTest1 = 0;        //LED current value Thermal Test started
   public double measVoltageThermTest1 = 0;        //LED voltage value Thermal Test started
   public DateTime timeStampThermTest1 = DateTime.MinValue; //time when Thermal Test started

   public double measCurrentThermTest2 = 0;        //LED current value when Thermal Test finished
   public double measVoltageThermTest2 = 0;        //LED voltage value when Thermal Test finished
   public DateTime timeStampThermTest2 = DateTime.MinValue; //time when Thermal Test finished

   public double measVoltageDrift = double.MaxValue;   //measured/calculated LED voltage drift durring
Thermal Test (First Usage Test)

   public List<LedValue> values = new List<LedValue>(); //list of LED measured values
}



public class LedValue
{
   public STATUS STATUS = new STATUS();                    //measured LED value STATUS
   public DateTime timeStamp = DateTime.MinValue;  //time of LED values measurement
   public double voltage = double.MinValue;        //measured LED voltage value
   public double current = double.MinValue;        //measured LED current value
}
```

```
public class TempResult
{
   public string name = String.Empty;            //NTC resistor name
   public STATUS STATUS = new STATUS();          //NTC temperature STATUS
   public Limits limitsAbsVal = new Limits(); //limits for NTC temperature
   public Limits limitsDrift = new Limits();     //limits for NTC temperature drift durring Thermal Test

   public double measTempThermTest1 = 0;   //LED voltage value when Thermal Test started
   public DateTime timeStampThermTest1 = DateTime.MinValue; //time when Thermal Test started

   public double measTempThermTest2 = 0;             //LED voltage value when Thermal Test finished
   public DateTime timeStampThermTest2 = DateTime.MinValue; //time when Thermal Test finished

   public double measDrift = double.MinValue;        //measured/calculated NTC temperature drift durring
thermal test

   public List<TempValue> values = new List<TempValue>();  //list of NTC temperature values
}
public class TempValue
{
   public STATUS STATUS;          //measured NTC temperature value STATUS
   public DateTime timeStamp;     //time of NTC temperature measurement
   public double value;           //measured NTC temperature value
}

public class Limits
{
   public double min = double.MaxValue;          //minimal specified value
   public double max = double.MinValue;          //maximal specified value
}
```

Příloha 2 - Zdrojové kódy třídy STATUS

```
public class STATUS
{
    public enum StatusEnum
    {
        NotUsed = 3,
        NotDefined = 2,
        OK = 1,
        ERROR = 0
    };

    private StatusEnum enValue;
    public int ErrorCode { get; set; }
    public int LastHandledErrorCode { get; set; }
    public string ErrorText { get; set; }
    public DateTime TimeOfLastStatusChange { get; set; }
    public bool ModuleTestConditionPass = false;

    // Undefined Error
    public const int ErrorCode_NotDefError = 0;

    // Fan Error
    public const int ErrorCode_Fan_WrongCurrentVal = 1010;

    // LED Errors
    public const int ErrorCode_LED_BinnResNoValidVal = 2000;
    public const int ErrorCode_LED_BinnResTooHighVal = 2001;
    public const int ErrorCode_LED_Binn2DCod_ResDiff = -2003;
    public const int ErrorCode_LED_TooHighCurrent = 2020;
    public const int ErrorCode_LED_TooHighVoltage = 2030;
    public const int ErrorCode_LED_TooHighVoltageDec = 2131;
    public const int ErrorCode_LED_TooHighTmptre = 2040;
    public const int ErrorCode_LED_TooHighNtcResVal = 2041;
    public const int ErrorCode_LED_TooHighTmptreIncH = 2142;
    public const int ErrorCode_LED_TooHighTmptreIncL = 2143;
    public const int ErrorCode_LED_NotReallyOffPcbEr = 2220;
    public const int ErrorCode_LED_CathodeVoltgPcbEr = 2230;
    public const int ErrorCode_LED_VoltageRgndPcbEr = 2240;

    // Motor(s) Errors
    public const int ErrorCode_Motor_ReferenceErr = 4020;
    public const int ErrorCode_Motor_RangeErr = 4040;
    public const int ErrorCode_Motor_RangeErrEP1 = 4041;
    public const int ErrorCode_Motor_RangeErrEP2 = 4042;

    // TTT Errors
    public const int ErrorCode_TTT_ErrorMinValue = 5000;
    public const int ErrorCode_TTT_Failed = 5001;
    public const int ErrorCode_TTT_CommExcp = 5080;
    public const int ErrorCode_TTT_CommTimeout = 5081;
    public const int ErrorCode_TTT_UndefExcp = 5098;
    public const int ErrorCode_TTT_ErrorMaxValue = 5099;

    // REMAN Errors
    public const int ErrorCode_REM_ErrorMinValue = 5100;
    public const int ErrorCode_REM_UndefExcp = 5198;
    public const int ErrorCode_REM_ErrorMaxValue = 5199;
```

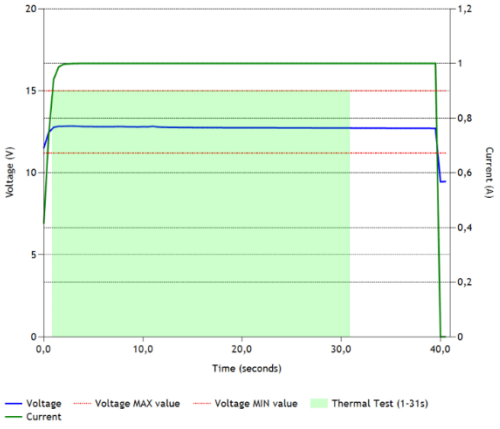
```
    // Visual Tests Errors
    public const int ErrorCode_LightDistrb_Error = 5210;
    public const int ErrorCode_VisualInsp_Error = 5220;

    // LED Module Check HW Errors
    public const int ErrorCode_HW_DynMeasuremtBinErr = 8209;
    public const int ErrorCode_HW_PowerCurrentVal = 8220;
    public const int ErrorCode_HW_PowerVoltageVal = 8230;
    public const int ErrorCode_HW_DynMeasuremtNtcErr = 8249;
    public const int ErrorCode_HW_StepperMotorInitErr = 8410;

    // LED Module Check SW Errors
    public const int ErrorCode_SW_UndefErr = 9000;
    public const int ErrorCode_SW_BinChooseFormExcp = 9001;
    public const int ErrorCode_SW_CfgConsistErr = 9002;
    public const int ErrorCode_SW_2DCodeDoPrListExcp = 9003;
    public const int ErrorCode_SW_2DCodeRcvExcp = 9004;
    public const int ErrorCode_SW_2DCodeProcessExcp = 9005;

    public const int ErrorCode_SW_FanNotDefExcp = 9100;

    public const int ErrorCode_SW_BinNtcNotDefExcp = 9110;
    public const int ErrorCode_SW_BinNtcThdStartExcp = 9111;

    public const int ErrorCode_SW_LedxNotDefErr = 9200;
    public const int ErrorCode_SW_LedxThdStartExcp = 9201;
    public const int ErrorCode_SW_LedxBinnMeasExcp = 9202;
    public const int ErrorCode_SW_LedIsNotOnErr = 9211;
    public const int ErrorCode_SW_LedIsNotOffErr = 9212;

    public const int ErrorCode_SW_MotorsDrvInitExcp = 9401;
    public const int ErrorCode_SW_MotorsThdStartExcp = 9402;
    public const int ErrorCode_SW_MotorSetParamErr = 9403;
    public const int ErrorCode_SW_MotorGetSensorExcp = 9404;
    public const int ErrorCode_SW_MotorGoPosExcp = 9405;

    public const int ErrorCode_SW_TestGuiStateErr = 9501;

    public const int ErrorCode_SW_HwCfgLedSuplNotDef = 9601;

    // No Errors - Just helps codes
    public const int ErrorCode_NoErr = -100;
    public const int ErrorCode_NoErr_AmbTempNotKnown = -2010;
    public const int ErrorCode_NoErr_AmbTempOutOfRng = -2011;
    public const int ErrorCode_NoErr_TempTooDiff2Amb = -2015;
    public const int ErrorCode_NoErr_NotAllLEDsAreOn = -2020;
    public const int ErrorCode_NoErr_MotorsDrvNull = -4010;
}
```

Příloha 3 – PDF report Detail

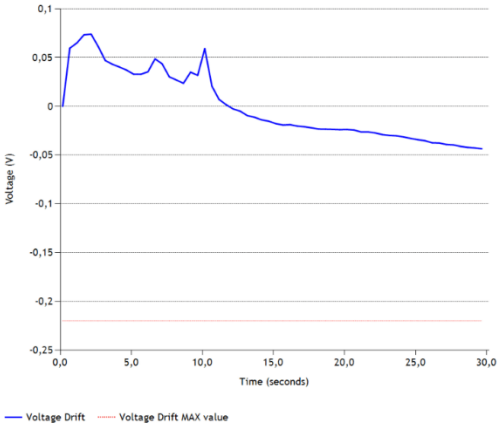
Module Name: DC_W222-205-166_LFX__R
Module SN: 1307023809001902140741
Module new SN: R307023809001902140741
Module TTNr: 1307023809
Module new TTNr: R307023809
Module Desc: RE ECE RHT (ALDE-BD)
Result: OK
Time: 28. 4. 2015 9:33:34
Ambient Temperature: 23,7°C
Computer name: LT0071202
User name: MMEMEA\F80200A
HW ID: 17802F9
SW version: 1.1.13.3

HW Status	LED Result - Main	Temperature Result - NTC1	Fan Result	Motor Result - Motor1	Motor Result - Motor2
Status: OK **HW Config:** HW_Config **HW ID:** 17802F9	**Result:** OK **Binning:** W **Binning Resistor:** NotUsed **Operating Current:** 1,0000 A **Voltage Drift:** -0,1070 V **LIMITS: MAX MIN** **Voltage Abs (V):** 15,0 11,2 **Voltage Drift (V):** -0,2 0,0	**Result:** OK **Temperature Drift:** 2,5 °C **LIMITS: MAX MIN** **Temp Abs (°C):** 50,0 **Temp Drift (°C):** 5,0 0,3	**Result:** OK **First Current:** 0,1890 A **LIMITS: MAX MIN** **Current (A):** 0,035 0,300	**Result:** OK **Measured Range:** End Position OUT: -422 Hs End Position IN: 304 Hs **MIN REQUIRED RANGE:** End Position OUT: -404 Hs End Position IN: 273 Hs	**Result:** OK

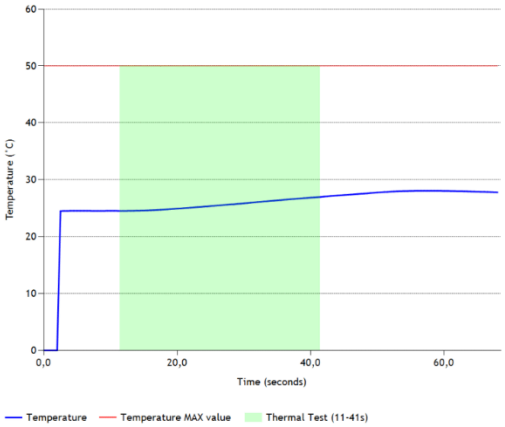
LED Main - Voltage and Current



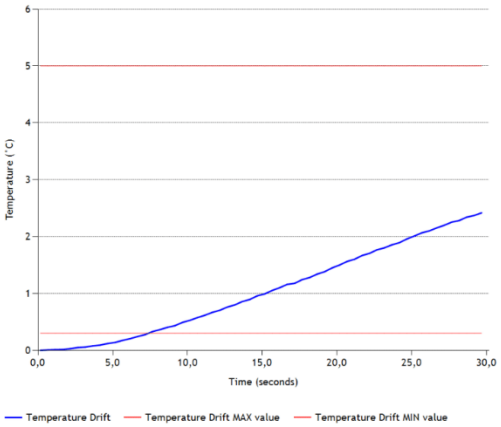
LED Main - Voltage Drift



NTC1 - Temperature

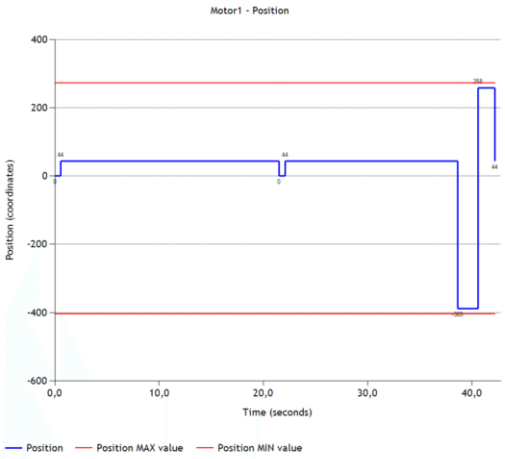


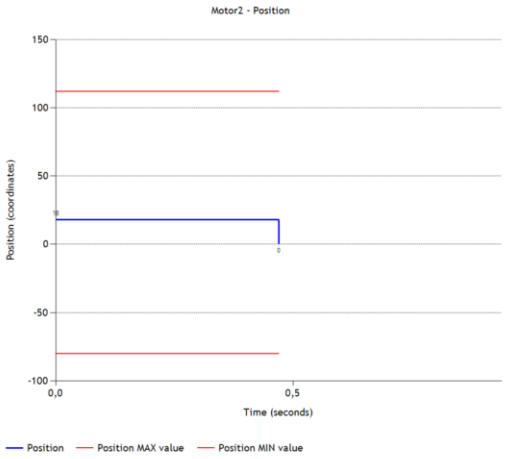
NTC1 - Temperature Drift



Page: 1 of 2

MODULE: DC_W222-205-166_LFX__R - SN: 130702380900190214074 1 - TTNr: 1307023809 - SW: 1.1.13.3 - DATE: 28. 4. 2015 9:33:34

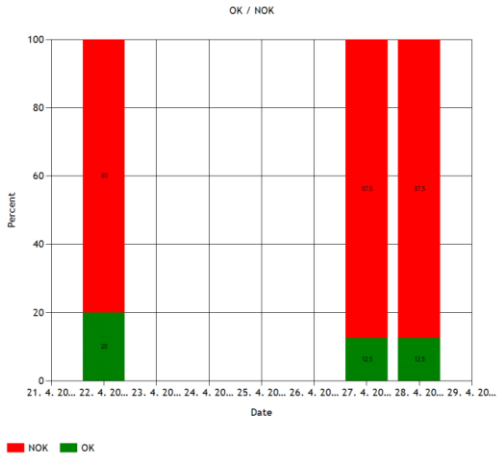


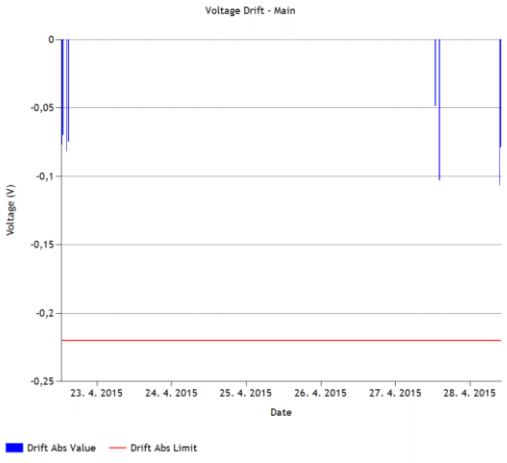


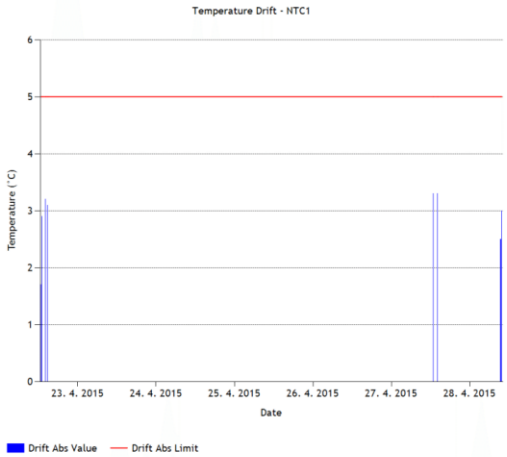
Příloha 4 – PDF report Analysis

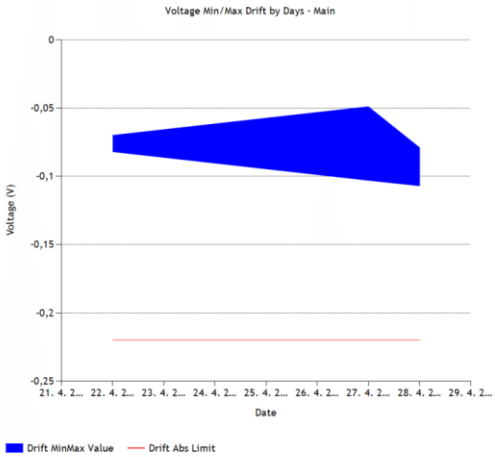
Module name: DC_W222-205-166_LFX__R **Date from:** 22. 4. 2015 12:47:10 **Date to:** 28. 4. 2015 14:08:57 **Modules count:** 26 **Status:** OK - 4 , NOK - 22

Voltage Drift - Main	Temperature Drift - NTC1
MAX: -0,049V	**MAX:** 3,300°C
MIN: -0,107V	**MIN:** 1,700°C
Median: -0,077V	**Median:** 3,100°C









Page: 1 of 2

DC_W222-205-166_LFX__R