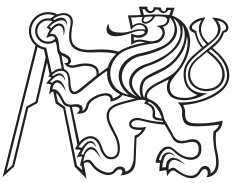
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídicí techniky

# Implementace služby pro zápis a čtení auditních záznamů

Martin Zázvorka
Studijní program: Kybernetika a robotika, obor: Systémy a řízení

2018
Vedoucí práce: Ing. Jan Šulc

# Poděkování / Prohlášení

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Janu Šulcovi za odborné vedení, za pomoc a rady při zpracování této práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne Datum 24.5.2018

.....

# Abstrakt / Abstract

Tato bakalářská práce se zabývá kompletním návrhem meziplatformové aplikace umožňující ukládání, synchronizaci a načítání auditních záznamů.

**Klíčová slova:** Auditní záznamy, .NetCore, Systémy na ukládání auditních dat

This bachelor thesis focuses on from scratch design to implementation of cross-platform aplication used for storing and synchronization of audit data.

**Keywords:** Audit Records, .NetCore, Systems for storing audit records

**Title translation:** Implementation of service for writting and reading audit records

# Obsah

# Tabulky / Obrázky

# Kapitola 1
# Úvod

Tato práce se zabývá kompletním návrhem a implementací meziplatformové aplikace sloužící k uchovávání záznamů z provozu průmyslových technologií. Navržená aplikace bude integrována do již existujícího systému firmy Energocentrum Plus. V první části této práce jsou nastíněny základní principy funkce stávajícího systému společně s aspekty, které by měla navrhovaná aplikace vylepšit. V následující kapitole 2 je podrobně popsán stávající systém, do kterého bude aplikace integrována. V této kapitole jsou specifikované všechny požadavky na vyvíjenou aplikaci. V sekcích 2.5 a 2.6 jsou představeny použité technologie včetně jejich výhod a důvodů, proč byly vybrány. V kapitole 3 je podrobně popsána konkrétní implementace celého projektu pomocí technologie .NetCore 2.0 od firmy Microsoft, relační databáze SQLite a nerelační databáze MongoDB. V kapitole 4 je ukázáno, jak byl celý projekt testován a poté jsou zde rozebrané některé jednotkové a integrační testy. Poslední sekce 4.3 je věnována různým druhům indexování databáze MongoDB.

## 1.1 Motivace

Bakalářská práce vznikla ve spolupráci s firmou Energocentrum Plus, která se zabývá projektováním, TZB (Technické zařízení budov) a měřením a regulací. V mnohých systémech je nutné uchovávat přesné záznamy provozních veličin jako je tlak, teplota, otevření ventilů nebo signalizace alarmů. Překročení prahových hodnot těchto veličin může vést k rizikovým scénářům. Mezi tyto scénáře se řadí například odstavení kotelny nebo výpadek systému vytápění v budově. Je žádoucí ukládat podrobné záznamy o tom, co který uživatel od systému žádal či co mu bylo zobrazeno. V případě poruchy je potom zřejmé, zda uživatel manuálně potvrdil varování bez adekvátní reakce. Pomocí těchto záznamů je možné zpětně určit, zda byla nehoda způsobena technickou chybou či pochybením ze strany uživatele. Tato data jsou nezbytná pro určení osoby zodpovědné za vzniklou situaci.

Ukládání záznamů se značně komplikuje s přibývajícím množstvím řízených objektů. Řízené objekty je z výpočetních důvodů nutné rozdělit na více skupin, které jsou přiděleny více řídicím serverům. Záznamy o projektech jsou uchovávány na jednotlivých SCADA (Supervisory Control And Data Acquisition - dispečerské řízení a sběr dat) serverech, což značně komplikuje přístup k datům. Vytvoření aplikace, která zajistí centrální ukládání dat ze všech SCADA serverů, umožní lepší přístup k datům. Pomocí vhodně zvolené databáze, která bude součástí auditovací komponenty, bude přístup k záznamům projektů rychlejší než v dosavadním systému. Díky této aplikaci nebude nutné, aby klient dotazující se na data konkrétního projektu věděl, kterému SCADA serveru je daný projekt přidělen. Tato skutečnost umožňuje zefektivnění práce řídicích serverů v mnoha ohledech:

- **Dynamické přerozdělování objektů mezi řídicí servery**
  Pokud máme centrálně ukládaná data nebo řídicí prvek, který přesměrovává dotazy, můžeme jednotlivé objekty přeřazovat z jednoho serveru na druhý. Náročnost

řízení jednotlivých objektů i výkon jednotlivých serverů se může lišit, a proto je výhodné využít dynamického rozdělování podle aktuálního vytížení jednotlivých serverů. Tento přístup vede k rovnoměrné zátěži serverů a umožňuje v případě výpadku některého serveru přesunout jím řízené objekty na servery ostatní.

- **Centralizované ukládání dat**
  Celý systém pracuje na základě jednotlivých komponent, které se starají o různé funkce celku. Tento přístup je preferován v aspektově orientovaném programování(AOP). Hlavní výhodou je větší modularita celého systému. Další výhoda je specializace jednotlivých komponent, která vede k většímu výkonu, neboť daná komponenta se zabývá pouze jednou činností. V tomto případě by SCADA server nebyl zatěžován dotazy klienta, protože by pouze odevzdal svá data a jakékoliv další zpracování by bylo provedeno jinde.

- **Bezpečnost**
  Izolovaná komponenta, která bude klientům umožňovat přístup k datům, výrazně zlepší bezpečnost celého systému. Klient se v případě dotazu bude schopný dostat pouze k datům uloženým na konkrétní komponentě, která data ukládá. To znamená, že klient nebude schopen komunikovat přímo s řídicími servery, což je z bezpečnostního hlediska žádoucí.

## 1.2 Související práce

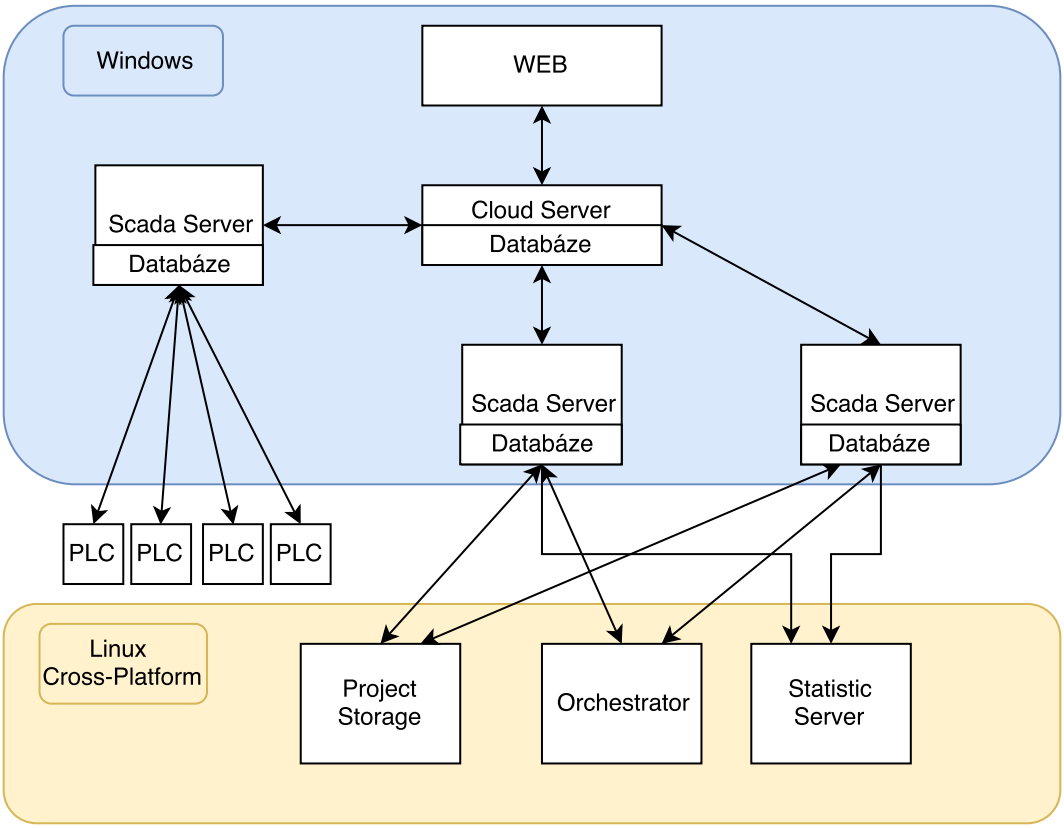
Rád bych uvedl několik jak českých, tak zahraničních prací, které se zaměřují na technologie využité v této práci. Práce Miroslava Čermáka[1] se zabývá nerelačními databázemi a výhodami databáze MongoDB, která je využita jako hlavní databáze v tomto projektu. Práce Lukáše Matouška[2] se zaměřuje na jednotkové testování pomocí testů xUnit, které jsou později v projektu využity k ověření základní funkcionality. Poslední prací, na kterou bych chtěl odkázat, je práce od dvojice Friedrich Steimann a Philip Mayer [3], která částečně ovlivnila způsob, jakým byl celý projekt navržen a podrobně se zabývá některými koncepty využití rozhraní, které jsou v práci využity.

# Kapitola 2
# Struktura a cíl práce

Hlavním cílem této práce je navrhnout stabilní a spolehlivý systém, který bude umožňovat ukládání a načítání dat. Navržený systém musí být kompaktní, dobře udržitelný a otevřený změnám částí implementace. Z těchto důvodů bude celá aplikace navrhována s ohledem na možné budoucí změny. Nejpravděpodobnějším a největším zásahem do chodu aplikace by mohl být přechod na jinou databázi, popřípadě různé optimalizace výpočetního jádra aplikace. Pro dosažení těchto cílů bude využito principů AOP. Aplikace bude rozdělena na jednotlivé komponenty, které budou využívat vhodná rozhraní. Díky této skutečnosti bude možné jednotlivé komponenty snadno nahrazovat, popřípadě měnit implementace jednotlivých funkcí bez nutnosti zasahovat do zdrojových souborů ostatních komponent. Veškeré další požadavky vycházejí z nároků zadávající firmy Energocentrum Plus a jsou podrobně popsány v sekci 2.3.

## 2.1 Popis stávajícího systému

Schéma stávajícího systému je zobrazeno na obrázku 2.1.




**Obrázek 2.1.** Schéma aktuálního systému rozděleného podle využívaných operačních systémů.

Nejprve budou popsány jednotlivé komponenty a poté funkcionalita celého systému. V popisu jednotlivých komponent je využito názvů, které odpovídají názvu projektů či zdrojových souborů.

- **Webové rozhraní**
  Veřejná část systému, přes kterou se uživatelé mohou podle nastavených práv dotazovat na data jednotlivých projektů. Přes toto rozhraní si také mohou zobrazit auditní záznamy. Výpis těchto záznamů v aplikaci Mervis [1] je na obrázku 2.2.
- **Scada Server**
  Scada server je řídicí prvek zodpovědný za monitorování a řízení projektů. Příkladem může být sledování spotřeby elektřiny v budovách, řízení průmyslových procesů či ovládání vzduchotechniky. Stávající systém využívá takovýchto serverů více a je navržen tak, aby bylo možné tento počet přizpůsobovat výkonnostním požadavkům spravovaných projektů.
- **Databáze**
  Ve firmě Energocentrum Plus je využíváno mnoho databází pro různé účely. Mezi využívané databáze patří například:
  - SQLite,
  - MongoDB,
  - MariaDB.

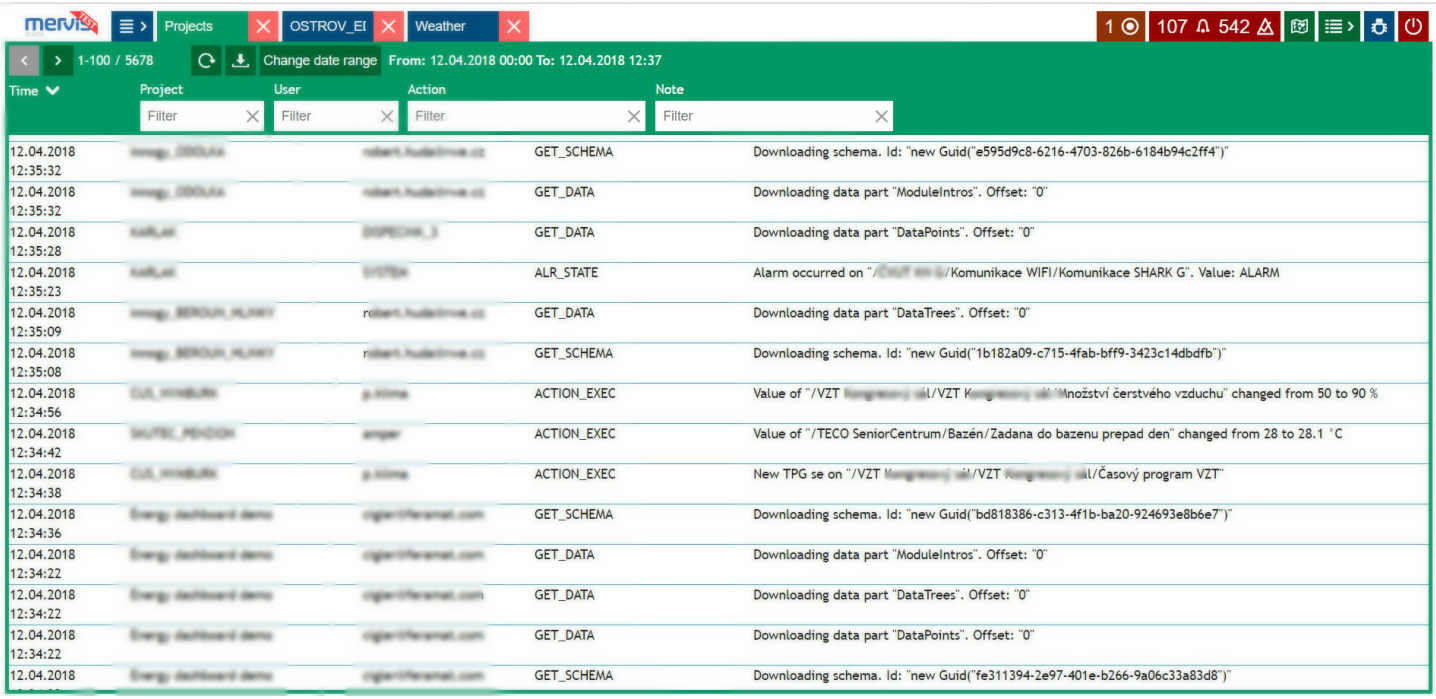
  Pro potřeby auditování jsou využity databáze SQLite [2], které jsou součástí všech SCADA serverů.
- **PLC**
  PLC je programovatelný logický automat schopný řízení akčních členů a komunikace se senzory.
- **Project Storage**
  Centrální prvek systému, který obsahuje soubory různých projektů, které nepotřebují být nutně přímo na řídicích serverech.
- **Orchestrator**
  Centrální řídicí prvek, který je zodpovědný za rozdělení projektů mezi jednotlivé řídicí servery. V tuto chvíli je využito rozdělení statické, a v případě dotazu na konkrétní projekt je předem určeno, kterému řídicímu serveru je přiřazen. Každý projekt ukládá auditní záznamy lokálně do SQLite databáze daného serveru. Po spuštění auditního serveru se počítá s možností dynamického přerozdělování projektů. Dotazování na data jednotlivých projektů je v případě dynamického rozdělování bez možnosti centrálního ukládání dat velmi složité, a proto bude implementováno až po nasazení a otestování auditního serveru.
- **Statistic Server**
  Server sloužící k diagnostice ostatních subsystémů a výpočtu statistických dat.

### 2.1.1 Existující záznamy

Při přechodu na centrální způsob ukládání dat bude potřeba přesunout všechny již existující záznamy. Počet těchto záznamů se pohybuje v řádu milionů. Vzhledem

[1] https://scada.mervis.info/
[2] https://www.sqlite.org/index.html

k využití relačních SQLite databází mají tyto záznamy jasně danou strukturu s kterou bude potřeba počítat i v případě využití nerelační databáze. Na obrázku 2.2 je zobrazena část výpisu auditních záznamů ze dne 12.04.2018. Citlivé informace o projektech či uživatelích jsou zde rozostřeny. V posledních dvou sloupcích je vypsán druh auditovaných událostí společně s podrobnostmi těchto událostí. Na obrázku 2.2 jsou události typu ACTION_EXEC, která reprezentuje provedení určité akce či nastavení parametru. Tyto dvě události reprezentují povel pro vzduchotechniku pro zvýšení množství čerstvého vzduchu a zvednutí žádané teploty v bazénu. Další typy auditovaných událostí jsou dotaz na data, dotaz na schéma projektu či upozornění na aktivaci alarmu. Stávající typy auditovacích událostí se mohou změnit, ale vzhledem k využití SQLite databáze struktura zůstane vždy stejná.




**Obrázek 2.2.** Ukázka výpisu událostí, které jsou auditovány. Obrázek je z aplikace Mervis firmy Energocentrum Plus.

## 2.2 Auditní server

V této sekci bude vysvětleno, jaké vlastnosti auditování budou v tomto projektu dodrženy. Auditní server by měl zajišťovat několik základních úkonů:

- **Zaznamenávat klíčové události**
  Navržená aplikace bude umožňovat ostatním komponentám ukládat záznamy. V případě výpadku spojení budou záznamy uchovány na straně klienta. Po obnovení komunikace bude umožněno tato data zpětně synchronizovat, čímž se zamezí jejich ztrátě.
- **Zaznamenávat činnosti jednotlivých uživatelů**,
  Tento bod bude v navržené aplikaci plně podporován, záznamy jako takové ovšem musí dodat ostatní komponenty systému.
- **Zaznamenávat možná výkonnostní rizika**
  Auditní server bude umožňovat ukládat záznamy o výkonnostních problémech jednotlivých součástí systému. Zdrojem záznamů budou jednotlivé SCADA servery.

Ukládat záznamy tohoto charakteru bude také statistický server, který detekuje hrozící přetížení SCADA serverů či jiné problémy.

- **Zajistit neměnnost záznamů**
  Tento požadavek jako jediný nebude zcela striktně dodržen. Bude zaručena neměnnost záznamů klientem nebo jinou komponentou systému. Této vlastnosti bude dosaženo pomocí absencí metod umožnující mazání, či změny záznamů ve všech veřejných rozhraních. Pro úplné zajištění neměnnosti záznamů by bylo nezbytné využít neměnných (angl. immutable) objektů a dalších bezpečnostních opatření, která nebudou implementována.
- **Zajistit dostupnost záznamů, a to i v případě výpadku zbylých částí systému**
  Této vlastnosti u navržené aplikace bude dosaženo pomocí těchto kroků:
  - Podle principů AOP bude aplikace auditního serveru navržena jako samostatná nezávislá komponenta a poběží na jiném serveru než ostatní části systému.
  - Bude zvolena databáze, která umožňuje replikaci dat.
  - Replikace celého systému na více nezávislých serverech. Tato možnost nebude implementována. Zmenšení rizika výpadku touto metodou je pro konkrétní projekt příliš nákladné.

## 2.3 Požadavky na SW aplikaci

Hlavními požadavky na SW aplikaci jsou:

- Vytvořit aplikaci a všechny knihovny multiplatformní za pomoci technologie .NetCore 2.0 pro aplikaci a .NetStandard 2.0 pro knihovny. Tento požadavek umožní výslednou aplikaci spouštět na různých operačních systémech.
- Vytvořit konzolovou aplikaci pro převedení stávajících záznamů z SQL databází do databáze nové. Tato aplikace musí být spustitelná a plně nastavitelná pomocí parametrů příkazového řádku. Aplikace dále musí generovat informace o výsledcích synchronizace. Výsledky musí obsahovat počty zkopírovaných záznamů a vynechaných duplicitních záznamů, a to jak souhrnně, tak podle dnů.
- Využití asynchronního principu pro volání hlavních metod.
- Navrhnout vhodná rozhraní, která umožní odstínění konkrétních implementací a usnadní případné změny.
- Navrhnout vhodné rozhraní pro databázi v klientské části celé aplikace, která usnadní napojení ostatních aplikací s již existující jinou databázi.
- Využít dependency injection [1].
- Využít AOP.
- Vytvořit jednotkové testy.
- Vytvořit integrační testy.
- Umožnit odesílání statistických informací na již existující server.
- Využití existujícího logování činnosti do souboru.

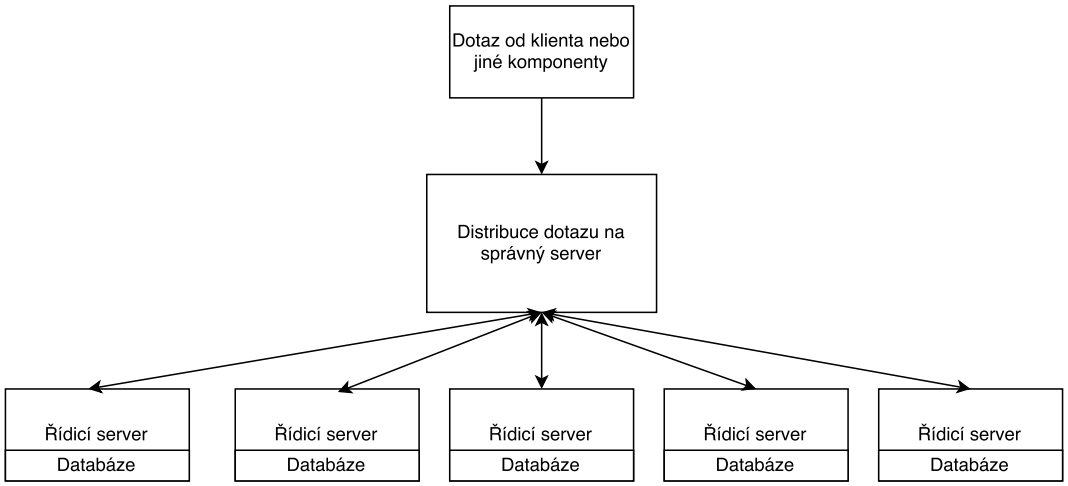
[1] Technika vkládání závislosti mezi komponenty využívaná v objektovém programování. Umožňuje jedné komponentě využívat komponentu druhou bez toho, aby na ni měla referenci při sestavování.

## 2.4 Struktura

Pokud systém obsahuje více řídicích serverů, které nezávisle řídí velké množství systémů, lze způsob ukládání dat rozdělit do tří kategorií:

- **Distribuovaný**

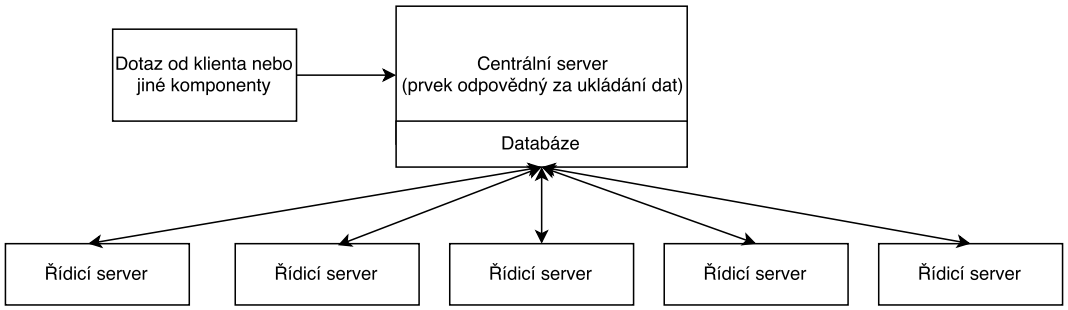
  Při využití distribuovaného způsobu ukládání dat si každý řídicí server udržuje vlastní záznamy. V případě zobrazení konkrétních záznamů je třeba vědět, na kterém serveru se daná data nachází. Hlavní výhodou je ukládání dat přímo na řídicím serveru, tedy absence požadavku na stabilitu spojení či komunikace s jinými částmi systému. Nevýhodou je nutnost vědět, na kterém serveru jsou která data. Další nevýhodou je vliv dotazů na data ostatních částí systému na výkon řídicího serveru.




**Obrázek 2.3.** Ukázka distribuovaného systému řízení.

- **Centralizovaný**

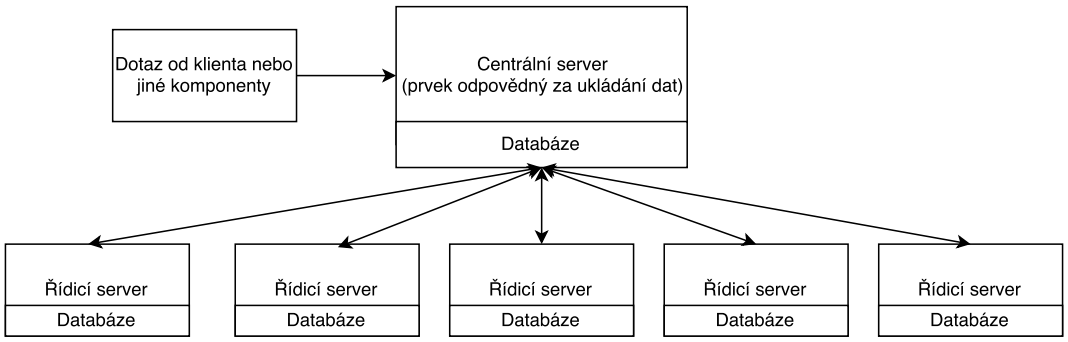
  V případě centralizovaného ukládání dat se v systému nachází subsystém, který umožňuje ostatním subsystémům ukládat záznamy. Hlavní výhodou je centralizace veškerých záznamů a odstranění vlivu dotazů na výkon jednotlivých řídicích serverů. Řídicí server pouze odevzdá svoje záznamy a veškerá další práce nad těmito daty ho nijak výkonnostně nezatěžuje. Nevýhodou je nutnost celý tento subsystém realizovat. Pro funkčnost tohoto typu ukládání je potřeba zajistit spolehlivost a stabilitu komunikace s centrální komponentou.




**Obrázek 2.4.** Ukázka centralizovaného systému řízení.

- **Hybridní**

  Tento druh ukládání dat kombinuje výhody a nevýhody obou výše zmíněných způsobů. Stejně jako v případě centralizovaného ukládání dat existuje samostatný subsystém, kterému všechny řídicí servery posílají svá data. Rozdílem je však řešení úložišť na jednotlivých řídicích serverech. Ty totiž stále mají možnost ukládat data lokálně. Toto řešení umožňuje záchranu dat v případě výpadku auditního serveru, a následnou synchronizaci v době, kdy se spojení obnoví. Nevýhodou je komplexnost tohoto řešení. Je potřeba centrální prvek a databáze na každém řídicím serveru. Dále je potřeba, aby byla implementována synchronizace záznamů mezi centrálním prvkem a ostatními databázemi. Výhodou je větší spolehlivost.




**Obrázek 2.5.** Ukázka hybridního systému řízení.

Z důvodu rostoucího počtu serverů je potřeba stávající řešení, tedy distribuované ukládání dat, změnit. Přechod na jinou strukturu a způsob ukládání dat umožní zefektivnit chod celého systému a dosáhnout větší nezávislosti jednotlivých subsystémů. Navrhovaný systém bude využívat hybridní způsob ukládání dat. Tato možnost usnadní potencionální růst celého systému a zajistí spolehlivější přístup k datům.

## 2.5 Výběr technologie

Výběr technologie na rozdíl od výběru databáze (sekce 2.6) byl jasně definován požadavky firmy Energocentrum Plus. Z důvodu integrace tohoto projektu do již existujícího a poměrně velkého systému. Z hlediska integrity celého systému je žádoucí, aby byl tento projekt napsán v technologii .NetCore od firmy Microsoft. Důvody pro volbu této technologie v rámci celého systému jsou následující:

- **Multiplatformní software (angl. cross-platform)**

  Technologie .NetCore je multiplatformní, tedy stejně jako například Java může být provozována na více platformách, například Windows, Linux nebo Mac OS. Výhodou je možnost nasazení na serverech běžících na operačním systému Linux. Tato možnost je velmi výhodná, protože pronájem Linux serverů je dlouhodobě cenově výhodnější. Nevýhodou je potřeba mít na cílovém zařízení nainstalován runtime (knihovna nutná ke spuštění programu), který kód přeloží do instrukcí dané platformy. Tato knihovna může vyžadovat místo na disku v řádu stovek MB a v případě nasazení velmi malého počtu projektů je výhodné tuto knihovnu částečně obejít pomocí kompilace na cílovou platformu.

- **Kompabilita**
  .NetCore je plně kompatibilní s knihovnami .NetStandard (použítá verze .NetCore 2.0 je kompatibilní s knihovnami .NetStandard 2.0), což umožňuje propojení technologií .NetCore s .Net framework, Xamarin[1] a Mono[2].

- **Podpora**
  Tato technologie je aktivně vyvíjena firmou Microsoft. Nejnovější verze 2.0 vyšla 14. srpna 2017. Tato technologie je prezentována jako jedna z hlavních zájmů firmy Microsoft, a proto se nepředpokládá, že by v blízké době přestala být podporovaná. Oficiální dokumentace je dostupná online [5].

- **Licence**
  .NetCore je otevřený software a je možné ho bezplatně využívat i pro komerční účely.

## 2.6 Výběr databází

V případě ukládání dat je zcela stěžejní výběr databáze. Databáze se dají rozdělit na dvě základní kategorie. Relační (SQL) a nerelační (NoSQL) databáze [13]. Pro tento projekt je potřeba vybrat dvě databáze, hlavní databázi, ve které budou uchovány veškeré záznamy, a sekundární databázi pro klientskou část, která bude využita pouze v případě výpadku spojení pro dočasné uchování záznamů.

### 2.6.1 Relační Databáze

Relační databáze ukládá data do struktur ve formě tabulek. Záznamy jsou reprezentovány řádky a jednotlivé atributy jsou reprezentovány sloupci. Základní vlastností relačních tabulek je unikátnost jednotlivých záznamů. Tato vlastnost je zaručena pomocí unikátního klíče, který je většinou označen Id (identification) a je povinný. Protože jsou záznamy reprezentovány jako řádky v tabulce, musí všechny záznamy obsahovat stejné atributy. Výhodou takto exaktně definované struktury dat je velmi rychlé vyhledávání jednotlivých atributů. Mezi nevýhody patří téměř nulová flexibilita při změně struktury ukládaných dat. Mezi nejvyužívanější relační databáze patří k dubnu 2018 [11]:

- Oracle,
- MySQL,
- Microsoft SQL Server,
- PostgreSQL,
- DB2,
- Microsoft Access,
- SQLite.

### 2.6.2 Nerelační databáze

Nerelační databáze nejsou přesně definovány. Označují se NoSQL (Not only SQL) a zahrnují obecně veškeré databáze, které nejsou založeny na principu relačních databází. NoSQL databáze lze rozdělit na 5 základních druhů: [9]

[1] https://www.xamarin.com/
[2] https://www.mono-project.com/

- **Key - Value databáze**,
  Tento druh databáze slouží k uchovávání takzvaných `Key - Value` dvojic. Tato dvojice může například reprezentovat uživatele a všechny jeho objednávky.
- **Sloupcově orientované databáze**,
  Tento druh NoSQL databází je koncepčně nejblíže relačním databázím. Jednotlivé atributy jsou ukládány do sloupců stejně jako v případě SQL databáze, ale nejsou ukládány ve formě tabulek, nýbrž ve formě distribuovaných struktur. Sloupcově orientované databáze vynikají velmi dobrou škálovatelností. Nejvíce používané databáze tohoto typu jsou databáze Big Table vyvíjená společností Google a databáze Cassandra od firmy Apache Software Foundations.
- **Grafové databáze**,
  Grafové databáze ukládají data ve formě grafu obsahujícího uzly a hrany. Uzly reprezentují jednotlivé záznamy a hrany potom vztahy mezi jednotlivými uzly. Prohledávání grafu je možno provádět velmi rychle a proto jsou tyto databáze rychlejší než databáze relační. Srovnání rychlost obou druhá databází je zmíněno v článku [9] a podrobněji rozebráno v článku [10]. Nevýhodou těchto databází je složitá škálovatelnost.
- **Objektově orientované databáze**,
  Objektově orientovaná databáze ukládá veškerá data ve formě objektů a využívá objektově orientovaného programování. Objekty uložené v databázi využívají vlastností jako dědičnost či polymorfismus. Nevýhodou tohoto typu databáze je silná vazba na konkrétní programovací jazyk.
- **Dokumentově orientované databáze**.
  V případě dokumentově orientovaných databází jsou veškerá data ukládána ve formě dokumentů. Hlavní předností tohoto druhu databází je velká flexibilita, které je dosaženo absencí schématu ukládaných záznamů. Každý jednotlivý dokument je jednoznačně identifikován, ale není předem pevně určeno, jakou bude mít strukturu. Dokumentově orientované databáze se velmi snadno horizontálně škálují, protože na rozdíl od grafových databází jednotlivé dokumenty ve většině případů nemají mezi sebou pevné vazby. Z této vlastnosti je patrné, že není zcela vhodné používat tento druh databáze na ukládání dat, která na sebe odkazují.

### 2.6.3 Vybrané databáze

Pro vyvíjenou aplikaci je potřeba vybrat databázi pro centrální ukládání dat a dále databázi, která bude využita v klientské části aplikace pro případné ukládání záznamů pro pozdější synchronizaci. Z důvodu jednoduchosti relačních databází a jejich podpory v .NetCore bude SQL databáze využita pro klientskou část. Po konzultaci se zástupci firmy Energocentrum Plus byla pro tento projekt vybrána databáze SQLite, která je již použita ve stávajícím systému. Vlastnosti díky kterým je tato databáze vhodná pro navrhovanou aplikaci jsou následující [12]:

- Databáze SQLite umožňuje zapisovat přímo do souborů na disku.
- SQList je ucelená aplikace a neobsahuje externí reference.
- Veřejně dostupný buglist, ve kterém jsou zaznamenány nahlášené chyby systému a případné změny kódu.
- Spolehlivost. Každá nová verze je podrobně testována.

Vzhledem k velkému počtu a možným změnám struktury ukládaných záznamů bude pro centrální ukládání dat využita databáze typu NoSQL. Nejpoužívanější NoSQL databází je k listopadu 2017 MongoDB [11]. Po konzultaci bylo MongoDB vybráno jako hlavní databáze a budou nastíněny jeho základní vlastnosti a výhody. Zdrojem je oficiální dokumentace [8].

- **Způsob ukládání jednotlivých záznamů**
  Databáze MongoDB ukládá jednotlivé záznamy ve formátu BSON. Tyto dokumenty jsou velice podobné JSON objektům (Javascript Notation). MongoDB využívá agregátní datový model, takže všechna data příslušící k jednomu záznamu jsou obvykle obsažená v jednom dokumentu. Dokumenty v databázi na rozdíl od SQL nemají pevně danou strukturu a mohou mít zcela rozdílné položky nebo velikost. Struktura záznamů se může změnit bez nutnosti předělání celé databáze. Tato vlastnost je stěžejní, neboť struktura ukládaných dat není přesně daná a pravděpodobně se bude dodatečně měnit.
- **Možnosti dotazování**
  Vzhledem ke struktuře dokumentů umožňuje MongoDB dotazování na základě primárního klíče nebo kompozitního klíče. Primární klíč pouze označuje konkrétní dokument, zatímco kompozitní klíč využívá struktury daného dokumentu. Kompozitní klíč je vytvořen kombinací atributů uloženého dokumentu. MongoDB také nabízí pro technologii .NetCore balíček MongoDB.Driver.Core, který ve verzi 2.4.4 podporuje .NetCore 2.0.
- **Rychlost**
  MongoDB je velice náročné na operační paměť RAM. Tento požadavek je ovšem kompenzovaný rychlostí, která se projevuje hlavně při zpracování velkého množství dat. V článku [6] je ukázáno, že MongoDB je v případě vkládání nezávislých dat až stokrát rychlejší než SQL Server 2008 a skoro třikrát rychlejší v případě základních dotazů. V případě komplexnějších dotazů bylo MongoDB opět mnohonásobně rychlejší. Podobných výsledků bylo dosaženo i v tomto článku [7].
- **Spolehlivost**
  MongoDB podporuje sharding[1] a také replikaci[2], což vede k větší spolehlivosti.
- **Dokumentace**
  Oficiální dokumentace[8] je volně dostupná na internetu a je velmi podrobná.
- **Zkušenosti**
  Ve firmě Energocentrum Plus byla již tato databáze využita v jiných projektech.
- **Licence**
  MongoDB je otevřený software a je možné ho bezplatně využívat i pro komerční účely.

## 2.7 Ukládaná data

Záznam bude reprezentovaný pomocí objektu, který bude veškerá data obsahovat jako své proměnné. Do databáze bude objekt uložen ve formě BSON dokumentu. Tento způsob ukládání je výhodný, protože umožňuje snadné převedení dokumentu zpět na objekt. Ukládaný záznam má strukturu uvedenou v tabulce 2.1.

[1] Rozdělení dat mezi více instancí databáze MongoDB podle předem definovaných parametrů.
[2] Více instancí databáze MongoDB obsahuje identická data, v případě výpadku nedojde ke ztrátě záznamů.

| Datový typ | Jméno | Popis |
|---|---|---|
| String | SourceType | Odkud přišla data (např. řídicí server) |
| String | SourceId | Id zdroje dat |
| Guid? | ProjectId | Id projektu, kterého se data týkají |
| String | Category | Kategorie dat (např. alarm) |
| String | FormattingStr | Informace o formátování |
| String[] | Parameters | Parametry |
| DateTime | Timestamp | Čas ve formátu UTC |
| String | User | Uživatel |

**Tabulka 2.1.** Struktura záznamu, který bude při ukládání modifikován pro potřeby databáze.

## 2.8 Navrhované řešení

Vzhledem k požadavku na využití AOP je aplikaci rozdělena na pět komponent. Komponenty implementované jako samostatné projekty jsou pojmenovány anglicky a uvedené názvy odpovídají poslední části názvů jednotlivých knihoven či aplikací. Komponenty reprezentující databáze jsou pojmenovány česky, protože jsou součástí jiného projektu. Hlavní databáze je součástí projektu Engine a klientská databáze je součástí projektu Client.

### 2.8.1 Kompenty navrhované aplikace

- **Klientská databáze**
  Tato komponenta zajišťuje dočasné ukládání záznamů v případě výpadku spojení. Na rozdíl od komponenty Hlavní databáze umožňuje také mazání uložených záznamů. Tuto funkci je žádoucí využívat striktně pouze po úspěšné synchronizaci. Nedodržením této podmínky dojde ke ztrátě záznamů.

- **Client**
  Zajišťuje navázání komunikace s komponentou Host. V případě výpadku spojení využívá dočasně lokální klientské databáze a po obnovení spojení odešle komponentě Host celý obsah klientské databáze s požadavkem na synchronizaci. Po potvrzení úspěšné synchronizace vymaže synchronizované záznamy z klientské databáze a další záznamy odesílá opět přímo komponentě Host.
  Tato komponenta musí nabízet funkcionalitu využívanou již existujícím řešením. Nabízené metody musí umožnit:

  - Uložit list záznamů,
  - Načíst historii záznamů určitého projektu.

- **Host**
  Umožní spuštění služby na dané adrese a portu. Přijímá požadavky od jednotlivých klientů. Tato komponenta komunikuje se statistickým serverem a vytváří logovací záznamy.

- **Engine**
  Hlavní část aplikace, zpracovává požadavky od komponenty Host a plní je pomocí funkcí hlavní databáze. Komunikace s komponentou Host probíhá pomocí internetového protokolu HTTP, komunikace s komponentou Hlavní databáze protokolu HTTP nevyužívá.

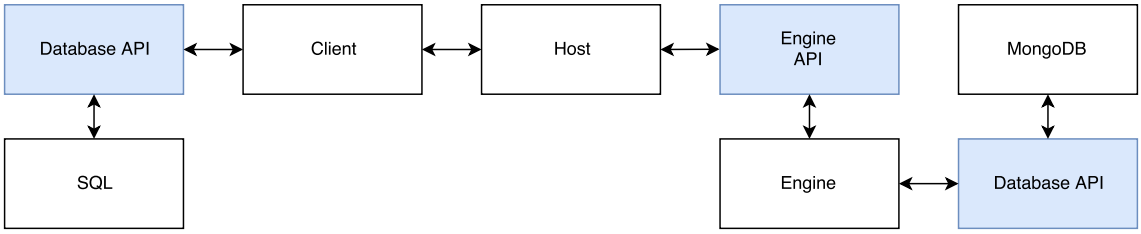
- **Hlavní databáze**
  Tato komponenta obsahuje následující funkce:
  - uložení listu záznamů,
  - synchronizace listu záznamů,
  - načtení historie záznamů určitého projektu.

Grafické znázornění rozdělení aplikace do jednotlivých komponent je zobrazeno na obrázku 2.6.


**Obrázek 2.6.** Zjednodušené schéma navrhované aplikace zobrazující jednotlivé komponenty.

### 2.8.2 Rozhraní navrhované aplikace

Jedním z požadavků na vyvíjenou aplikaci je vytvoření vhodných rozhraní pro odstínění konkrétní implementace. V tomto případě je žádoucí odstínit všechny databáze. Rozhraní pro hlavní databázi umožní snadnou změnu implementace v případě přechodu na jiný druh databáze. Rozhraní pro klientskou databázi zjednoduší napojení ostatních aplikací, které již mají implementovány své vlastní databáze. Poslední rozhraní bude pro výpočetní jádro aplikace Engine. Toto rozhraní umožní případné změny implementace bez nutnosti zásahu do ostatních komponent. Schéma výsledného návrhu s využitím API je na obrázku 2.7.




**Obrázek 2.7.** Schéma navrhované aplikace obsahující všechna potřebná rozhraní

Názvy jednotlivých rozhraní jsou zde uvedena včetně názvů využitých při implementaci a budou obsahovat následující metody:

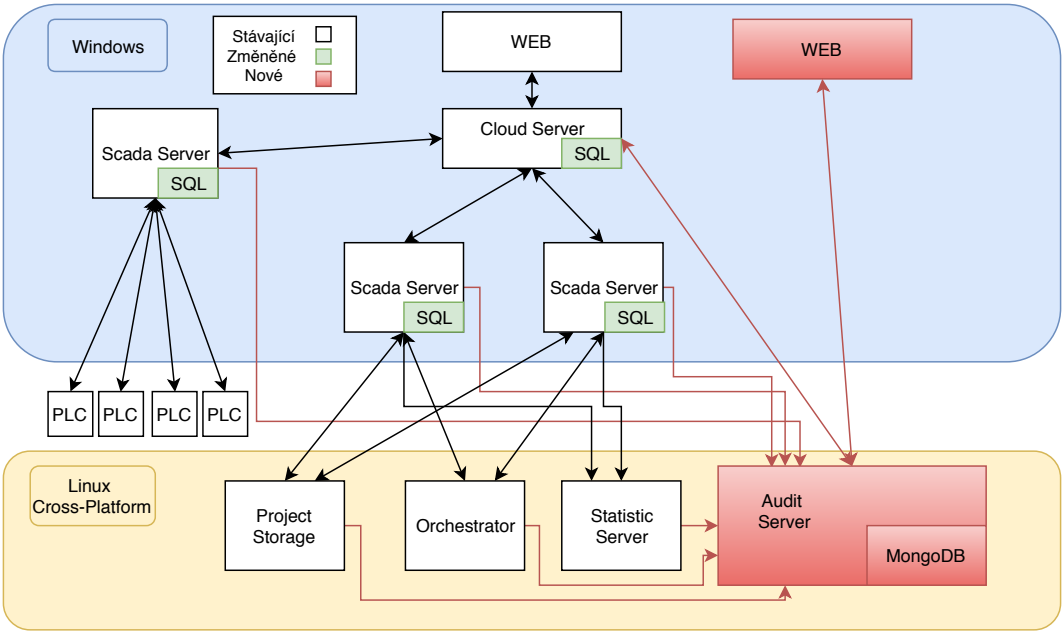
- **API výpočetního jádra - IAudit**
  Všechny metody využívají protokolu HTTP.
  - PingAsync
    Pomocí metody PingAsync lze ověřit spojení s touto komponentou. Díky proměnné typu bool? je možné ověřit stav databáze.
  - AddNewEvents
    Uloží list záznamů do databáze. Obsah listu není kontrolován, případné duplicitní záznamy jsou uloženy.
  - GetEventHistory
    Tato metoda vrátí historii daného projektu. Je možné nastavit časový interval, ve kterém se záznamy nacházejí, popřípadě maximální počet položek historie, které budou zobrazeny.

- SynchronizeEvents
  Tato metoda slouží k synchronizaci listu záznamů, na rozdíl od metody AddNewEvents je celý list kontrolován, a do databáze jsou uloženy pouze záznamy, které databáze neobsahuje.

- **API Hlavní databáze - IStorage**
  API je identické k API Engine. Vstupní a výstupní objekty funkcí již nebudou využívat třídy pro HTTP komunikaci.

- **API Klientské databáze - IDatabase**
  - AddNewEvents
    Tato metoda uloží list záznamů, obsah listu nikterak nekontroluje.
  - GetAllEvents
    Tato metoda vrátí list obsahující veškeré záznamy v databázi.
  - DeleteAllRecordsToDate
    Tato metoda umožňuje smazaní veškerých záznamů starších než zvolené datum. Tato možnost bude využívána pouze po úspěšné synchronizaci a bude mazat pouze lokální záznamy, které jsou uloženy jinde.

Na schématu 2.8 je zobrazen výsledný systém po integraci auditního serveru. Na schématu je zobrazen nový WEB, tento prvek reprezentuje nová webová rozhraní, která umožní uživateli přístup k záznamům a informacím uložených na auditním serveru. Tato rozhraní v tuto chvíli nejsou vytvořena, ale do budoucnosti se s nimi počítá a auditní server je pro ně nezbytný. Pro porovnání schéma původního systému je na obrázku 2.1.




**Obrázek 2.8.** Schéma výsledného systému s auditním serverem

# Kapitola 3
# Implementace

V této kapitole je podrobně popsána implementace všech dílčích součástí navržené aplikace. V podkapitolách jsou popsány třídy jednotlivých knihoven či aplikací.

## 3.1 ESG.Audit.Shared

Knihovna Shared obsahuje všechny základní objekty, které jsou obsažené v rozhraních pro auditování. Jako jediná je v projektu obsažena dvakrát a to jako .Net Standard 2.0 a .Net framework 4.6.1. Cílem tohoto dělení je omezení závislostí jednotlivých projektů a hlavně možnost využití čistě frameworkových klientů, kteří neumí využívat .Net Standard knihovny. Většina stěžejních metod využívá objekty, které reprezentují vstup a výstup dané metody. Příkladem je metoda AddNewEvent, který využívá třídy AddNewEventInput a AddNewEventOutput. Využití těchto tříd vede k větší přehlednosti kódu za cenu více zdrojových souborů.

- **AddNewEventInput**
  Tato třída je vstupním objektem metody AddNewEvent a obsahuje list auditních záznamů, které jsou určené k uložení.
- **AddNewEventOutput**
  Tato třída je výstupním objektem metody AddNewEvent a obsahuje proměnnou typu Bool, která je true v případě úspěšného uložení.
- **AuditConstants**
  Tato třída slouží k nastavení Url pro HTTP komunikaci mezi klientem a hostem. Z tohoto důvodu je třída statická a obsahuje všechny potřebné adresy. V případě rozšíření služby například o paralelní logovací server je možné tyto adresy využít a pouze měnit takzvaný BasePath. Výsledná dotazovaná adresa je ve tvaru - adresaServeru:port/BasePath/api/get/metoda.
- **Errorcode.cs**
  Statická třída AuditResultCodeHelper slouží k předávání chybových zpráv. Obsahuje enum AuditResultCode s následujícími možnostmi:
  - 0 - OK,
  - 1 - InvalidParameters,
  - 2 - InternalError,
  - 3 - Timeout,
  - 300 - None,
  - 301 - Storage_WrongRecordId,
  - 302 - Storage_RecordNotFound,
  - 303 - Storage_RecordAlreadyExists,
  - 304 - Storage_RecordIsAmbiguous.

  Nabízená statická metoda CodeToMessage slouží k překladu chybového kódu a může obsahovat podrobnější vysvětlení chyby.

- **EventHistory.cs**
  Tento zdrojový soubor obsahuje více tříd využívaných třídou GetHistoryOutput. Využitím více dílčích tříd je dosaženo větší přehlednosti a srozumitelnosti. Jednotlivé třídy jsou následující:

  - **EventHistoryItem**
    Tato nejmenší položka obsahuje pouze jeden auditní záznam.

  - **EventProjectHistory**
    Třída EventProjectHistory již obsahuje celou historii daného projektu, proto obsahuje list objektů typu EventHistoryItem a dále nese informaci o tom, ke kterému projektu dané záznamy patří. Tato informace nemusí být z různých důvodů vždy dostupná a proto je nesena objektem typu guid?.

  - **EventHistory**
    Poslední třída sloužící k reprezentaci historie záznamů. Obsahuje list objektů typu EventProjectHistory a pomocí tohoto objektuje možné vrátit najednou historii více samostatných projektů.

- **GetHistoryInput**
  Pomocí objektu této třídy jsou předávány veškeré informace potřebné k identifikaci části historie auditních záznamů, tato množinu záznamů je identifikována podle těchto parametrů:

  - Guid ProjectId,
  - DateTime UtcFrom,
  - DateTime UtcTo,
  - int ValueOffset,
  - int ValueCount.

- **GetHistoryOutput**
  Využití této třídy je sice na první pohled zcela nadbytečné, ale nevyužití přímo objektu typu EventHistory umožnuje v budoucnosti rozšíření předávaných parametrů bez nutnosti změn v části EventHistory, což povede k lépe funkčně rozdělenému kódu.

- **PingInput**
  Tato třída slouží k nastavení funkce PingAsync. Promněná ReadyCheck umožňuje ověřit stav databáze. Toto ověření není povinné a proto je ReadyCheck typu bool?

- **PingOutput**
  Objekt této třídy popisuje výsledek funkce PingAsync. Obsahuje DateTime s časem dotazu a odpověd typu bool? na připravenost databáze, kde null je vyplněn v případě, že byl předán null na vstupu metody.

- **SynchronizeEvents**
  Synchronizování záznamů zde probíhá pomocí dvou databází, databáze hlavní (centrální) a databáze lokální. V případě výpadků komunikace či jiných problémů se data uloží do lokální databáze. Při synchronizaci se odešle list těchto záznamů společně s časem dotazu na lokální databázi. Pokud vše proběhne v pořádku, metoda SynchronizeEvents potvrdí synchronizaci záznamů a všechny záznamy z lokální databáze do daného času mohou být smazány.

- **SynchronizeEventsInput**
  Tato třída obsahuje data pro synchronizaci: čas dotazu na lokální databázi a všechny záznamy určené k synchronizaci.
- **SynchronizeEventsOutput**
  Pomocí objektu této třídy je předáván výsledek synchronizace. Veškeré informace jsou uloženy v proměnných:
  - Bool Result
    Tato proměnná obsahuje návratovou hodnotu celé metody, kde true značí, že vše proběhlo v pořádku.
  - DateTime Time
    Díky této proměnné lze následně odmazat záznamy z lokální databáze.
  - String Log
    Do této proměnné lze vložit dodatečné informace, jako počet nově uložených záznamů a počet duplicit.

- **AuditRecord**
  Klíčová třída celé aplikace. Tato třída reprezentuje auditní záznam a obsahuje tyto proměnné:
  - DateTime Timestamp(Z důvodu možných problémů při předávání času mezi různými databázemi je čas ve formátu UTC a je ukládán s přesností na milisekundy.),
  - String SourceType,
  - String SourceId,
  - Guid? ProjectId,
  - String User,
  - String Category,
  - String FormattingStr,
  - String Parameters.

  Třída dále obsahuje metodu AddSource, která umožňuje do záznamu vyplnit proměnné SourceId a SourceType. Lokálně jsou vždy záznamy vyplňovány bez zdroje, ale centrální databáze potřebuje tuto doplňující informaci. Stěžejní je metoda GetIdentity, která vrací textový řetězec sloužící k identifikaci záznamů a případnému rozeznání duplicit.

## 3.2 ESG.NetCore.Audit.Engine

Knihovna Engine tvoří výpočetní jádro celé aplikace, obsahuje rozhraní pro hlavní databázi IStorage a rozhraní pro výpočetní část EngineApi. Daná rozhraní jsou dále implementovány s využitím databáze MongoDB. Databáze MongoDB je vzhledem k jejímu přístupu k objektům velmi snadno využitelná, ale i přesto jsou zde vytvořeny objekty typu MongoAuditRecord, které například ukládají ProjectID, která je typu Guid jako String. Tento přechod mezi auditním záznamem a objektem v databázi umožňuje snazší práci se záznamy a usnadní případné změny ve struktuře záznamů.

- **EngineApi**
  Toto rozhraní obsahuje definici všech metod výpočetní části aplikace. Všechny definované metody jsou asynchronní a využívají objektů GlobalResponse, GlobalRequest a CancellationToken.

- **IStorage**
  Rozhraní IStorage obsahuje definici všech potřebných metod databáze.Všechny definované metody jsou asynchronní, navazují na metody definované v rozhraní EngineApi, ale již nevyužívají třídy GlobalResponse a GlobalRequest.

- **AuditImpl**
  Tato třída implementuje rozhraní EngineApi a je pro lepší přehlednost rozdělená do dvou zdrojových souborů ApiImpl.cs a ApiImplMethods.cs. Třída neobsahuje žádné interní metody kromě těch obsažených v implementovaném rozhraní, většina další funkcionality je očekávána od třídy implementující rozhraní IStorage.

- **AuditException**
  Hlavním úkolem této třídy je umožnit předávání chybových zpráv do vyšších vrstev.

- **AuditExceptionHelper**
  Tato statická třída tvoří chybové a logovací zprávy.

- **MongoAuditRecord**
  MongoAuditRecord je pozměněnou reprezentací třídy AuditRecord, která je ukládána do databáze MongoDB a obsahuje proměnné uvedené v tabulce 3.1.

| Proměnná AuditRecord | Proměnná MongoAuditRecord | Název BsonElementu |
|---|---|---|
| — | ObjectId Id | BsonId |
| DateTime TimeStamp | DateTime TimeStamp | ts, kind = UTC |
| String SourceType | String SourceType | st |
| String SourceId | String SourceId | si |
| Guid ProjectId | String ProjectId | p |
| String User | String User | u |
| String Category | String Category | c |
| String FormattingStr | String FormattingStr | fs |
| String Parameters | String[ ] Parameters | fp |

**Tabulka 3.1.** Promněné třídy MongoAuditRecord společně s názvy jednotlivých BsonElementů

- **MongoMessageConverterExtensions**
  Tato statická třída umožňuje využívat metodu ConvertMessage, která umožňuje převézt AuditRecord na MongoAuditRecord a naopak. Zajišťuje veškeré potřebné přetypování proměnných a dále převádí proměnnou parameters mezi polem a jedním textovým řetězcem a to pomocí metody ParseParams.

- **MongoStorage**
  Nejobsáhlejší třída této knihovny, která implementuje rozhraní IStorage a dále nabízí další metody, které jsou využity při migrování již existujících databází. V konstruktoru probíhá prvotní nastavení databáze MongoDB, ověření dodaného připojovacího řetězce a navázání komunikace. Nakonec je nastaven kombinovaný

index, který byl vybrán na základě sekce 4.3. Je zde využito dědičnosti a proto část využívající tuto třídů pouze jako zástupce rozhraní IStorage nemůže využívat některé další metody, které jsou implementovány pro účely testování či migraci. Je nezbytné, aby k těmto metodám neměl uživatel přístup. Příkladem takové funkce je metoda DropDatabaseAsync, která nevratně vymaže všechny záznamy z databáze.

## 3.3 ESG.Audit.Host

- **MainController**

  Tato třída je potomkem abstraktní třídy Controller. Při spuštění aplikace Host jsou automaticky spuštěni všichni potomci této třídy. Třída MainController nabízí následující metody:

  - **GetPingAsync**
    Tato metoda umožňuje otestovat komunikaci s databází pomocí požadavku typu get.
  - **PostPingAsync**
    Druhá z dvojice metod na testování komunikace. Tato varianta využívá stejně jako všechny následující metody požadavek typu post.

  Další metody již přímo navazují na rozhraní EngineApi. Zajišťují kontrolu příchozích dat a následné předání požadavku na instanci třídy AuditImpl. Tyto metody jsou:

  - **GetEventHistory**,
  - **AddNewEvents**,
  - **SynchronizeEvents**.

- **Program**
  Úkolem třídy Program je spuštění a nastavení aplikace. Tato třída obsahuje hlavní metodu main. Při spuštění jsou využity výchozí hodnoty některých parametrů, jako například DefaultServerUri nebo DefaultLogFile. Zbylé parametry jsou načteny z konfiguračního souboru appsettings.json. Tento soubor je součástí projektu a je zde jako vestavěný zdroj. Nastavování parametrů ze souboru je výhodné v případě více přepínatelných konfigurací. Konfigurační soubor obsahuje následující parametry:

  - **Host**
    Tato sekce obsahuje parametr ServerUrl, který obsahuje adresu, která bude použita při nastavování aplikace.
  - **Engine**
    Parametrem StorageUri je nastavena adresa, na které běží databáze MongoDB.
  - **StatServer**
    Tato sekce konfiguračního souboru obsahuje Uri adresu na StatServer. Tento server je vyvíjen společností Energocentrum Plus a slouží k diagnostice a měření výkonu jednotlivých komponent.

- **Swagger**
  Parametr Enabled je využit při nastavení swaggeru [1]. Swagger je volný software, který lze velmi snadno využít na testování nabízených metod. S využitím znalosti návratových a vstupních objektů je swagger schopný velmi přesně navrhnout vstupní data do testované funkce, což celé testování výrazně urychluje. Jedinou proměnou, kde swagger nedokáže vyplnit validní výchozí hodnotu je proměnná ProjectId, která je typu Guid. Ukázka využití swaggeru zobrazující uživatelské prostředí společně s odpovědí ze strany aplikace je zobrazena na obrázku 3.1.

- **Logging**
  V této sekci je obsaženo kompletní nastavení logování aplikace. V této části bylo podle požadavků využito stávající logování do souboru dodané firmou Energocentrum Plus.

  - **Console**
    Parametr Disabled určuje, zda budou logovací záznamy vypsány do konzole programu.

  - **File**
    Parametr Disabled určuje, zda budou logovací záznamy zapsány do souboru. Dále je nastaveno jméno logovacího souboru včetně celé cesty a možnosti pro jednotlivé soubory. Parametrem NewFileOnStart je vynuceno vytvoření nového souboru pro každé spuštění této aplikace. Pokud je tento parametr nastaven na false, aplikace připojí nové záznamy do již existujícího souboru. Dále jsou zde nastaveny parametry pro soubory logování. Tyto parametry jsou společně s nastavenými hodnotami uvedeny v následujícím seznamu:

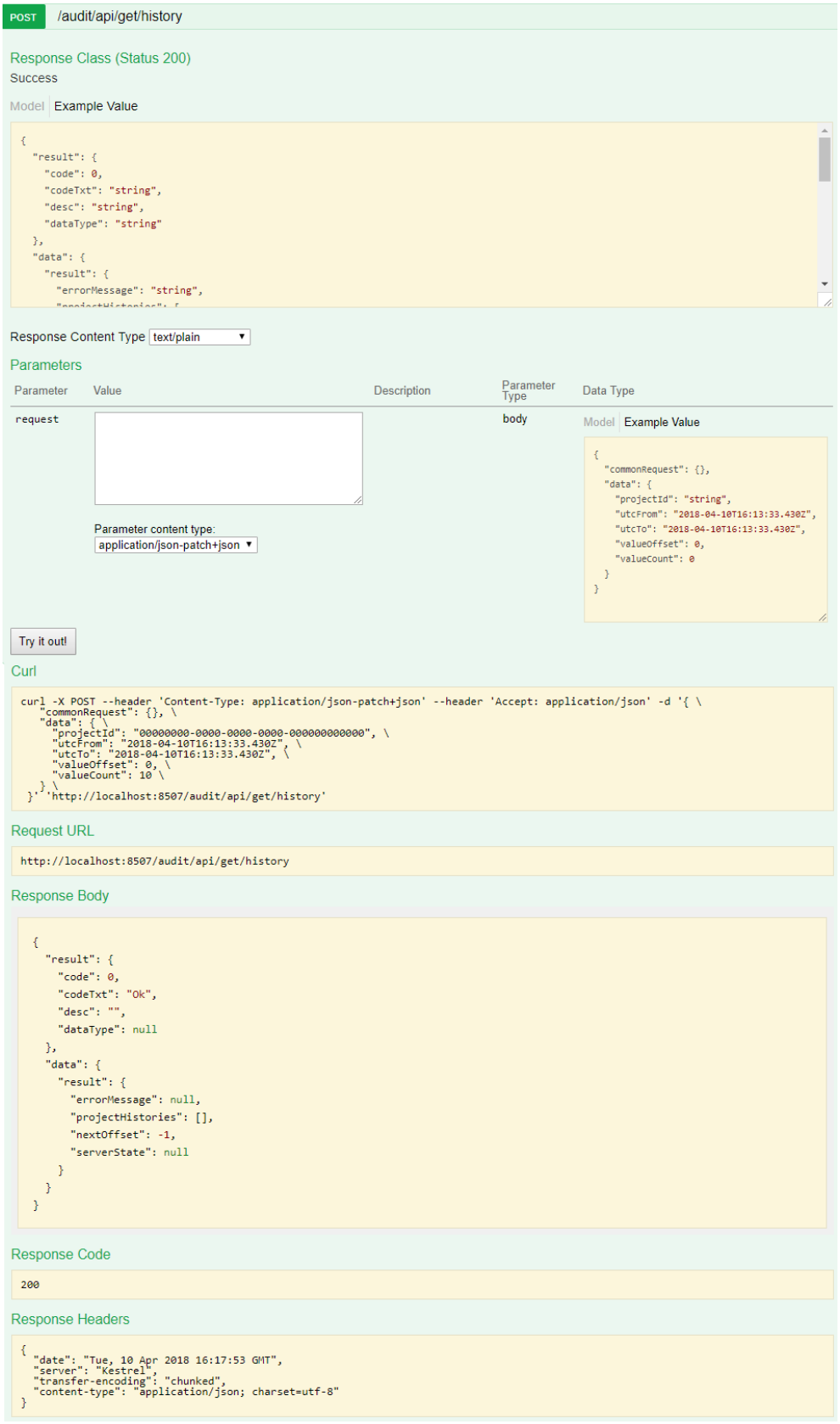
    - Interval logovacího souboru = day,
    - Maximální velikost logovacího souboru = 20 MB,
    - Celková velikost souborů = 2000 MB,
    - Maximální počet logovacích souborů = 1000.

  - **Level**
    V této sekci je nastavena výchozí úroveň logování na úrovni aplikace společně s nastavením logování na vyšších úrovních. Nastavení logovacích úrovní je následovné:

    - **Default** = Trace,
    - **System** = None,
    - **Windows** = None.

- **Startup**
  Tato třída se podílí na nastavení aplikace, její metody jsou volané třídou Program. Tato třída by mohla být součástí třídy program, ale takto je dosaženo větší přehlednosti funkcionality.

[1] `https://swagger.io/`




**Obrázek 3.1.** Ukázka uživatelského prostředí swaggeru pro funkci GetEventHistory. Část Curl obsahuje objekt GetHistoryInput ve formátu JSON. V části Request URL je zobrazena adresa, na kterou je daný požadavek zaslán. Část ResponseBody obsahuje odpověď ze strany aplikace. V odpovědi je vidět, že požadavek byl zpracován úspěšně, ale danému požadavku neodpovídají žádné záznamy.

## 3.4 ESG.Audit.Client

- **ClientBase**

  Třída ClientBase implementuje rozhraní IDisposable. Tato třída obsahuje metodu CallAPIMethodAsync, která zprostředkovává jednotlivé požadavky a v případě neúspěchu je zodpovědná za logování.
- **Client**

  Potomek třídy ClientBase. V této třídě jsou implementovány metody volající metody nabízené komponentou Host. V případě ukládání a snychronizace záznamů je zde další řídicí logika, která využívá lokální databáze.
- **AuditMethodClientResult**

  Pomocná třída, která slouží ke zpracování chybových hlášek a jejich výpisu.
- **IDatabase**

  Rozhraní, které definuje základní metody potřebné pro lokální databázi. Tyto metody jsou:

  - **AddNewEvents**

    Tato metoda slouží k uložení listu auditních záznamů. Díky této metodě je možné v případě výpadku spojení s hlavní databází uložit záznamy dočasně do databáze lokální.
  - **GetAllEvents**

    Pomocí metody GetAllEvents lze jednoduše získat vstupní data pro synchronizaci celé databáze.
  - **DeleteAllRecordsToDate**

    Po úspěšné synchronizaci je žádoucí smazat lokální záznamy, které již nemají využití a pouze zabírají místo na disku. Tato metoda vymaže všechny záznamy do určitého data dodaného parametrem. Toto datum by mělo obsahovat přesný čas poslední úspěšné synchronizace.

- **EventSystemImplementation**

  Tato třída implementuje rozhraní IStorage s využitím databáze SQLite. Pro přehlednost jednotlivých tabulek je databáze automaticky rozdělena na tabulky podle roku a měsíce, z kterého záznamy pochází.

## 3.5 ESG.Audit.MigrationCore

Z důvodu nasazení auditního serveru do již existujícího systému bude nezbytné všechny stávající záznamy přenést do nové databáze. Za tímto účelem byla vytvořena konzolová aplikace, která umožňuje synchronizace mezi databázemi MongoDB a SQLite.

- **Program**

  Hlavní třída konzolové aplikace. Metoda main zpracovává parametry popsané v třídě CmdLineOptions a ukládá výsledek synchronizace do souboru.

- **CmdLineOptions**

  V této třídě je definován způsob zápisu parametrů, které lze využít při spouštění aplikace z příkazového řádku. Parametry lze předávat pomocí jedné ze dvou metod

zápisu. Buď může být využit zápis -A X, kde A je zkratka názvu parametru a X hodnota parametru, nebo je možné využít zápis -Jméno X, kde Jméno je celý název parametru a X je hodnota parametru. Jednotlivé parametry se mohou vylučovat. Příkladem může být směr synchronizace MognoDB do SQLite a SQLite do MongoDB. V tomto případě je potřeba využít možnosti MutuallyExclusiveSet, která zaručí, že ze všech takto označených parametrů bude vybrán maximálně jeden. Nastavitelné parametry jsou:

- **DatabaseConnectionString**
  Tento parametr je povinný a reprezentuje připojovací řetězec do SQLite databáze.
- **ConnectionString**
  Parametr MongoConnectString je připojovací řetězec pro databázi MongoDB a je povinný.
- **DatabaseToMongo**
  Tento parametr slouží k nastavení směru synchronizace z SQLite do MongoDB. Tento parametr je společně s MongoToDatabase v MutuallyExclusiveSet a proto je možné vybrat pouze jeden z nich.
- **MongoToDatabase**
  Tento parametr slouží k nastavení směru synchronizace z MongoDB do SQLite.
- **TimeMode**
  Tento parametr je celočíselného typu int, nastavuje interval synchronizace a je možné ho nastavit na jednu z následujících možností:
  - 0 - bez časového omezení,
  - 1 - všechny záznamy od data specifikovaného parametrem Date,
  - 2 - rok shodný s parametrem Date,
  - 3 - měsíc shodný s parametrem Date,
  - 4 - den shodný s parametrem Date,
  - 5 - den ve kterém byla aplikace spuštěna.

  Výchozí hodnotou je zde 0.
- **Statistics**
  Pomocí parametru Statistics je možné zvolit, zda bude vytvořen soubor obsahující záznam o synchronizaci. Dále je možné zvolit, v jakém formátu budou data ukládána. Tato možnost je popsána v rozhraní IStatistics. Možnosti nastavení parametru Statistics jsou:
  - 0 - Soubor nebude vytvořen,
  - 1 - Soubor bude vytvořen a uložen ve formátu .csv.
- **Date**
  Parametr typu Datetime. Tento parametr je vyžadován pouze pokud je hodnota parametru TimeMode nastavená na hodnoty 1, 2, 3 nebo 4.
- **SourceId**
  Parametr SourceId je povinný. Záznamy v centrální databázi obsahují položku SourceId a SourceType, aby bylo možné rozlišit zdroj záznamů. V případě synchronizace je tedy potřeba tyto dodatečné údaje předat parametrem.

- **SourceType**
  Parametr SourceType je povinný.

- **EventSystemImplementation**
  Třída umožnující přístup do databáze SQLite. Tato třída je velmi podobná třídě EventSystemImplementation v části ESG.Audit.Client, ale má pouze minimální funkcionalitu potřebnou k provedení synchronizace.

- **DatabaseSynchronization**
  Synchronizace záznamů je prováděna pomocí dílčích kroků v podobě synchronizace jednotlivých dnů. Třída DatabaseSynchronization vytvoří objekt typu IStatistics a poté postupně sesynchronizuje veškeré záznamy po jednotlivých dnech.

- **IStatistics**
  Rozhraní definující potřebné metody pro vytváření záznamů o synchronizaci. Definované metody jsou následující:

  - AddHeaderToStatistics
    Tato metoda je volána po vytvoření objektu typu IStatistics a slouží k zápisu hlavičky. Tato hlavička obsahuje informace o směru synchronizace a zvoleném formátu.

  - AddDayToStatistics
    Přidá záznam o synchronizaci jednoho dnu. Parametrem je datum daného dnu, počet úspěšně přidaných záznamů a počet záznamů přeskočených, které již jsou v cílové databázi uloženy.

  - AddSummaryToStatistics
    Tato metoda je volána na konci celé synchronizace a slouží k zápisu celkového počtu přidaných a přeskočených záznamů.

  - string GetStatistics
    Návratovou hodnotou této metody je celá statistika ve formě stringu.

  - string GetFileNameExtension
    Tato metoda vrací koncovku souboru zvoleného formátu. společně s metodou GetStatistics je možné snadno uložit záznam do souboru.

- **StatisticCSV**
  Tato třída implementuje rozhraní IStatistics pro formát csv.

- **StatisticNone**
  Tato třída implementuje rozhraní IStatistics, nevytváří žádný soubor, ale může být časem využita pro interní logování.

## 3.6 ESG.Audit.EngineUnitTest

Tato knihovna obsahuje jednotkové testy ke výpočetní části celé aplikace a je podrobně popsána v kapitole testování.

## 3.7 ESG.Audit.Client.UnitTests

Tato knihovna obsahuje integrační testy aplikace a je podrobně popsána v kapitole testování.

# Kapitola 4
# Testování

Primárním cílem implementovaného testování je zaručení správné funkčnosti celé aplikace. Jedinou výjimkou je podsekce 4.3, ve které je testováno indexování databáze MongoDB za účelem optimalizace rychlosti přístupu k záznamům v databázi. Ostatní implementované testování poskytuje záruku funkčnosti celé aplikace a umožňuje kontrolu v případě budoucích změn implementace některých částí aplikace.

Veškerá uvedená data z provedených testů, jako využití procesoru, paměti či časová náročnost jednotlivých testů, jsou zaznamenána z průběhů testů ve vývojovém prostředí Visual Studio 2017, které bylo spouštěno na notebooku HP Elitebook Folio 9470m s následující specifikací:

- Operační systém: Windows 10 pro,
- Procesor: Intel Core i5-3437U,
- Pamět RAM: 16 GB DDR3,
- Grafická karta: Integrovaná - Intel graphics 4000,
- Disk: Samsung 850 EVO 500GB.

## 4.1 Nastavení unit testů

Jednotkové testy jsou implementovány pomocí technologie xUnit [4].

V knihovně ESG.NetCore.Audit.Engine.UnitTests se nachází třída DatabaseFixture, která zajišťuje spuštění databáze MongoDB. S využitím technologie xUnit není možné používat metody SetUp a TearDown, které jsou velmi často využívány při testování pomocí technologie NUnit[1]. Při spuštění testování je možné pomocí parametru vybrat, jak bude databáze MongoDB spuštěna. Možnosti jsou následující:

- **MongoDB database server**

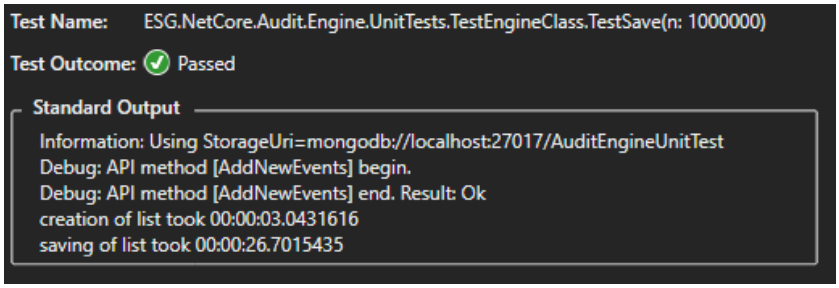
  Standardní varianta MongoDB, která je určena pro běžný provoz. Vyžaduje instalaci na zařízení, na kterém jsou testy prováděny.

- **Mongo2Go**

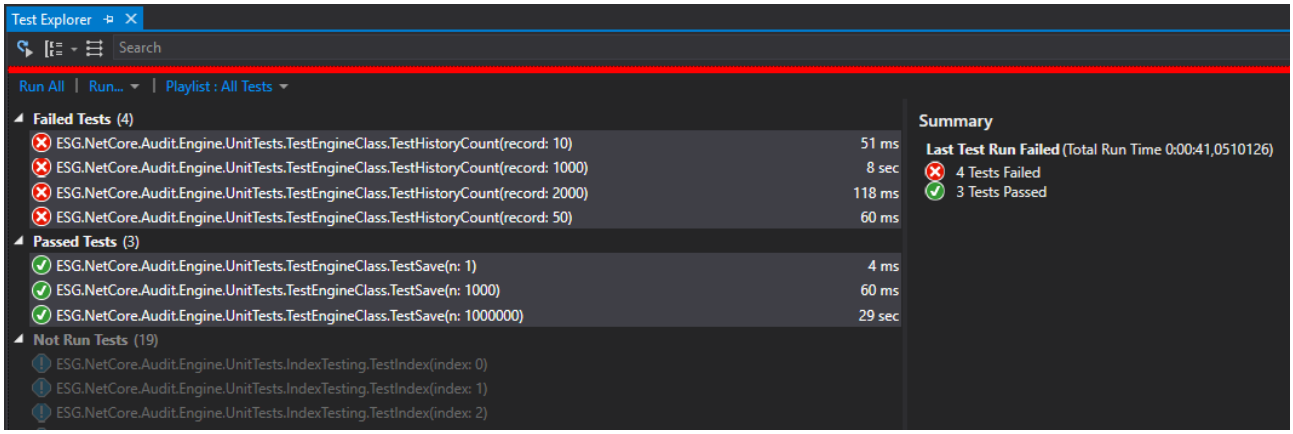
  Tato varianta je speciálně navržena pro integrační testy. Je možné vytvořit instanci databáze i bez instalace MongoDB. Po provedení testů se instance databáze ukončí a již nepotřebná data jsou smazána. Proto je tato možnost ideální na testování, pokud není určeno na jakém zařízení budou testy spouštěny.

Při spouštění testů je použita třída ITestOutputHelper, která umožňuje logování k výsledkům jednotlivých testů. Pomocí této třídy je možné zapisovat k testům například jejich čas či průběžné stavy proměnných. Instanci této třídy je potřeba pomocí Dependency injection předat do třídy DatabaseFixture, poté je možné si záznamy

[1] `https://github.com/nunit/docs/wiki/NUnit-Documentation`




**Obrázek 4.1.** Ukázka logování konkrétního testu s využitím třídy ITestOutputHelper.




**Obrázek 4.2.** Ukázka přehledu testů v programu Visual Studio 2017.

prohlížet ve VisualStudiu po kliknutí na konkrétní již dokončený test v záložce TestExplorer. Ukázka tohoto výpisu je na obrázku 4.1.

Na obrázku 4.2 je zobrazena ukázka testování v programu Visual Studio 2017. Testy jsou zde rozděleny podle výsledku. Červenou značkou jsou označeny testy, které neprošly, zelenou značkou jsou označeny testy, u kterých byly splněny všechny příkazy typu Assert a modrou značkou jsou označeny testy, které ještě nebyly provedeny. Na pravé straně vedle testů jsou vypsány časy trvání jednotlivých testů.

### 4.1.1 Testování Engine a Hlavní databáze

V této části jsou stručně popsány všechny jednotkové testy. Pro potřeby testování byla vytvořena třída TestHelper, která obsahuje pomocné metody pro hlavní testovací třídu TestEngineClass. Tato pomocná třída umožňuje provést testování bez nutnosti dalších pomocných souborů. Metody pomocné třídy TestHelper jsou následující:

- **GetRandomString(int length)**
  Tato metoda vrátí textový řetězec o délce zadané parametrem.
- **GetRecord()**
  Tato metoda vrací objekt typu AuditRecord, který s nastaveným Date na DateTime.UtcNow a id na Guid.empty. Zbytek parametrů je volen náhodně.
- **GetRecord (Guid id)**
  Tato metoda rozšiřuje metodu GetRecord() o možnost nastavit id na hodnotu dodanou parametrem.
- **GetAuditWithHistory(int i,Guid id)**
  Tato metoda vrací list záznamů, které mají stejné id. Počet položek listu je omezen celočíselným parametrem i.

### 4.1.2 Unit testy

V této sekci budou stručně popsány jednotlivé jednotkové testy. Podrobněji budou popsány pouze vybrané složitější testy. Veškeré testy se nacházejí v třídě TestEngineClass.cs a jsou následující:

- **StorageReadyTest**
  Tento test ověřuje metodu PingAsync a to s využitím volitelného parametru ReadyCheck = True. Očekávaným výstupem je zde objekt PingOutput, který má následující proměnné:
  - DateTime CurrentTimeUTC - Tato proměnná obsahuje čas, ve kterém k dotazu došlo.
  - bool? StorageIsReady - Výchozí hodnota této proměnné je null. V případě využití parametru ReadyCheck je její hodnota změněna na True/False podle stavu databáze.

- **TestSave(int n)**
  Tento test je označen pomocí tagu [Theory], což v technologii xUnit umožňuje spustit ten samý test vícekrát s různými parametry. V tomto případě je parametrem určen počet záznamů, které budeme ukládat do databáze. Test můžeme rozdělit na dvě části. Nejprve vygenerujeme list auditních záznamů pomocí třídy TestHelper a následně se jej pokusíme uložit do databáze. Výsledné časy jsou uvedeny v tabulce 4.1.

| n | Čas vytváření listu | Čas ukládání do databáze | normalizovaný čas |
|---|---|---|---|
| 1 | 0,0000094 s | 0,0007937 s | 0,7937000 s |
| 1000 | 0,0011812 s | 0,1759115 s | 0,1759115 s |
| 1000000 | 3,7196097 s | 27,2018801 s | 0,0272019 s |

**Tabulka 4.1.** Čas vytváření a ukládání listu záznamů do databáze pro různé hodnoty parametru n (počet záznamů) v testu ParalelAddRecord. Normalizovaný čas je vypočtená doba, za kterou by se uložilo tisíc záznamů.

- **ParalelAddRecord(int records, int threadCount)**
  Tento test testuje přidávání záznamů do databáze pomocí více paralelně běžících vláken. Jednotlivé varianty testu se liší v rozdělení záznamů mezi jednotlivá vlákna. Celkem je vždy uložen konstantní počet záznamů, které jsou rovnoměrně rozděleny mezi všechna vlákna. Výchozí maximální hodnota ThreadPool je počet jader počítače, která je v případě použitého procesoru Intel Core i5-3437U rovna čtyřem. Test s více vlákny tedy nebude zcela paralelní. Časy pro jednotlivé hodnoty parametrů records a threadCount jsou uvedeny v tabulce 4.2.

- **TestHistory()**
  Tento test slouží k ověření správně funkce metody GetEventHistory. Do databáze je nejprve vloženo 100 záznamů s vyplněným projectID. Poté jsou tyto záznamy načteny z databáze a porovnány se záznamy původními.

- **TestHistoryCount(int record)**
  Tento test ověřuje vlastnosti funkce TestHistory. Maximální počet záznamů, které databáze vrací, je omezen na 1000. Pokud je požadován menší počet záznamů,

je možné nastavit příslušný počet do proměnné ValueCount objektu GetHistoryInput. Pokud se v daném intervalu nachází více než 1000 záznamů, je potřeba je načítat postupně pomocí využité proměnné ValueOffset.

| Počet záznamů | Počet vláken | Čas ukládání do databáze | Normalizovaný čas |
|---|---|---|---|
| 8000 | 1 | 0,1979023 s | 0,0247377 s |
| 4000 | 2 | 0,1560204 s | 0,0195026 s |
| 2000 | 4 | 0,2664014 s | 0,0333002 s |
| 1000 | 8 | 0,1080696 s | 0,0135087 s |
| 80 | 100 | 0,2252534 s | 0,0281566 s |
| 80000 | 1 | 1.5936003 s | 0,0199200 s |
| 40000 | 2 | 1.0169242 s | 0,0127115 s |
| 20000 | 4 | 1.0029408 s | 0,0125367 s |
| 10000 | 8 | 1.0623947 s | 0,0132799 s |
| 800 | 100 | 1.3791632 s | 0,0172354 s |

**Tabulka 4.2.** Celkový čas ukládání záznamů do databáze pro různé parametry records a threadCount v testu ParalelAddRecord. Normalizovaný čas je vypočtená doba, za kterou by se uložilo tisíc záznamů.

## 4.2 Integrační testy

Integrační testy slouží k ověření funkčnosti projektu jako celku. Ověřují komunikaci mezi jednotlivými komponentami. V ideálním případě by integrační testy měly probíhat mezi komponentami, které jsou jednotlivě otestované. V tomto projektu jsou implementované testy mezi komponentami Engine, Host a Client. Engine je testován pomocí jednotkový testů viz sekce 4.1.2. Komponenta Host umožňuje interaktivní testování pomocí swaggeru a Client není samostatně testován. Absence testování komponenty Client je způsobena jednoduchostí této komponenty, a proto z časových a finančních důvodů jsou implementovány pouze testy integrační.

Integrační testy se nacházejí v dílčím projektu ESG.Audit.Client.Test ve třídě Program.cs. Ke spuštění těchto testů je potřeba nejprve spustit komponentu Host, ke které se poté nově vytvořený klient připojí a otestuje jednotlivé metody. Spuštění více projektů najednou lze ve VS pomocí kliknutí na dílčí projekt pravým tlačítkem a zvolením možnosti Debug - Start new instance, nebo nastavením výchozího projektu na `Multiple Startup Project`.

## 4.3 Indexování hlavní databáze MongoDB

Správné indexování je jedním ze základních předpokladů pro rychlý chod celé databáze. Databáze MongoDB nabízí více druhů indexů, viz [8]:

- **Jednoduchý index (Single field)**
  Tento druh indexu je podobný indexování například v SQL databázích a obsahuje pouze jednu položku daného záznamu. Tento index je automaticky vytvořen na položce ID, pokud není specifikován index jiný.
- **Složený index (Compound)**
  Tento druh indexu obsahuje kombinaci více položek, proto je velmi výhodný, pokud je známé, podle jaké kombinace parametrů budou záznamy vyhledávány. V případě tohoto projektu to bude kombinace položky ProjectId a Timestamp.

- **Výcenásobný index (Multikey Index)**
  Tento druh indexu se využívá pro záznamy obsahující pole. Je vytvořen index pro všechny položky v poli, což vede k efektivnímu dotazování na záznamy, které obsahují daný element pole.

Testování indexů je implementováno s využitím složeného indexu, konkrétně kombinací položky ProjectId a Timestamp. Primárním cílem použití daného typu indexu je minimalizace doby trvání dotazu GetHistory, tedy zlepšení schopnosti databáze rychle najít a přečíst záznamy. Doba zápisu není pro tento projekt klíčová. Na začátku testu proběhne uložení celkem 1 000 000 záznamů, kde 999 jich obsahuje dané projectId. Následně proběhne dotazování na historii daného projektu. Výsledky tohoto testu pro jednotlivé indexy jsou uvedeny v tabulce 4.3.

| Index ProjectId | Index Timestamp | Doba T1 | Doba T2 |
|---|---|---|---|
| Žádný | Žádný | 19,0729620 s | 7,8416357 s |
| Ascending | Ascending | 27,3713057 s | 0,3293460 s |
| Descending | Descending | 26,4829307 s | 0,3235231 s |
| Geo2D | Geo2D | 20.7027267 s | 9,0227071 s |
| Descending | Hashed | 19,0225245 s | 7,5658190 s |
| GeoHaystack | GeoHaystack | 18,6425203 s | 7,5411088 s |
| Hashed | Hashed | 19,3796057 s | 8,0532540 s |

**Tabulka 4.3.** Časy zápisu a vyhledávání v databázi s využitím různých kombinovaných indexů. T1 je doba zápisu milionu záznamů. T2 je doba za kterou bylo desetkrát vyhledáno 999 záznamů s daným projectId.

Vzhledem k požadavkům na rychlost vyhledávání záznamů je na základě těchto dat nastaveno indexování pomocí kombinovaného indexu:

- Timestamp - Descending,
- ProjectId - Descending.

# Kapitola 5
# Závěr

Cílem této bakalářské práce bylo navrhnout, implementovat a otestovat aplikaci pro čtení a zápis auditních záznamů, která bude využita firmou Energocentrum Plus. Bylo popsáno více způsobů, jak mohou systémy na ukládání dat vypadat. Aplikace auditního serveru byla navržena ve formě hybridního systému, který využívá centrálního řídicího prvku s hlavní databází a mnoha dalších komponent, které mohou obsahovat své vlastní databáze.

Po konzultaci se zástupci firmy Energocentrum Plus byly vybrány vhodné technologie pro implementaci této aplikace. Vybranou technologií pro implementaci auditního serveru je .NetCore verze 2.0. Mezi databázemi byla vybrána NoSQL databáze MongoDB, která je využita jeho hlavní databáze aplikace a databáze relační SQLite, která je využita pro ostatní části systému pro dočasné ukládání záznamů.

Tato struktura a technologie byly zvoleny za účelem zlepšení výkonu ostatních komponent a maximalizace dostupnosti auditních záznamů. Ve spolupráci s firmou Energocentrum Plus nadále pokračuji ve vývoji a rozšiřování této aplikace. Nad rámec této práce je vyvíjeno rozšíření auditního serveru o zpracování parametrů jednotlivých záznamů a je dále optimalizována synchronizace s ostatními databázemi. Za využitím navržené struktury je vyvíjen logovací server, který využívá některá v této bakalářské práci navržená rozhraní a dále rozšiřuje funkcionalitu databázových operací o periodické promazávání záznamů, rozdělení událostí podle úrovní či podrobnější možnosti filtrování dotazovaných dat.

Po navržení struktury celé aplikace byla podrobně popsána implementace včetně popisu jednotlivých tříd a stěžejních metod. Na závěr celého projektu byly vytvořeny jednotkové a integrační testy, které testují funkcionalitu celého systému a umožňují kontrolu v případě změn implementace.

Do navržené aplikace spuštěné v testovacím prostředí bylo úspěšně vloženo několik miliónů záznamů a proběhlo komplexní testování. Věřím, že jejím nasazením bude dosaženo všech vlastností a cílů, pro které byla navrhována.

# Literatura


[1] Čermák Miroslav - Důvody k nasazení NoSQL , a speciálně MongoDB. Bakalářská práce. Vysoká škola ekonomická v Praze, fakulta informatiky a statistiky. Vedoucí práce RNDR. Helena Pavlovská, Ph.D. https://vskp.vse.cz/id/1268285 [cit. 13.04.2018]

[2] Matoušek Lukáš - Best Practices pro xUnit testování. Bakalářská práce. Vysoká škola ekonomická v Praze, fakulta informatiky a statistiky. Vedoucí práce Ing. Pavlíčková Jarmila https://vskp.vse.cz/id/1243980 [cit. 13.04.2018]

[3] Friedrich Steimann, Philip Mayer: Patterns of Interface-Based Programming, in Journal of Object Technology, vol. 4, no. 5, July-August 2005, pp. 75-94 http://www.jot.fm/issues/issue_2005_07/article1

[4] Oficiální dokumentace technologie xUnit https://xunit.github.io/ [cit. 13.04.2018]

[5] Oficiální dokumentace k technologii .NetCore, Microsoft https://docs.microsoft.com/en-us/dotnet/core/ [cit. 13.04.2018]

[6] Michael Kennedy - MONGODB VS. SQL SERVER 2008 PERFORMANCE SHOWDOWN. Published April 29, 2010 by Michael Kennedy in NoSQL https://blog.michaelckennedy.net/2010/04/29/mongodb-vs-sql-server-2008-performance-showdown/ [cit. 13.04.2018]

[7] Chieh Ming Wu, Yin Fu Huang, John Lee. Comparisons between MongoDB and MS-SQL Databases on the TWC Website. American Journal of Software Engineering and Applications. Vol. 4, No. 3, 2015, pp. 35-41. doi: 10.11648/j.ajsea.20150402.12 ,ISSN: 2327-2473

[8] Oficiální dokumentace databáze MongoDB, MongoDB Inc.https://docs.mongodb.com/ [cit. 13.04.2018]

[9] Ameya Nayak, Anil Poriya, Dikshay Poojary : Type of NOSQL Databases and its Comparison withRelational Databases, in International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5– No.4, March 2013 – www.ijais.org

[10] ShefaliPatil, GauravVaswani, Anuradha Bhatia : Graph Databases- An Overview in ShefaliPatil et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (1) , 2014, 657-660, ISSN:0975-9646

[11] DB-Engines Ranking - popularity ranking of database management systems. https://db-engines.com/en/ranking/ [cit. 13.04.2018]

[12] S.T. Bhosale, Miss. Tejaswini Patil, Miss. Pooja Patil : SQLite: Light Database System, in S.T. Bhosale et al, International Journal of Computer Science and Mobile Computing, Vol.4 Issue.4, April- 2015, pg. 882-885, ISSN 2320–088X

[13] Supriya S. Pore, Swalaya B. Pawar : Comparative Study of SQL & NoSQL Databases in International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 4 Issue 5, May 2015, ISSN: 2278 – 1323

# Příloha A
# Zkratky

| | |
|---|---|
| AOP | Aspektově orientované programování |
| API | Application Programming Interface - rozhraní pro programování aplikací |
| ČVUT | České vysoké učení technické v Praze |
| FEL | Fakulta elektrotechnická ČVUT |
| HVAC | Heating, ventilation, air conditioning |
| RAM | Random access memory - paměť s náhodným přístupem |
| SCADA | Supervisory Control And Data Acquisition - dispečerské řízení a sběr dat |
| SW | Software - programové vybavení |
| TZB | Technické zařízení budov |
| VS | Visual Studio |

# Příloha B
## Zadání

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

### I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zázvorka** Jméno: **Martin** Osobní číslo: **457217**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Systémy a řízení**

### II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Implementace služby pro zápis a čtení auditních záznamů**

Název bakalářské práce anglicky:

**Implementation of service for writing and reading audit records**

Pokyny pro vypracování:

1. Zhodnoťte současné databázové technologie pro práci s komplexními záznamy, relační a nerelační databáze z hlediska rychlosti, spolehlivosti, dostupnosti dokumentace, atd.
2. Vyberte vhodnou technologii a navrhněte aplikaci auditního serveru, který spravuje data z různých SCADA systémů.
3. Aplikaci implementujte v prostředí .NET za pomoci technologie .NetCore 2.0 pro aplikaci a .NetStandard 2.0 pro knihovny.
4. Aplikaci otestujte pomocí jednotkových testů (unit testů) a komplexnějších migračních testů.

Seznam doporučené literatury:

[1] Čermák M., Důvody k nasazení NoSQL a speciálně MongoDB. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky.
https://vskp.vse.cz/id/1268285 , 2015
[2] Matoušek, L. Best Practices pro xUnit testování. Bakalářská práce. Vysoká
škola ekonomická v Praze, fakulta informatiky a statistiky, https://vskp.vse.cz/id/1243980, 2014.
[3] Steimann, F. a Mayer, P.: Patterns of Interface-Based Programming,
in Journal of Object Technology, vol. 4, no. 5, July-August 2005, pp. 75-94
http://www.jot.fm/issues/issue 2005 07/article1

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Jan Šulc, UCCEB Buštěhrad**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.01.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

| Ing. Jan Šulc<br>podpis vedoucí(ho) práce | prof. Ing. Michael Šebek, DrSc.<br>podpis vedoucí(ho) ústavu/katedry | prof. Ing. Pavel Ripka, CSc.<br>podpis děkana(ky) |
|---|---|---|

### III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta