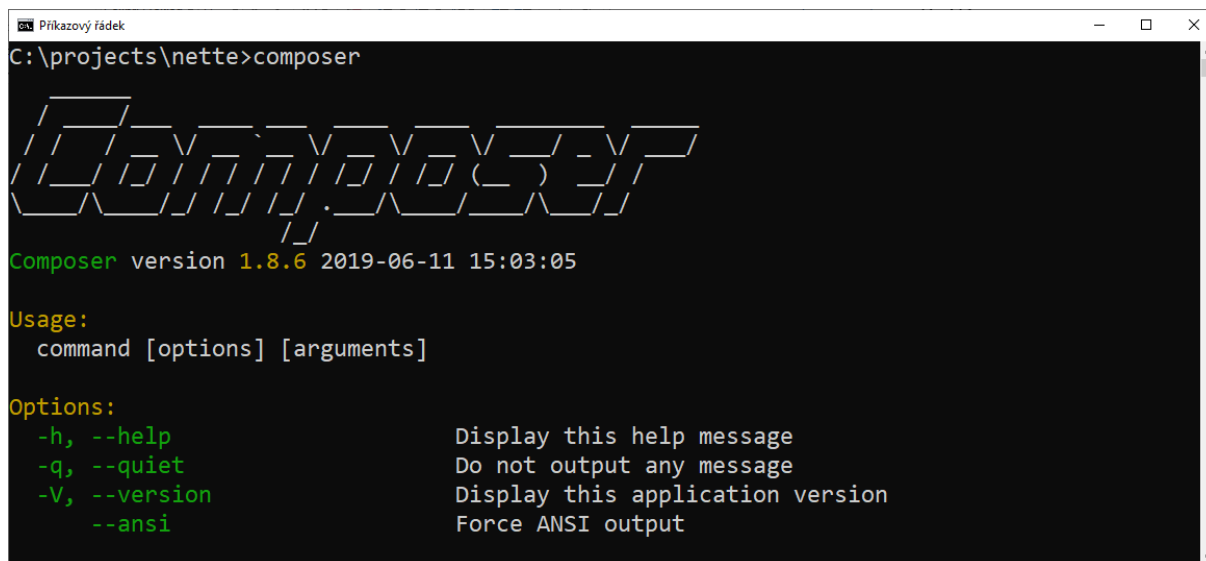


KALENDÁŘ AKCÍ V NETTE FRAMEWORK

I. PŘÍPRAVA SANDBOXU

1. Instalace nástroje composer.

Composer je multiplatformní nástroj pro snadnou správu závislostí v PHP. Dovoluje nám deklarovat knihovny, na kterých je náš PHP projekt závislý, a následně tyto závislosti instalovat a aktualizovat. Můžeme ho stáhnout ze stránky <https://getcomposer.org/download/>. Composer by měl být po instalaci integrován v daném operačním systému a spouštěn z příkazového řádku.



```
Příkazový řádek
C:\projects\nette>composer

Composer version 1.8.6 2019-06-11 15:03:05

Usage:
  command [options] [arguments]

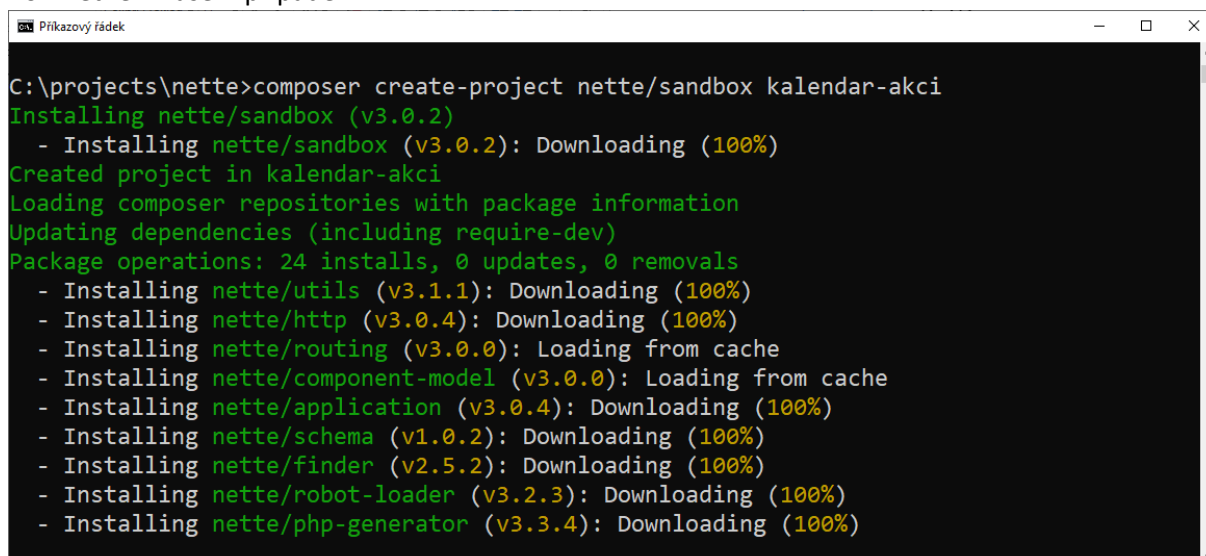
Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
  --ansi               Force ANSI output
```

2. Vytvoření nového projektu Nette Framework.

Použijeme composer a v připravené složce (zde c:\projects\nette) pomocí něj vytvoříme nový projekt v Nette na základě sandboxu, který je k dispozici na <https://github.com/nette/sandbox>. Zde rovněž najdeme podrobnější návod k použití.

```
composer create-project nette/sandbox path/to/install
cd path/to/install
```

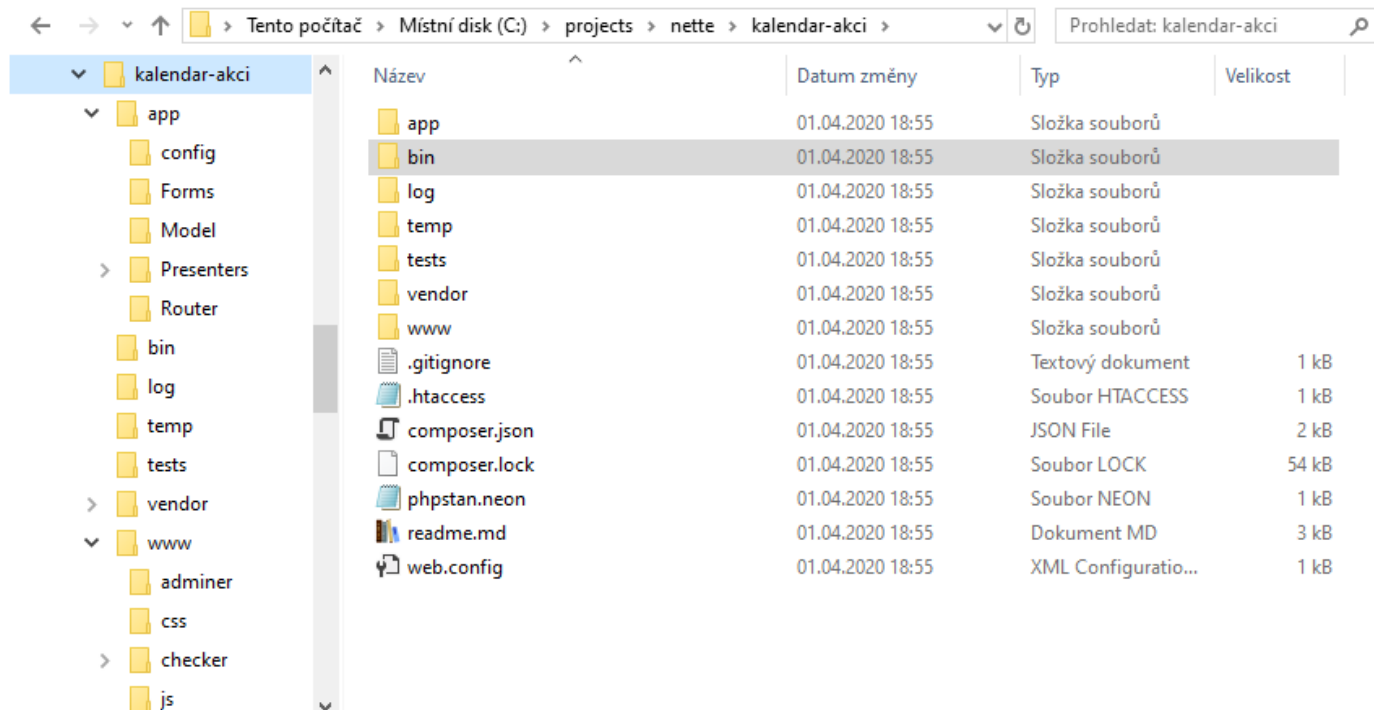
Konkrétně v našem případě:



```
Příkazový řádek
C:\projects\nette>composer create-project nette/sandbox kalendar-akci
Installing nette/sandbox (v3.0.2)
- Installing nette/sandbox (v3.0.2): Downloading (100%)
Created project in kalendar-akci
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 24 installs, 0 updates, 0 removals
- Installing nette/Utils (v3.1.1): Downloading (100%)
- Installing nette/http (v3.0.4): Downloading (100%)
- Installing nette/routing (v3.0.0): Loading from cache
- Installing nette/component-model (v3.0.0): Loading from cache
- Installing nette/application (v3.0.4): Downloading (100%)
- Installing nette/schema (v1.0.2): Downloading (100%)
- Installing nette/finder (v2.5.2): Downloading (100%)
- Installing nette/robot-loader (v3.2.3): Downloading (100%)
- Installing nette/php-generator (v3.3.4): Downloading (100%)
```

3. Struktura projektu

Po instalaci sandboxu byla v základní složce projektu (kalendar-akci) vytvořena řada podsložek, které tvoří kostru našeho webového projektu. Pro samotný vývoj našeho webu hrají nejvýznamnější roli složky **app** (zde se nachází celá struktura námi vyvíjené aplikace - formuláře, modely, prezenty (včetně šablon Latté), konfigurační soubory i router, obsahující pravidla pro směrování požadavků na konkrétní části naší aplikace).



Název	Datum změny	Typ	Velikost
app	01.04.2020 18:55	Složka souborů	
bin	01.04.2020 18:55	Složka souborů	
log	01.04.2020 18:55	Složka souborů	
temp	01.04.2020 18:55	Složka souborů	
tests	01.04.2020 18:55	Složka souborů	
vendor	01.04.2020 18:55	Složka souborů	
www	01.04.2020 18:55	Složka souborů	
.gitignore	01.04.2020 18:55	Textový dokument	1 kB
.htaccess	01.04.2020 18:55	Soubor HTACCESS	1 kB
composer.json	01.04.2020 18:55	JSON File	2 kB
composer.lock	01.04.2020 18:55	Soubor LOCK	54 kB
phpstan.neon	01.04.2020 18:55	Soubor NEON	1 kB
readme.md	01.04.2020 18:55	Dokument MD	3 kB
web.config	01.04.2020 18:55	XML Configuratio...	1 kB

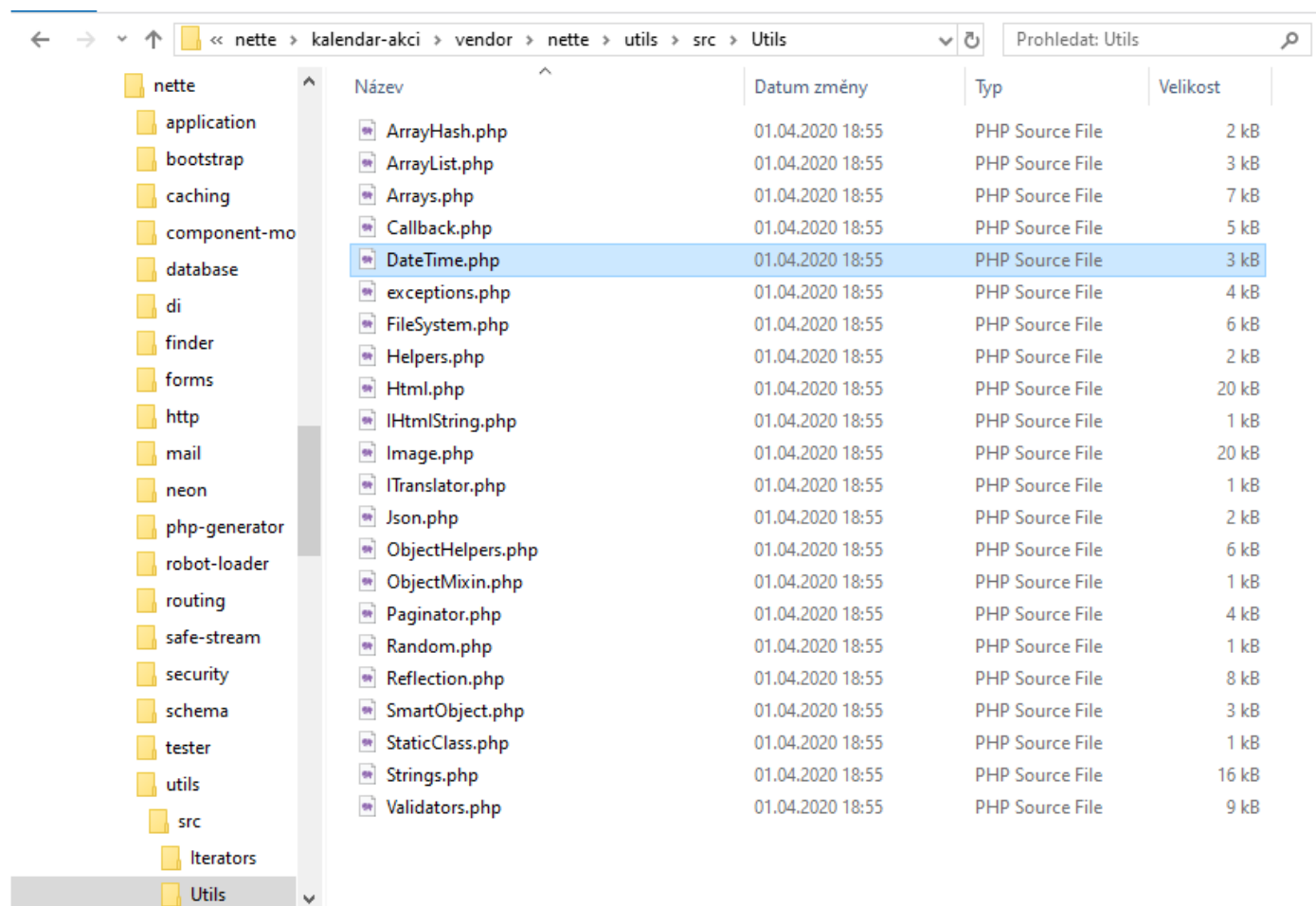
- Do složky **bin** mohou být ukládány php skripty, které usnadňují nějaké užitečné operace. Například pomocí skriptu **create-user.php** můžeme vytvořit uživatelské účty pro autorizovaný přístup k některým částem aplikace.
- Ve složce **log** se nacházejí logovací soubory, do kterých jsou zapisovány informace o případných chybách v aplikaci nebo o sledovaných objektech, podle záměrů autora aplikace.
- Složka **temp** slouží k ukládání dočasných souborů (například během jejich uploadu) a obsahuje také cache - vyrovnávací paměť aplikace. Jejím cílem je v produkčním nasazení zajistit co nejrychlejší načítání jednotlivých stránek. Během vývoje aplikace však může způsobit zmatek v hlavě méně zkušeného programátora, když se mu například díky cache stále ukazují chyby, které již odstranil. Cache je v takovém případě možné promazat, ještě lépe je ji však během vývojové fáze v konfiguraci Nette vypnout.
- Složka **tests** se využívá k testování aplikace. Před nasazením do reálného provozu je vhodné podle známých metodik (např. tzv. unit testy) připravit sadu testovacích skriptů, které prověří funkčnost a integritu jednotlivých částí aplikace.
- Složka **vendor** obsahuje všechny programové moduly (knihovny), které aplikace využívá (nebo i nevyužívá :-)).
- Do složky **www** se umísťují tzv. statické soubory, které jsou spojeny s jednotlivými stránkami webu - kaskádové styly (složka css), skripty (složka js), grafické přílohy, případně celé javaskriptové knihovny (Bootstrap, JQuery UI, DataTables, TinyMCE atd.). Je zde rovněž vstupní skript celé webové aplikace - soubor **index.php**.
- Důležité soubory se nacházejí v kořenové složce projektu. Soubor **composer.json** specifikuje, jaké závislosti daný projekt má. V JSON formátu jsou v něm definovány údaje jako povinný název balíčku (sestavá z názvu tvůrce a názvu projektu oddělených lomítkem), popis balíčku, typ balíčku, klíčová slova balíčku, licenci balíčku atd. (Všechny možnosti naleznete v oficiální dokumentaci na adrese <https://getcomposer.org/...04-schema.md>). Do souboru **composer.lock** bychom zasahovat neměli, vytváří se totiž automaticky, a je vlastně aktuálním "snímek" všech závislostí v aplikaci. Oba soubory se

využijí v případě, kdy chceme buď závislosti znovu nainstalovat (příkazem `composer install`), nebo provést aktualizaci používaných modulů na novější verze (`composer update`). Více o této problematice naleznete např. na stránce <https://www.itnetwork.cz/php/ostatni/composer>.

- Soubory **.gitignore** a **readme.md** souvisejí s použitím verzovacího systému git (viz níže).
- Soubor **.htaccess** je konfigurační soubor webového serveru Apache. Obsahuje pravidla, pomocí nichž můžeme konfigurovat, chování serveru vůči naší webové aplikaci (například přístupnost některých složek, přesměrování, zpracování URL požadavků apod.). Aby byl **.htaccess** aktivní, musí ho administrátor Apache povolit v souboru `httpd.conf`. Textový konfigurační soubor se nejčastěji ukládá do kořenového adresáře webu. Do jednotlivých složek lze vložit další soubor, který bude určovat vlastnosti konkrétní složky (ta by jinak přejímala pravidla výše uložené **.htaccess**). Více o problematice najdete např. zde: <http://www.htaccess.cz/>
- Soubor **web.config** řeší obdobně chování jednotlivých složek, které tvoří strukturu webu, na webových serverech od Microsoftu. Má strukturu XML souboru.
- Soubor **phpstan.neon** slouží ke konfiguraci nástroje PHPStan, což je analytická utilita k odchyťování chyb v PHP aplikacích bez nutnosti psaní testů (více např. <https://ondrej.mirtes.cz/phpstan-hledani-chyb-v-kodu-bez-psani-testu>).

4. Moduly Nette

V tomto sandboxu jsou nyní nainstalovány důležité (ale i některé méně důležité) součásti běžných aplikací v Nette - kromě jádra samotného Nette například šablonovací systém Latté, debuggovací nástroj Tracy ("Laděnka"), moduly zajišťující cachování, routování, autentizaci a zabezpečení, rychlou tvorbu formulářů, odesílání mailů atd. Všechny nainstalované moduly se v projektu nacházejí ve složce `vendor` a podsložkách (nejvíce jich je ve složce `nette`).



	Název	Datum změny	Typ	Velikost
nette				
application				
bootstrap				
caching				
component-mo				
database				
di				
finder				
forms				
http				
mail				
neon				
php-generator				
robot-loader				
routing				
safe-stream				
security				
schema				
tester				
utils				
src				
Iterators				
Utils				
	ArrayHash.php	01.04.2020 18:55	PHP Source File	2 kB
	ArrayList.php	01.04.2020 18:55	PHP Source File	3 kB
	Arrays.php	01.04.2020 18:55	PHP Source File	7 kB
	Callback.php	01.04.2020 18:55	PHP Source File	5 kB
	DateTime.php	01.04.2020 18:55	PHP Source File	3 kB
	exceptions.php	01.04.2020 18:55	PHP Source File	4 kB
	FileSystem.php	01.04.2020 18:55	PHP Source File	6 kB
	Helpers.php	01.04.2020 18:55	PHP Source File	2 kB
	Html.php	01.04.2020 18:55	PHP Source File	20 kB
	IHtmlString.php	01.04.2020 18:55	PHP Source File	1 kB
	Image.php	01.04.2020 18:55	PHP Source File	20 kB
	ITranslator.php	01.04.2020 18:55	PHP Source File	1 kB
	Json.php	01.04.2020 18:55	PHP Source File	2 kB
	ObjectHelpers.php	01.04.2020 18:55	PHP Source File	6 kB
	ObjectMixin.php	01.04.2020 18:55	PHP Source File	1 kB
	Paginator.php	01.04.2020 18:55	PHP Source File	4 kB
	Random.php	01.04.2020 18:55	PHP Source File	1 kB
	Reflection.php	01.04.2020 18:55	PHP Source File	8 kB
	SmartObject.php	01.04.2020 18:55	PHP Source File	3 kB
	StaticClass.php	01.04.2020 18:55	PHP Source File	1 kB
	Strings.php	01.04.2020 18:55	PHP Source File	16 kB
	Validators.php	01.04.2020 18:55	PHP Source File	9 kB

Orientace v této struktuře je užitečná zejména ve chvíli, když chceme v aplikaci nějaký nástroj využít. Děje se tak pomocí klíčového slova `use`. Například, chceme-li použít utilitu pro zpracování data a času, odkážeme na PHP třídu `DateTime`, která se nachází ve jmenném prostoru `\Nette\Utils`:

```
use \Nette\Utils\DateTime
```

5. Spuštění webového serveru

Provedeme z kořenové složky aplikace zadáním příkazu:

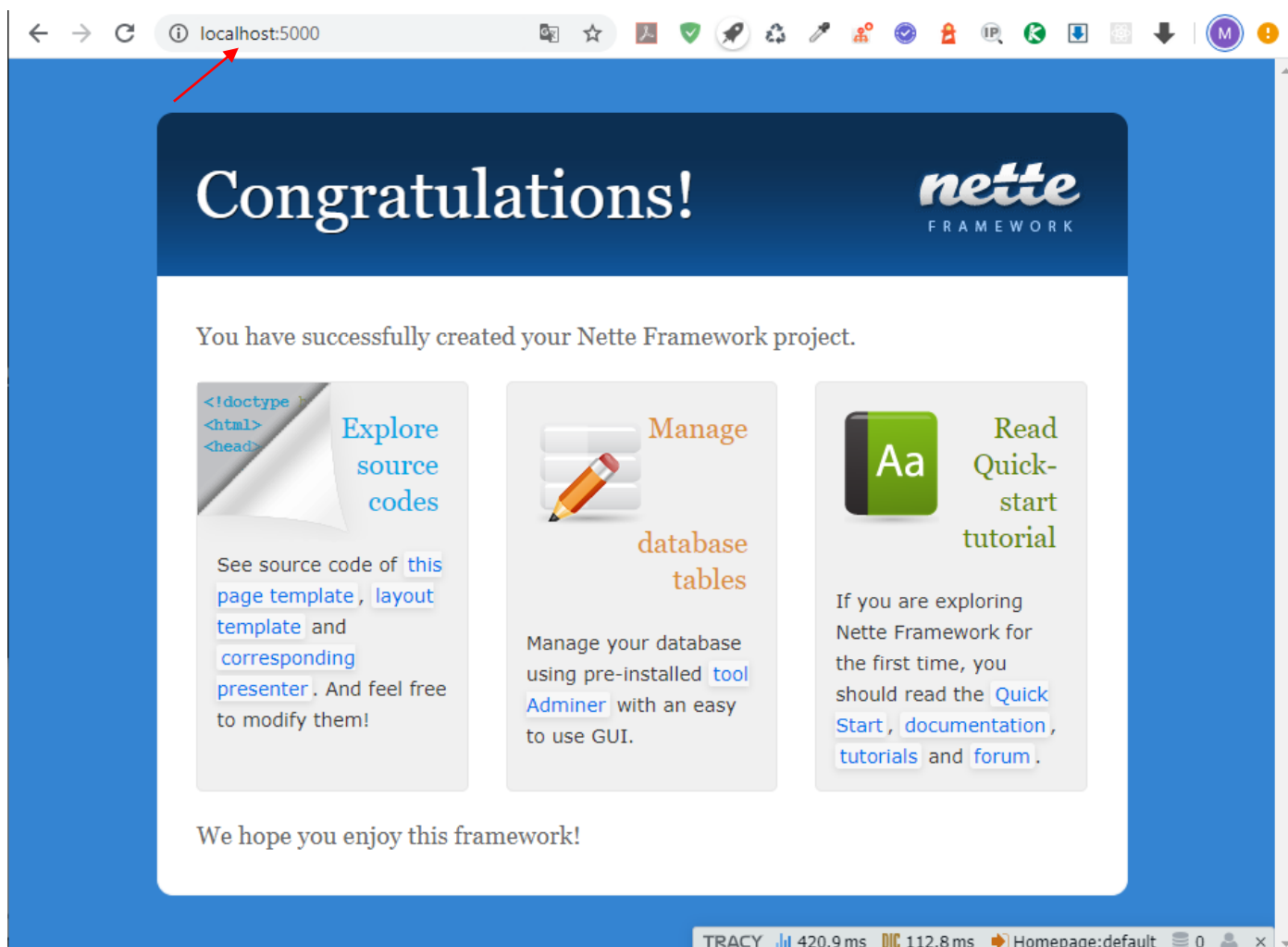
```
php -S localhost:5000 -t www
```

```
Příkazový řádek
C:\projects\nette>cd kalendar-akci

C:\projects\nette\kalendar-akci>php -S localhost:5000 -t www
PHP 7.3.7 Development Server started at Wed Apr  1 20:40:29 2020
Listening on http://localhost:5000
Document root is C:\projects\nette\kalendar-akci\www
Press Ctrl-C to quit.
[Wed Apr  1 20:40:56 2020] [::1]:52152 [200]: /
[Wed Apr  1 20:40:56 2020] [::1]:52153 [200]: /css/style.css
[Wed Apr  1 20:40:56 2020] [::1]:52154 [200]: /js/main.js
[Wed Apr  1 20:40:56 2020] [::1]:52155 [200]: /?_tracy_bar=js&v=2.7.3&XDEBUG_SESSION_STOP=1
[Wed Apr  1 20:41:07 2020] [::1]:52158 [200]: /favicon.ico

C:\projects\nette\kalendar-akci>
```

V prohlížeči zadáme adresu <http://localhost:5000> a uvidíme výchozí domovskou stránku.



6. Instalace externích knihoven

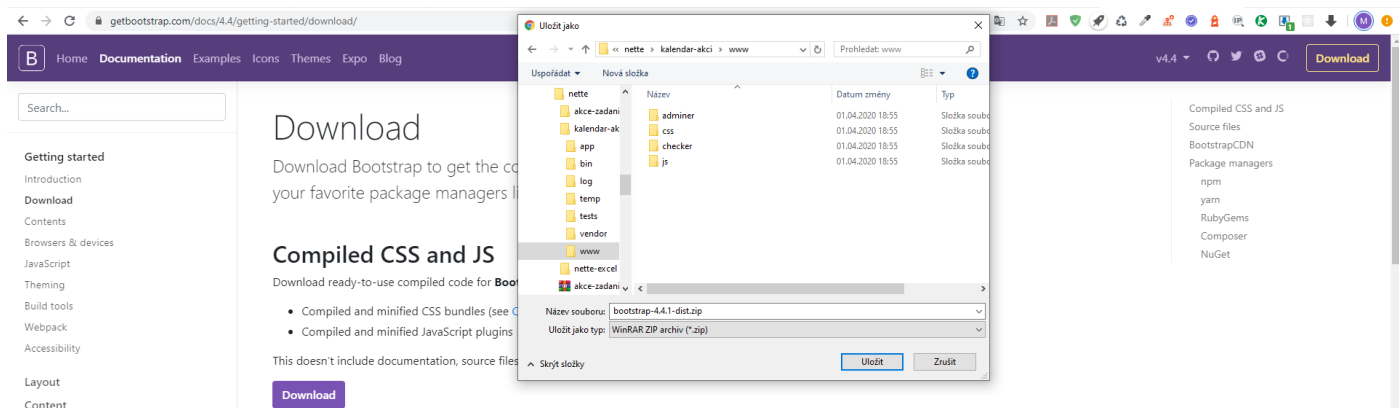
- Nástroj **composer** můžeme využít i k instalaci dalších PHP komponent určených pro Nette, které nejsou součástí základního sandboxu. Nejlepším zdrojem jsou stránky <https://componette.org/>
- Při výběru komponent je důležité sledovat, pro jakou verzi jsou určeny. Užitečnými informacemi jsou jistě také hodnocení a četnost využití dané komponenty, samozřejmě i datum poslední aktualizace.
- Příkladem užitečné komponenty může být **Datagrid** (<https://componette.org/contributte/datagrid/>). Pro jeho instalaci použijeme odkaz z <https://github.com/contributte/datagrid> na kompletní dokumentaci - <https://contributte.org/packages/contributte/datagrid/>

```
Příkazový řádek
C:\projects\nette\kalendar-akci>composer require ublaboo/datagrid
Using version ^6.2 for ublaboo/datagrid
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 5 installs, 0 updates, 0 removals
 - Installing symfony/polyfill-ctype (v1.15.0): Downloading (100%)
 - Installing symfony/inflector (v5.0.7): Downloading (100%)
 - Installing symfony/property-access (v5.0.7): Downloading (100%)
 - Installing contributte/application (v0.4.0): Downloading (100%)
 - Installing ublaboo/datagrid (v6.2.11): Downloading (100%)
symfony/property-access suggests installing psr/cache-implementation (To cache access methods.)
Writing lock file
Generating autoload files
C:\projects\nette\kalendar-akci>
```

- Ve formulářích se může hodit i DatePicker od Radka Dostála - <https://github.com/radekdostal/Nette-DateTimePicker>

```
Příkazový řádek
C:\projects\nette\kalendar-akci>composer require radekdostal/Nette-DateTimePicker
Using version ^3.0 for radekdostal/nette-datetimepicker
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
 - Installing radekdostal/nette-datetimepicker (3.0.0): Loading from cache
Writing lock file
Generating autoload files
C:\projects\nette\kalendar-akci>
```

- Pro úpravu stránek můžeme využít front-end framework Bootstrap 4. Z oficiálních stránek (<https://getbootstrap.com/docs/4.4/getting-started/download/>) si stáhneme kompilovanou verzi a rozbalíme v projektové složce www.



Název složky můžeme zjednodušit na bootstrap.

- Podobně bychom mohli do projektu přidat i další javaskriptové knihovny.

7. Úprava základní šablony stránek projektu

- Otevřeme projektovou složku **kalendar-akci** v některém z IDE nástrojů (NetBeans, VSCode ...).
- Poté otevřeme soubor **app\Presenters\templates\@layout.latte** a provedeme v kódu několik změn:

```
{**
 * @param string $basePath web base path
 * @param array $flashes flash messages
 **}

{import 'components/form.latte'}

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">

  <title>{ifset title}{include title|stripHtml} | {/ifset} Kalendář akcí</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="{ $basePath }/bootstrap/css/bootstrap.min.css">
  <link rel="stylesheet" href="{ $basePath }/css/style.css">
  {block head}{/block}
</head>

<body>
  <header class="jumbotron-fluid p5 bg-primary text-white">
    <h1 class="display-2 text-center">Kalendář akcí</h1>
  </header>
  <main>
    <div class=container>
      <div n:foreach="$flashes as $flash" n:class="alert, 'alert-' . $flash-
>type">{$flash->message}</div>

      {include content}
    </div>
  </main>
  <footer class="p-5 bg-secondary">
    <div class=container>
      <div class="row">
```

```

        <div class="col-sm-6"></div>
        <div class="col-sm-6">&copy; 2020, Marek Lučný, SŠPU Opava</div>
    </div>
</div>
</footer>
{block scripts}
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="https://nette.github.io/resources/js/3/netteForms.min.js"></script>
<script src="{ $basePath }/bootstrap/js/bootstrap.bundle.min.js"></script>
<script src="{ $basePath }/js/main.js"></script>
{/block}
</body>
</html>

```

- Značně zjednodušíme šablonu domovské stránky
app\Presenters\templates\Homepage\default.latte:

```

{* This is the welcome page, you can delete it *}

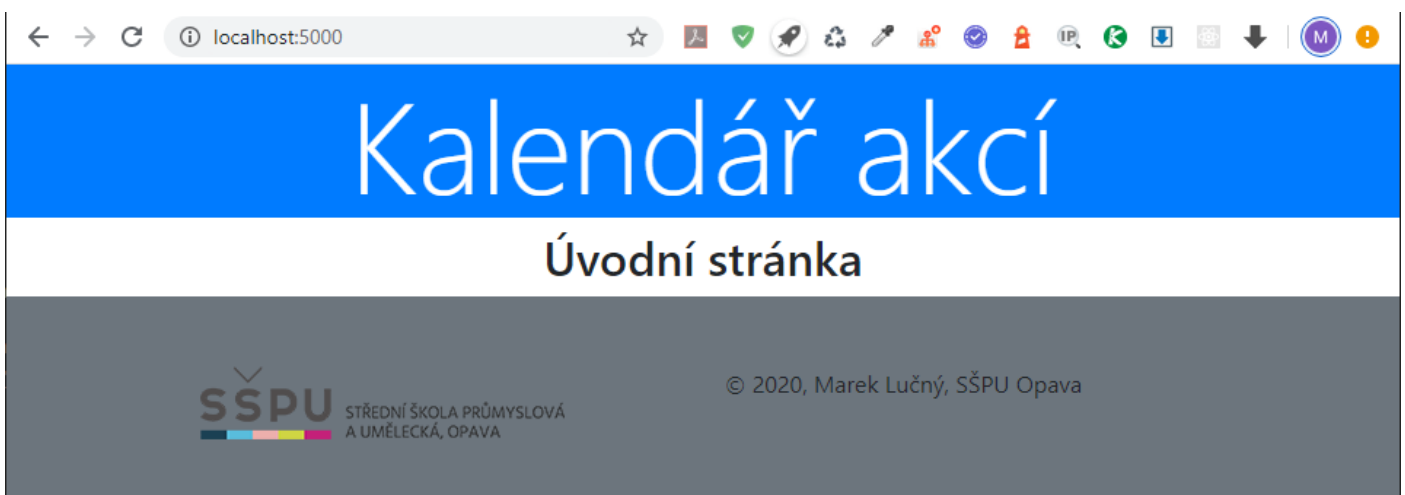
{block content}
    <h2 class="text-center">Úvodní stránka</h2>
{/block}

{block scripts}
{include parent}
{/block}

{block head}
{/block}

```

- Nakonec ještě smažeme všechna pravidla v souboru **www\css\style.css**.
- Po spuštění webového serveru si můžeme prohlédnout změněnou domovskou stránku:



8. Uložení sandboxu projektu na Github

- Založení nového repozitáře na github.com
- Vytvoření místního repozitáře v projektové složce:
C:\projects\nette\kalendar-akci>git init
- Konfigurace lokálního uživatele:


```
C:\projects\nette\kalendar-akci>git config --local user.user "sspu-opava"  
C:\projects\nette\kalendar-akci>git config --local user.email "itworkshop@sspu-  
opava.cz"
```

- Umístění všech souborů do staging area a první commit:

```
C:\projects\nette\kalendar-akci>git add .  
C:\projects\nette\kalendar-akci>git commit -m "My sandbox Nette-Framework"
```

- Připojení vzdáleného repozitáře a přenesení souborů na github.com:

```
C:\projects\nette\kalendar-akci>git remote add origin https://github.com/sspu-  
opava/kalendar-akci.git  
C:\projects\nette\kalendar-akci>git push -u origin master  
Username for 'https://github.com': sspu-opava  
Password for 'https://sspu-opava@github.com':  
Enumerating objects: 1096, done.  
Counting objects: 100% (1096/1096), done.  
Delta compression using up to 16 threads  
Compressing objects: 100% (1024/1024), done.  
Writing objects: 100% (1096/1096), 4.13 MiB | 1.04 MiB/s, done.  
Total 1096 (delta 154), reused 0 (delta 0)  
remote: Resolving deltas: 100% (154/154), done.  
To https://github.com/sspu-opava/kalendar-akci.git  
* [new branch]      master -> master  
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

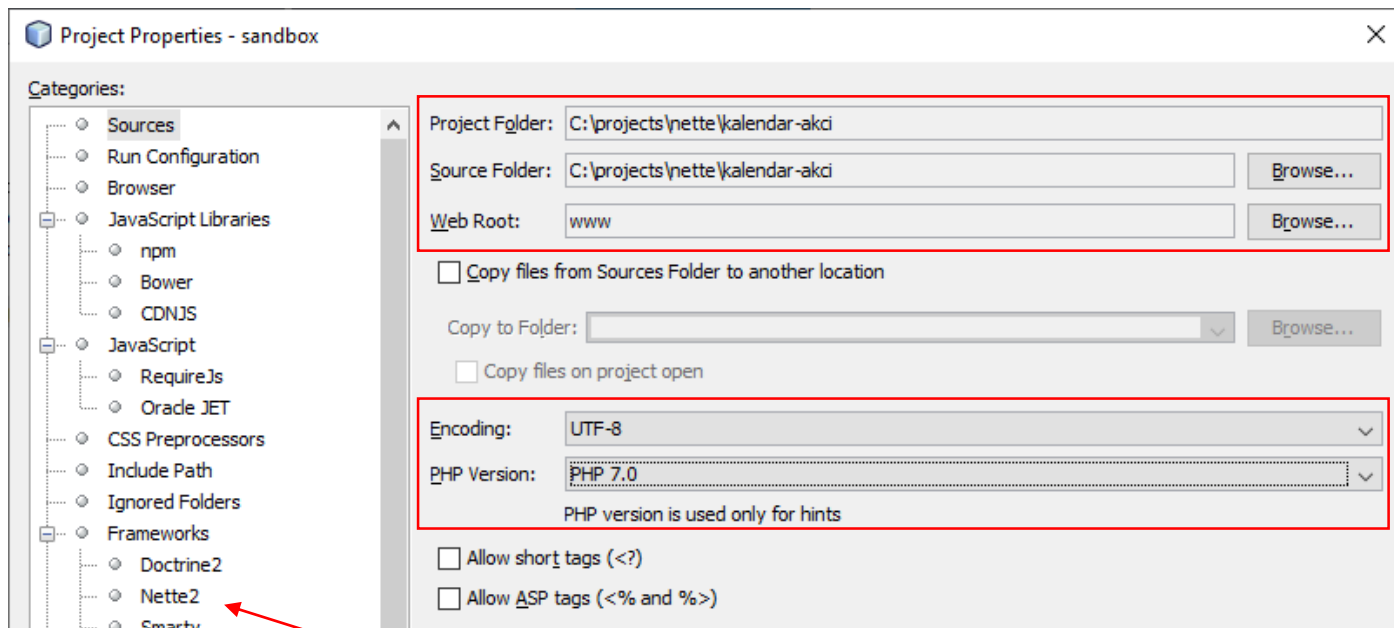

II. TVORBA JEDNODUCHÉ APLIKACE V NETTE FRAMEWORK

1. Založení projektu v NetBeans

IDE NetBeans nabízí dobrou podporu pro tvorbu webové aplikace v Nette, můžeme toho proto využít.

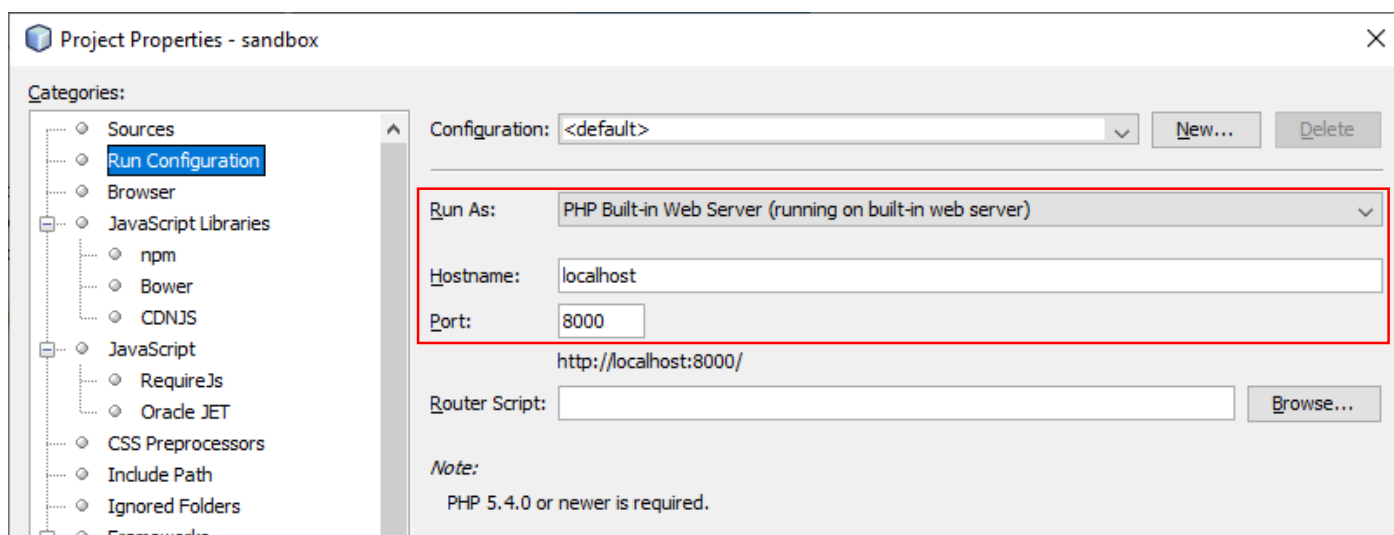
- Nejprve otevřeme připravený sandbox (složku kalendar-akci) jako projekt v NetBeans (File | Open project...).
- Poté postupně změníme nastavení projektu (File | Project properties (sandbox)).

Nejprve upřesníme údaje v nastavení Resources:

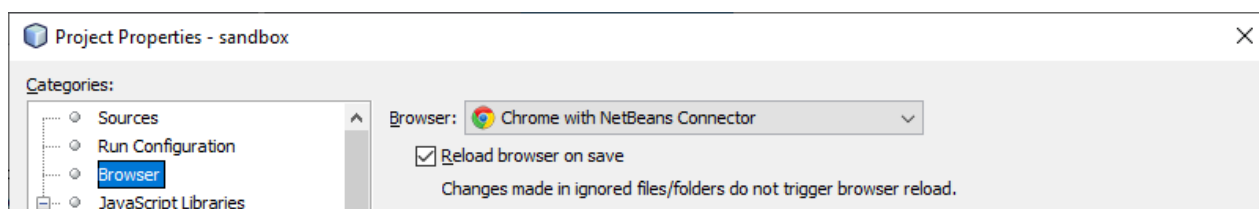


Na kartě Frameworks | Nette zaškrtneme podporu tohoto frameworku.

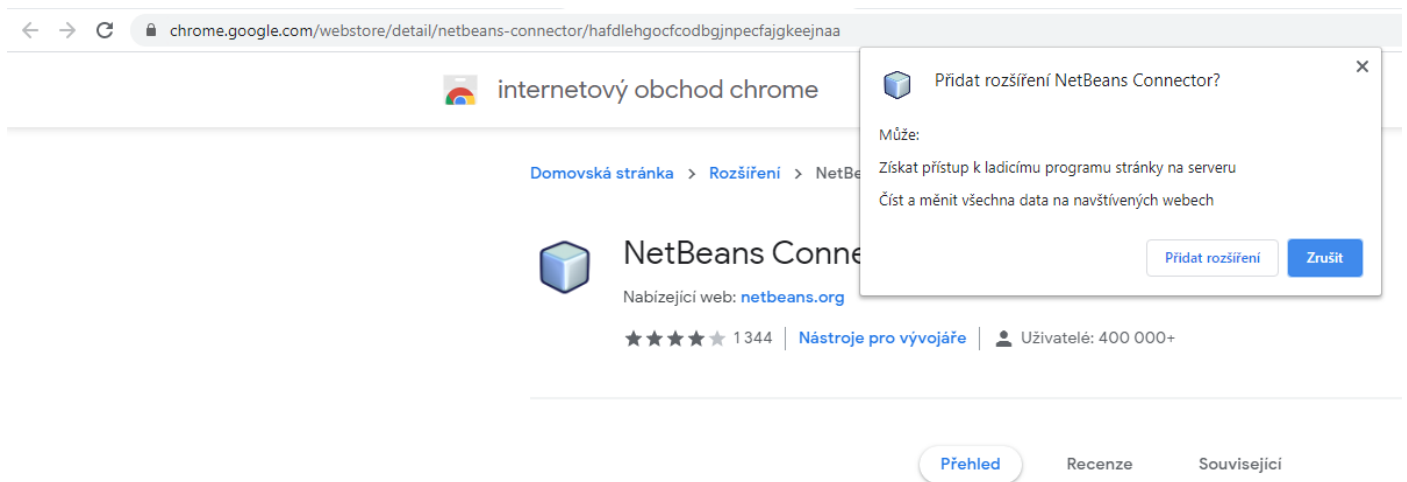
Na kartě Run Configuration zvolíme:



A pro nastavení browseru můžeme vybrat následující:



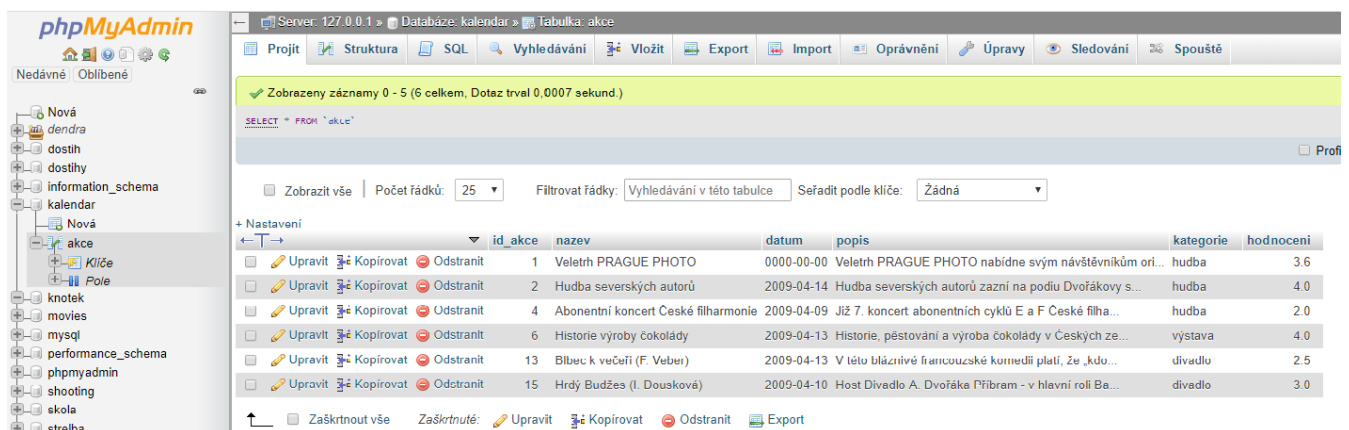
Podmínkou použití Chromu s NetBeans Connectorem je doinstalování příslušného rozšíření:



- Nyní můžeme zkusit spustit aplikaci (Run | Run Main Project, nebo klávesou F6). Pokud je vše nastaveno správně, měla by se v prohlížeči objevit úvodní stránka našeho webu a v pozadí běží webový server.

2. Základní konfigurace projektu

- Nejprve si například prostřednictvím aplikace PhpMyAdmin vytvoříme v MySQL novou databázi nazvanou kalendář. Do ní poté naimportujeme SQL dump (kalendář.sql).



- V základním konfiguračním souboru Nette `app\config\common.neon` provedeme změnu v nastavení databáze:

```
database:
  dsn: 'mysql:host=127.0.0.1;dbname=kalendář'
  user: 'root'
  password: ''
  options:
    lazy: yes
```

- Případně je možné v konfiguračním souboru pro develop fázi vypnout cache pro šablony v Latté, aby v některých situacích nebylo nutné mazat cache ručně. Provedeme následujícím nastavením v sekci services:

```
services:
  - App\Model\UserManager
  - App\Forms\FormFactory
  - App\Forms\SignInFormFactory
  - App\Forms\SignUpFormFactory
  router: App\Router\RouterFactory::createRouter
# Vypnutí cache pro Latté
```

```
latte.latteFactory:
  setup:
    - setTempDirectory(null)
```

- Nakonec ještě vytvoříme druhý commit v systému git a provedeme aktualizaci vzdáleného repozitáře.

Ještě před tím si všimněte změny, která nastala v souboru `.gitignore`:

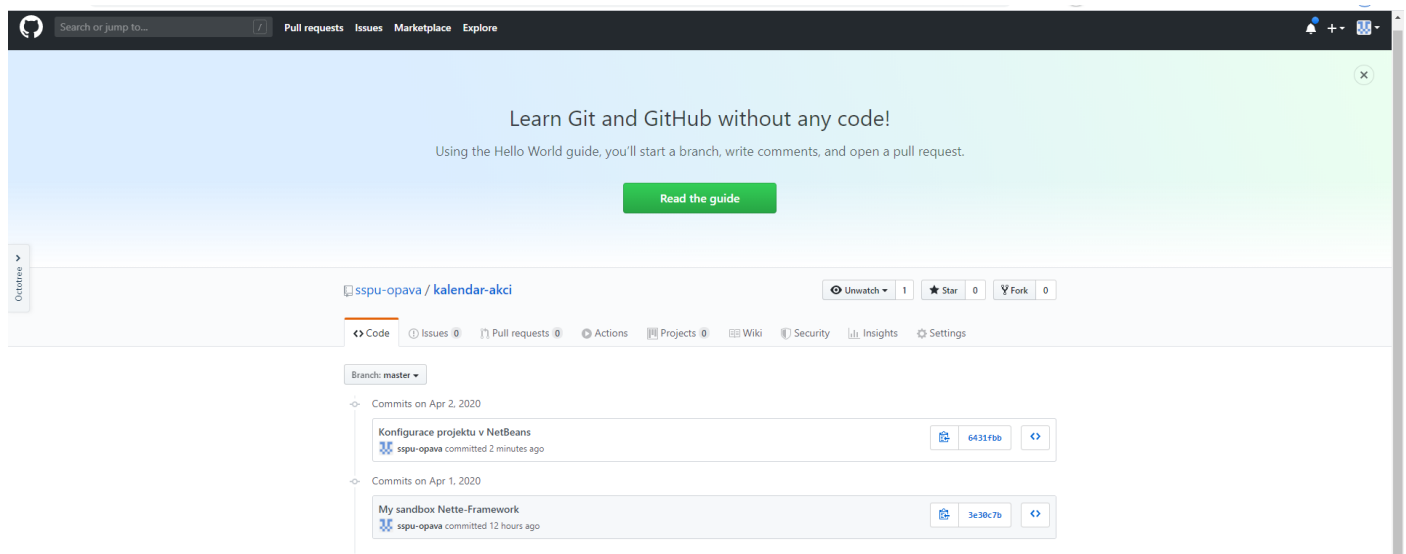
```
app/config/local.neon
/nbproject/private/
```

NetBeans do něj automaticky při založení projektu vložil řádek, který zajistí, že při ukládání do git repozitářů budou ignorovány změny v privátním nastavení projektu pro NetBeans.

```
C:\projects\nette\kalendar-akci>git add .
```

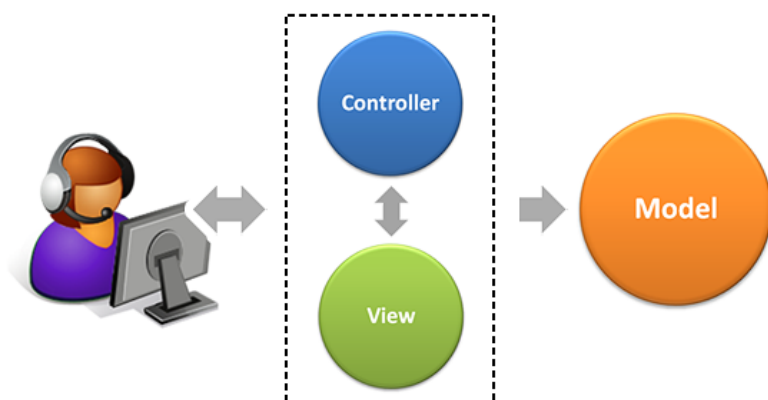
```
C:\projects\nette\kalendar-akci>git commit -m "Konfigurace projektu v NetBeans"
```

```
C:\projects\nette\kalendar-akci>git push
```



3. Vytvoření modelu aplikace

Řada moderních aplikací, a to nejen těch webových, vychází z obecné architektury označované **MVC** (Model - View - Controller):

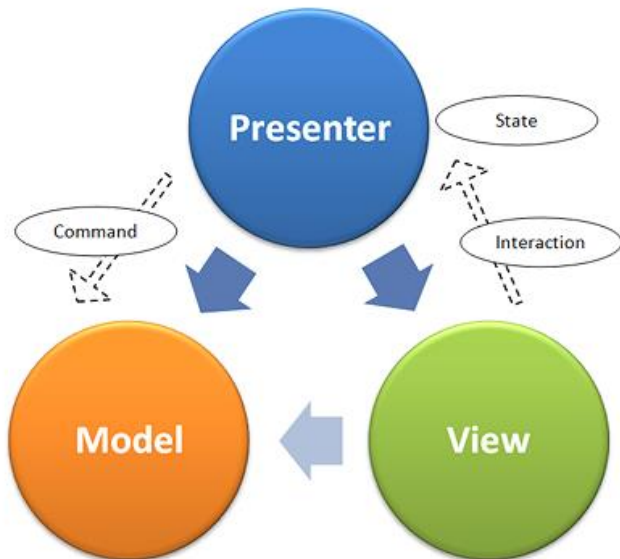


Podstatou této architektury je snaha oddělit od sebe kód uživatelského rozhraní od kódu aplikační logiky, který je označován jako **model**. Samotné rozhraní lze přitom ještě rozdělit na vrstvu, která obstarává výstup na monitor (**view**) a vrstvu zpracovávající vstupy z klávesnice nebo myši (**controller**).

V Nette používá modifikovanou architekturu **MVP** (Model - View - Presenter).

Presenter v Nette Frameworku má podobnou roli jako kontroler v MVC. Vybírá pohled (view) a předává mu model, nebo data z modelu. Udržuje stav persistentních proměnných. A především zpracovává reakce uživatele. Ty se dají rozdělit na tři typy:

- změna pohledu (např. obyčejná změna stránky)
- změna stavu (např. změna pořadí výpisu zpráv na stránce)
- příkaz modelu (např. vložení nové zprávy, její vymazání atd.)



Podrobněji o této problematice pojednává článek autora Nette *Davida Grudla*: <https://www.zdrojak.cz/clanky/nette-framework-mvc-mvp/>

Hlavním úkolem modelu je zpracovávat data. Ta mohou být uložena v souborech, v databázi, případně získávána prostřednictvím nějakého webového API (např. API Google, Facebook, map na Seznamu.cz atd.).

V našem příkladu používáme databázi MySQL (kalendář) a v ní zatím máme jednu tabulku (akce).

Vytvořme si proto nyní ve složce model vlastní třídu, která bude mít na starost správu dat pro tuto tabulku. Abychom si vše zjednodušili, můžeme si jako vzor vytvořit duplikát z už existující třídy **UserManager.php**.

Přejmenujeme ji na **AkceManager.php**, a změníme proto samozřejmě i název třídy - musí odpovídat názvu souboru.

Provedme nejprve promazání všeho nepotřebného a upravme konstanty tak, aby odpovídaly situaci v naší databázi, konkrétně v její tabulce akce (**app\Model\AkceManager.php**):

```
<?php

declare(strict_types=1);

namespace App\Model;

use Nette;

/**
 * Users management.
 */
final class AkceManager
{
    use Nette\SmartObject;
```

```

private const
    TABLE_NAME = 'akce',
    COLUMN_ID = 'id_akce',
    COLUMN_NAZEV = 'nazev',
    COLUMN_DATUM = 'datum',
    COLUMN_KATEGORIE = 'kategorie',
    COLUMN_POPOPIS = 'popis',
    COLUMN_HODNOCENI = 'hodnoceni';

/** @var Nette\Database\Context */
private $database;

public function __construct(Nette\Database\Context $database)
{
    $this->database = $database;
}
}

```

V konstruktoru je do privátní proměnné (atributu) `$database` injektován databázový modul Nette (**Nette\Database\Context**), jehož metody budeme následně v kódu využívat.

Nyní pod konstruktor zapíšeme metody, které budou zajišťovat zpracování dat v databázi:

```

public function __construct(Nette\Database\Context $database) {
    $this->database = $database;
}

/* Výpis všech záznamů z dané tabulky
 * $order - určuje řazení záznamů podle vybraného sloupce tabulky
 */
public function getAll($order = self::COLUMN_NAZEV) {
    return $this->database->table(self::TABLE_NAME)->order($order)->fetchAll();
}

/* Výpis jednoho záznamu podle id
 * $id - obsahuje odkaz na primární klíč vybraného záznamu
 */
public function getById($id) {
    return $this->database->table(self::TABLE_NAME)->get($id);
}

/* Vložení nového záznamu
 * $values - asociativní pole obsahující data vkládaného záznamu
 */
public function insert($values) {
    try
    {
        $this->database->table(self::TABLE_NAME)->insert($values);
        return true;
    }
    catch (Nette\Database\DriverException $e)
    {
        return false;
    }
}
}

```

```

/* Aktualizace záznamu
 * $id - odkaz na primární klíč aktualizovaného záznamu
 * $values - asociativní pole obsahující data aktualizovaného záznamu
 */
public function update($id, $values) {
    if ($zaznam = $this->getId($id)) return $zaznam->update($values);
    return false;
}
/* Vymazání záznamu
 * $id - odkaz na primární klíč odstraňovaného záznamu
 */
public function delete($id) {
    if ($zaznam = $this->getId($id)) return $zaznam->delete();
    return false;
}

```

Poznámky u jednotlivých metod vysvětlují, k čemu jsou určeny, a popisují i atributy, které jsou metodám předávány. Uvedené metody ošetřují nejčastěji používané operace v databázových aplikacích, které bývají označovány zkratkou **CRUD** (Create, Read, Update, Delete).

V metodě insert je použito ošetření tzv. **výjimek (exception)**. Blok **try** obsahuje příkazy, které mají být za normálních podmínek provedeny. Když však v některém z nich dojde k chybě ("výjimce"), program pokračuje blokem **catch**, který slouží k odchycení výjimek. V tomto konkrétním případě může nastat chyba během ukládání záznamu do databázové tabulky (např. kvůli nesprávné identifikaci sloupce). Když dojde k výjimce definované v třídě **Nette\Database\DriverException**, vrátí metoda hodnotu false.

V metodách update a delete zase podmínky nejprve zjistí, zda se podařilo v databázi najít záznam podle předaného id. Pokud tomu tak není, není možné záznam ani aktualizovat, ani smazat a výsledkem funkce je proto hodnota false.

4. Použití modelu v presenteru

Nejprve připravený model (tedy třídu **AkceManager**) zaregistrujeme jako novou službu v konfiguračním souboru **app\config\common.neon**:

```

services:
- App\Model\UserManager
- App\Model\AkceManager
- App\Forms\FormFactory
- App\Forms\SignInFormFactory
- App\Forms\SignUpFormFactory
router: App\Router\RouterFactory::createRouter

```

Díky tomu ji poté můžeme použít - injektovat - v libovolném modulu naší aplikace. Tato technika, nazývaná **Dependency Injection (DI)**, se v Nette hojně využívá. Podstatou Dependency Injection je odebrat třídám zodpovědnost za získávání objektů, které potřebují ke své činnosti (tzv. služeb) a místo toho jim služby předávat při vytváření. (Podrobněji zde: <https://doc.nette.org/cs/3.0/dependency-injection>)

Zcela konkrétně si to předvedeme při tvorbě presenteru AkcePresenter. Opět si vše zjednodušíme tím, že zkopírujeme už existující **HomepagePresenter** a po přejmenování souboru provedeme i změnu názvu třídy. Poté si připravíme základní kostru presenteru s metodou konstruktorem, v níž do privátní proměnné **\$akceManager** předáme parametr odkazující na model, tedy třídu **AkceManager**. Tím si pro tento presenter zpřístupníme všechny veřejné metody této třídy a můžeme díky nim pracovat s daty v databázi.

```

<?php
declare(strict_types = 1);

namespace App\Presenters;

/* Použití jmenného prostoru, v němž jsou uloženy třídy tvořící model aplikace */
use App\Model;

final class AkcePresenter extends BasePresenter {

    /* Privátní proměnná (atribut třídy), prostřednictvím které je "injektován" model
    AkceManager */
    private $akceManager;

    /* Metoda konstruktoru. Využití Dependency Injection - při vytváření objektu
    presenteru se připojí (injektuje) služba AkceManager */
    public function __construct(Model\AkceManager $akceManager) {
        $this->akceManager = $akceManager;
    }
}

```

Náš presenter je nyní připraven k tomu, abychom v něm vytvořili metody, které budou řešit jednotlivé akce **CRUD**. Vytvoříme si opět nejprve základní kostru těchto metod, abychom mohli otestovat jejich funkčnost:

```

/* Metoda konstruktoru. Využití Dependency Injection - při vytváření objektu
presenteru se připojí (injektuje) služba AkceManager */
public function __construct(Model\AkceManager $akceManager) {
    $this->akceManager = $akceManager;
}

/* Metoda zajistí přípravu a rendrování stránky se seznamem všech akcí */
public function renderList(): void {
}

/* Metoda zajistí přípravu a rendrování stránky pro detail jedné akce */
public function renderDetail(): void {
}

/* Metoda zajistí přípravu vložení nové akce a poté vyrendruje stránku s formulářem */
public function actionInsert(): void {
}

/* Metoda zajistí přípravu aktualizace akce a poté vyrendruje stránku s formulářem */
public function actionUpdate(): void {
}

/* Metoda provede smazání záznamu o akci a poté přesměruje zpět na seznam akcí */
public function actionDelete(): void {
    $this->redirect('list');
}

```

Názvy metod, které mají pouze vyrendrovat stránku na základě načtených dat, začínají slovem **render**. Metody, které provádějí nějakou změnu v datech (vložení, aktualizaci nebo smazání), začínají slovem **action**. I tyto metody nakonec

vyrenderují příslušnou stránku, pokud ještě předtím nedojde k přesměrování na jinou stránku. Tak je tomu u metody delete.

5. Vytvoření šablon jednotlivých stránek

Když se však nyní podíváte přes prohlížeč na běžící serverovou aplikaci, ukáže Laděnka chybu 404:

Nette\Application\BadRequestException #404

Page not found. Missing template 'C:\projects\nette\kalendar-akci\app\Presenters\templates\Akce\list.latte'.

Source file ▶

Call stack ▼

1. ...src\Application\UI\Presenter.php:481 source ▶ Nette\Application\UI\Component->error(arguments ▶)
2. ...src\Application\UI\Presenter.php:246 source ▶ Nette\Application\UI\Presenter->sendTemplate()
3. ...src\Application\Application.php:149 source ▶ Nette\Application\UI\Presenter->run(arguments ▶)
4. ...src\Application\Application.php:85 source ▶ Nette\Application\Application->processRequest(arguments ▶)
5. ...nette\kalendar-akci\www\index.php:10 source ▼ Nette\Application\Application->run()

```
1:  <?php
2:
3:  declare(strict_types=1);
4:
5:  require __DIR__ . '/../vendor/autoload.php';
6:
7:  App\Bootstrap::boot()
8:      ->createContainer()
9:      ->getByType(Nette\Application\Application::class)
10:     ->run();
11:
```

Důvodem je neexistence potřebných webových stránek, které se při každé akci mají vyrenderovat. V podsložce templates proto musíme nejprve založit složku nazvanou podle našeho prezenteru - Akce. V ní pak postupně vytvořit šablony latté pro každý požadavek na vykreslení stránky. Názvy šablon zadáváme podle názvů metod vytvořených v prezenteru - tedy **list.latte**, **detail.latte**, **insert.latte** a **update.latte**. Abychom pro začátek viděli aspoň stránku s nadpisem (a zbavili se chyby 404), vložíme do každé šablony nadpis h2 uvozující příslušnou stránku. Zde je příklad šablony list.latte:

KALENDAR-AKCI

- > Forms
- > Model
- AkceManager.php
- UserManager.php
- > Presenters
- > templates
- > Akce
 - detail.latte
 - insert.latte
 - list.latte
 - update.latte
- > components

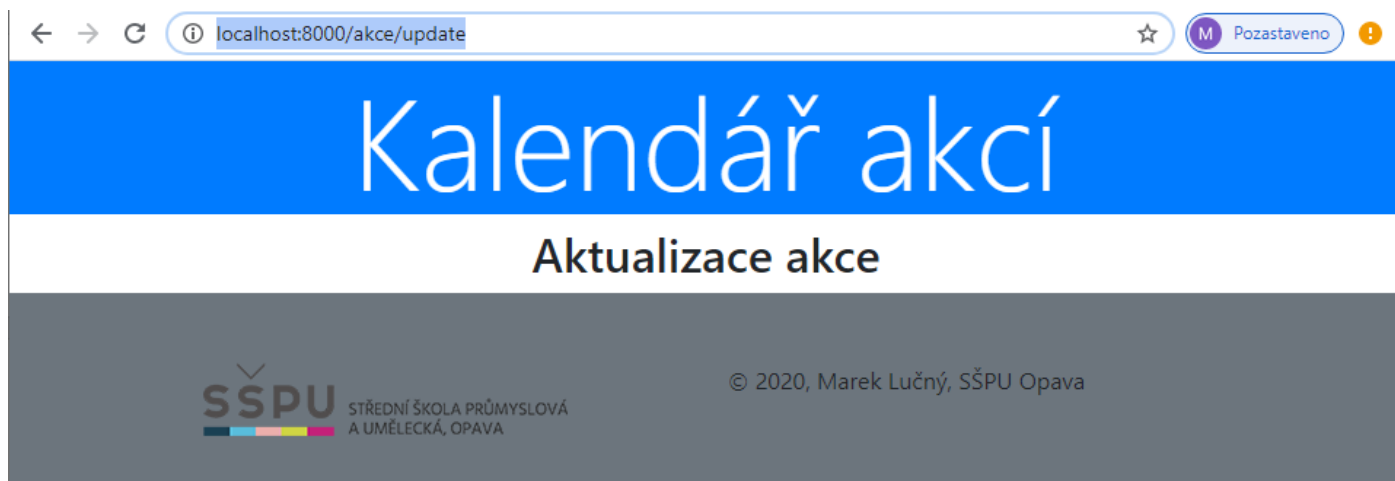
app > Presenters > templates > Akce > list.latte

```
1  { * Šablona pro seznam akcí *}
2  {block content}
3  <h2 class="text-center">Seznam akcí</h2>
4  {/block}
5
6  {block scripts}
7  {include parent}
8  {/block}
9
10
11 {block head}
12 {/block}
13
```

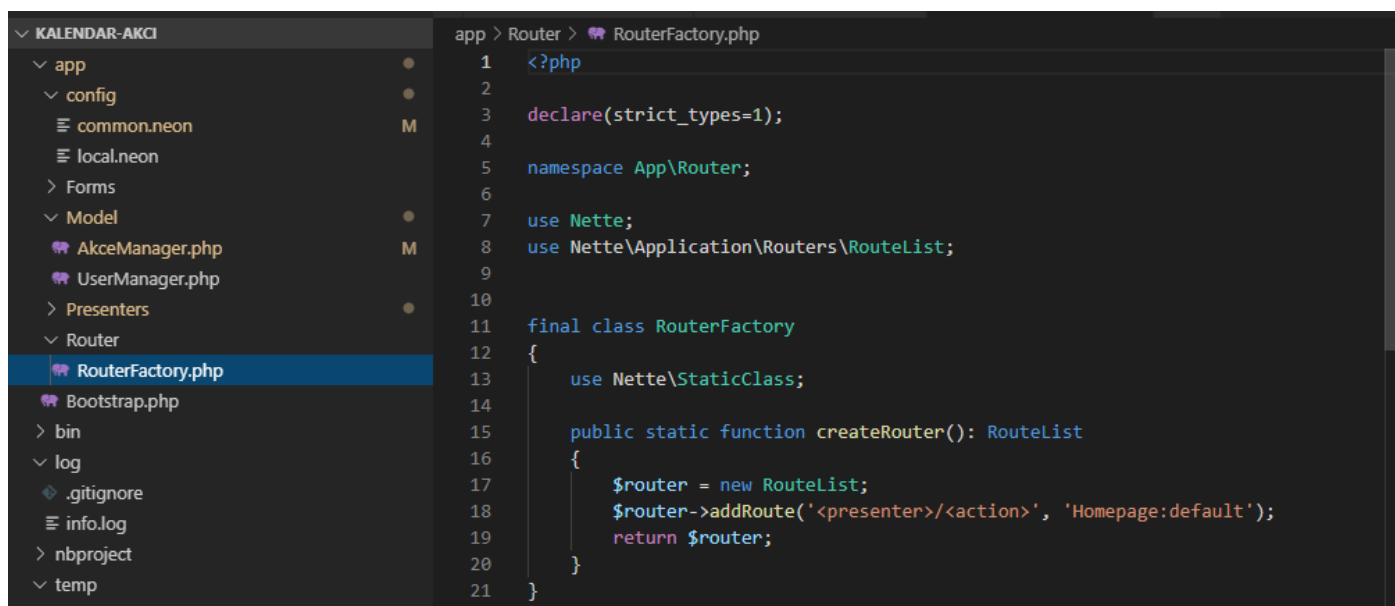
Stránku pro akci delete vytvářet nemusíme, protože tato metoda žádnou stránku nevyrenderuje, ale provede přesměrování na stránku list (tj. seznam akcí).

6. Směrování požadavků - routing

K ověření zobrazení jednotlivých stránek musíme zadat správné adresy do prohlížeče - např. URL požadavek <http://localhost:8000/akce/update> zobrazí tuto stránku:



Směrování požadavků na jednotlivé stránky (nebo akce webové aplikace) má v Nette na starost tzv. router. Ve složce **App\router** najdeme soubor **RouterFactory.php**, který můžeme použít ke konfiguraci routování v Nette:



Zatím zůstaneme u jednoduchého výchozího nastavení, které předpokládá, že v URL adrese požadavku se kromě adresy webového serveru (v našem případě <http://localhost:8000/>) se za lomítkem objevuje nejprve název presenteru (zde např. akce), a poté označení metody (např. update). Jako domovská složka serverové aplikace je nastavena stránka **Homepage:default**.

Další platné URL požadavky tedy jsou: <http://localhost:8000/akce/list>, <http://localhost:8000/akce/detail>, <http://localhost:8000/akce/insert> a <http://localhost:8000/akce/delete>

V našem presenteru však budeme jednotlivým metodám zadávat i parametry. V případě metod **detail**, **update** a **delete** to bude parametr **id**, do něhož bude uloženo id akce (primární klíč), kterou má metoda zpracovat. Metodě **list** zase můžeme předat parametr **order**, který bude rozhodovat o způsobu řazení záznamů v seznamu. Parametr **order** může obsahovat i výchozí hodnotu - například **'datum desc'** (preferujeme řazení podle data sestupně, tedy od nejnovějších akcí):

```
/* Metoda zajistí přípravu a rendrování stránky se seznamem všech akcí */
public function renderList($order = 'datum DESC'): void {
```

```

}

/* Metoda zajistí přípravu a rendrování stránky pro detail jedné akce */

public function renderDetail($id): void {

}

...

...

/* Metoda zajistí přípravu aktualizace akce a poté vyrenderuje stránku s formulářem */

public function actionUpdate($id): void {

}

/* Metoda provede smazání záznamu o akci a poté přesměruje zpět na seznam akcí */

public function actionDelete($id): void {
    $this->redirect('list');
}

```

Parametry požadavku mohou být do URL adresy zadány za otazník (např. <http://localhost:8000/akce/update?id=21>). V současných aplikacích se často setkáváme s použitím tzv. "pěkných URL", kdy se hodnota id předává v adrese za lomítkem (např. <http://localhost:8000/akce/update/21>). K tomu je ale potřeba provést drobnou úpravu v souboru **RouterFactory.php**:

```

public static function createRouter(): RouteList
{
    $router = new RouteList;
    $router->addRoute('<presenter>/<action>[/<id>]', 'Homepage:default');
    return $router;
}

```

Podrobněji se o možnostech routování v Nette dozvíte zde: <https://doc.nette.org/cs/3.0/routing>

7. Ladicí prostředky v Nette

Nette díky modulu **Tracy** (čeští programátoři používají něžný název "Laděnka") nabízí řadu prostředků jak v procesu ladění aplikace získávat důležité informace o jejím stavu. Podrobné informace o ladění v Nette i několik užitečných příkladů naleznete zde: <https://tracy.nette.org/cs/guide>

Ukážeme si alespoň základní možnosti.

V situaci, kdy uděláme chybu v kódu aplikace, se objeví okno Laděnky.

V tomto případě došlo k nekorektnímu pojmenování třídy presenteru malým písmenem. Pod jednotlivými odkazy okna nalezneme řadu užitečných informací týkajících se například stavu proměnných (**Variables**), způsobených výjimek (**Exception**), stavu aplikace Nette (**Nette Application**), konfiguraci webového serveru (**Environment**) i detailní výpisy hlaviček požadavku (**HTTP request**) i odpovědi (**HTTP response**) protokolu HTTP.

Vpravo dole se nachází lišta Tracy, která zobrazuje čas načtení stránky, data z kontejneru aplikace (**DIC** - *Dependency Injection Container*), aktuální routu (např. **Akce:detail**), SQL dotazy (pokud nějaké byly vykonány) a případně další důležité informace, které si do této lišty v konfiguračním souboru nasměrujeme.

← → ↻ ⓘ localhost:8000/akce/list

User Warning

Case mismatch on presenter name 'Akce', correct name is 'akce'.

Source file ▶

Call stack ▶

Variables ▶

Exception ▶

Nette Application ▶

Environment ▶

HTTP request ▶

HTTP response ▶

Please support Tracy via a donation ♥
Report generated at 2020/04/03 18:46:15
PHP 7.3.7
PHP 7.3.7 Development Server
Tracy 2.7.3

Container_b199f0a220

database.default.context	yes	Nette\Database\Context
database.default.conventions	yes	Nette\Database\Conventions\DiscoveredConventions
database.default.structure	yes	Nette\Database\Structure
http.request	yes	Nette\Http\Request #9de9 ▾ method private => "GET" url private => Nette\Http\UrlScript #9640 ▶ post private => array () files private => array () cookies private => array (3) ▾ csrfToken => "Y08gTHbTg3WTwOh3Zqcpc0i5JEbnchqGdBC16I1Y02g" PHPSESSID => "9bbod2iadfv40q3ld66nsh1ntp" nette-samesite => "1" headers private => array (13) ▶ remoteAddress private => "::1" remoteHost private => null rawBodyCallback private => Closure #8586 ▾ file => "C:\projects\nette\kalendar-akci\vendor\nette\htt line => 79 variables => array () parameters => ""
http.requestFactory	yes	Nette\Http\RequestFactory #09a0 ▶
http.response	yes	Nette\Http\Response #0986 ▶
latte.latteFactory	yes	Nette\Bridges\Application\Latte\LatteFactory

TRACY 2276.5 ms DIC 952.1 ms Akce:list

Nyní můžeme aktualizovat soubor AkcePresenter, tak abychom získali různé ladicí informace:

```
<?php

declare(strict_types = 1);

namespace App\Presenters;

/* Použití jmenného prostoru, v němž jsou uloženy třídy tvořící model aplikace */
use App\Model;

/* Připojení knihovny Debugger, která je součástí Laděnký (Tracy) */
use Tracy\Debugger;

final class AkcePresenter extends BasePresenter {
    /* Privátní proměnná (atribut třídy), prostřednictvím které je "injektován" model
    AkceManager */

    private $akceManager;

    /* Metoda konstruktoru. Využití Dependency Injection - při vytváření objektu
    presenteru se připojí (injektuje) služba AkceManager */
```

```

public function __construct(Model\AkceManager $akceManager) {
    $this->akceManager = $akceManager;
}

/* Metoda zajistí přípravu a rendrování stránky se seznamem všech akcí */

public function renderList($order = 'datum DESC'): void {
    /* Na začátek stránky budou vypsány ladicí informace - o proměnné order */
    Debugger::dump('Proměnná order: ', $order);
}

/* Metoda zajistí přípravu a rendrování stránky pro detail jedné akce */

public function renderDetail($id): void {
    /* Do okna ve spodní liště bude vložena ladicí informace o stavu proměnné $id */
    Debugger::barDump('Proměnná id: ', $id);
}

/* Metoda zajistí přípravu vložení nové akce a poté vyrenderuje stránku s formulářem */

public function actionInsert(): void {
    Debugger::log('Vložen nový záznam');
}

/* Metoda zajistí přípravu aktualizace akce a poté vyrenderuje stránku s formulářem */

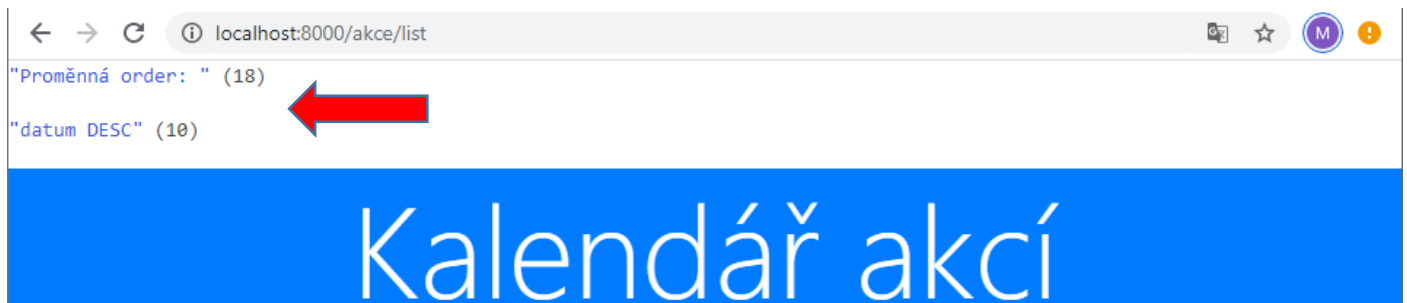
public function actionUpdate($id): void {
    Debugger::log('Aktualizován záznam '.$id);
}

/* Metoda provede smazání záznamu o akci a poté přesměruje zpět na seznam akcí */

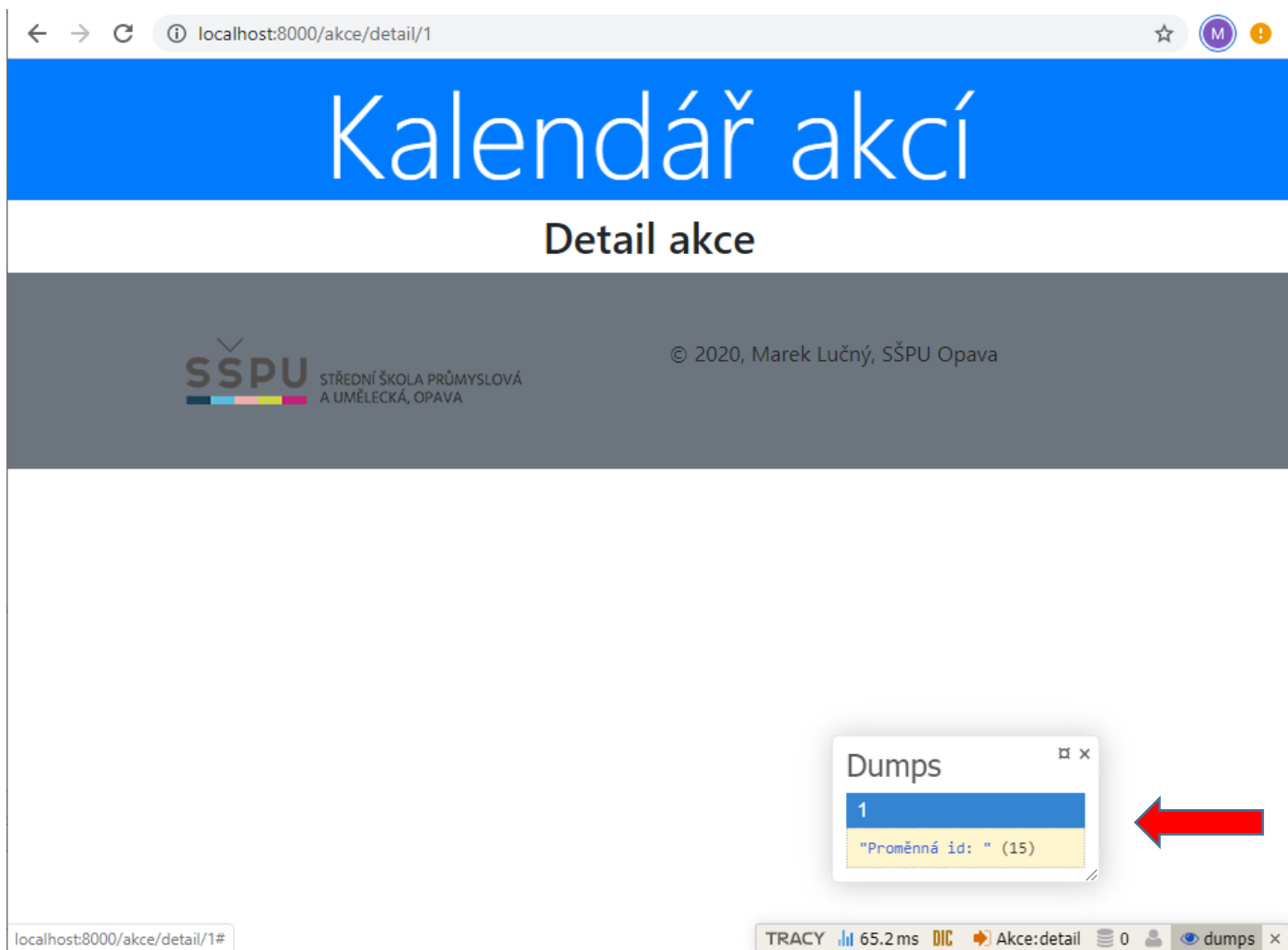
public function actionDelete($id): void {
    Debugger::log('Odstraněn záznam '.$id);
    $this->redirect('list');
}
}

```

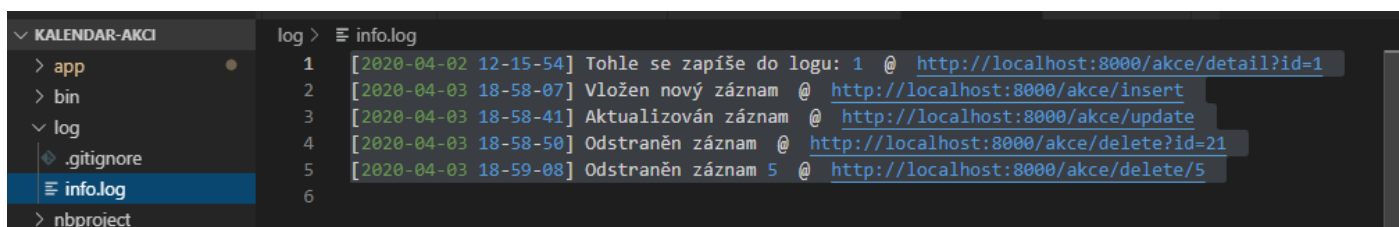
- Nejprve musíme připojit třídu **Tracy\Debugger**, která nabízí různé ladicí prostředky.
- Metoda **Debugger::dump()** vypisuje informace o sledovaných objektech přímo na stránku.



- Metoda **Debugger::barDump()** vypisuje informace o sledovaných objektech do okna v liště Tracy.



- Metoda **Debugger:log()** vypisuje informace o sledovaných objektech do souboru **log\info.log**:



Pomocí gitu vytvoříme další commit aplikace do lokálního repozitáře i vzdáleného repozitáře na GitHubu.

8. Vytvoření stránek s výpisy informací

Nejprve v souboru **AkcePresenter** doplníme metody **renderList()** a **renderDetail()**:

```
/* Metoda zajistí přípravu a rendrování stránky se seznamem všech akcí */

public function renderList($order = 'datum DESC'): void {
    /* Na začátek stránky budou vypsány ladicí informace - o proměnné order */
    // Debugger:dump('Proměnná order: ', $order);
    /* Příslušné šabloně (list.latte) bude předána proměnná $akceList (pole objektů),
    do níž budou díky metodě getAll() uloženy všechny záznamy o akcích
    uspořádané podle proměnné $order */
    $this->template->akceList = $this->akceManager->getAll($order);
}
```

```

/* Metoda zajistí přípravu a rendrování stránky pro detail jedné akce */

public function renderDetail($id): void {
    /* Do okna ve spodní liště bude vložena ladicí informace o stavu proměnné $id */
    // Debugger::barDump('Proměnná id: ', $id);
    /* Příslušné šablone (detail.latte) bude předána proměnná $akce,
    do níž bude díky metodě getById() uložen záznam (objekt) o akci
    vybrané podle proměnné $id */
    $this->template->akce = $this->akceManager->getById($id);
}

```

V obou případech používáme některou z připravených metod v modelu AkceManager k tomu, abychom získali požadovaná data z databáze a připravili je příslušné šablone v podobě proměnných. Na šablonu odkazujeme pomocí `$this->template->[nějaká proměnná]`.

Nyní můžeme doplnit obě šablony tak, abychom v nich zobrazili příslušná data.

app\Presenters\templates\Akce\list.latte

```

{* Šablona pro seznam akcí *}
{block content}
<h2 class="text-center bg-secondary text-white p-3">Seznam akcí</h2>
<table class="table table-striped table-inverse table-responsive">
    <thead class="thead-inverse">
        <tr>
            {* Odkazy, pomocí kterých zajišťujeme změnu řazení záznamů podle hodnoty
            předávané do parametru $order *}
            <th><a n:href="list 'nazev'">Název</a></th>
            <th><a n:href="list 'datum DESC'">Datum</a></th>
            <th><a n:href="list 'kategorie'">Kategorie</a></th>
            <th><a n:href="list 'hodnoceni DESC'">Hodnocení</a></th>
            <th>Operace</th>
        </tr>
    </thead>
    <tbody>
        {* Pro všechny prvky pole $akceList se opakuje výpis řádku tabulky. Aktuální
        hodnoty pro daný záznam obsahuje
        proměnná-objekt $akce. Ke každému sloupci tabulky přistupujeme jako k
        atributu objektu: např. $akce->nazev *}
        {foreach $akceList as $akce}
        <tr>
            <td scope="row"><a n:href="detail $akce->id_akce">{$akce->nazev}</a></td>
            <td>{$akce->datum|date:'j. n. Y'}</td>
            <td>{$akce->kategorie}</td>
            <td>{$akce->hodnoceni}</td>
            {* Odkazy, které směřují požadavky na metody update nebo delete v
            presenteru Akce.
            Jako parametr je v obou případech předávána jedinečná hodnota ze sloupce
            id_akce (primární klíč). *}
            <td><a n:href="Akce:update $akce->id_akce" class="btn btn-
            success">Editovat</a> <a n:href="delete $akce->id_akce" class="btn btn-
            danger">Smazat</a></td>
        </tr>
        {/foreach}
    </tbody>
</table>

```



```

        {/foreach}
    </tbody>
</table>
{* Odkaz, který směřuje požadavek na metodu insert v presenteru Akce. *}
<p><a n:href="Akce:insert" class="btn btn-primary">Nová akce</a></p>
{/block}

{block scripts}
{include parent}
{/block}

{block head}
{/block}

```

Šablonovací systém Latté (podrobně zde: <https://latte.nette.org/cs/>), podobně jako jiné podobné šablonovací systémy, umožňuje kombinovat kód HTML s programovým kódem, v tomto případě v jazyce PHP. Programový kód vkládáme nejčastěji do tzv. maker (nebo direktiv), které jsou uvozovány složenými závorkami - více <https://latte.nette.org/cs/guide#toc-tagy>.

Podstatou šablony **list.latte** je modře zvýrazněná část kódu, v níž příkaz **foreach** zajišťuje postupný výpis všech objektů, které jsou obsaženy v proměnné **\$akceList**. Jednotlivé údaje (sloupce databázové tabulky) vypisujeme do buněk v rámci řádku tabulky. Každý záznam představuje samostatný objekt uložený pro danou chvíli do proměnné **\$akce**. Každý údaj (sloupec tabulky) je atributem tohoto objektu a v PHP je přístupný prostřednictvím symbolu ->

Části kódu zvýrazněné hnědou barvou obsahují odkazy s tzv. n-atributy (n:href) - více <https://latte.nette.org/cs/guide#toc-n-atributy>. Pomocí odkazů můžeme provádět přesměrování na různé části webové aplikace.

Pro odkazy n:href platí následující syntax:

```
[Presenter:]action [,] [arg1] [, arg2] [, ...]
```

Proto například odkaz `<a n:href="Akce:update $akce->id_akce">Editovat` provede přesměrování na presenter **Akce**, metodu **update** a předá jí parametr **\$akce->id_akce**

```

public function actionUpdate($id): void {
    Debugger::log('Aktualizován záznam '.$id);
}

```

Název presenteru nemusí být uveden v případě, když odkazujeme přímo z kontextu tohoto presenteru (což by platilo i pro tento případ). Proto jsou odkazy v záhlaví tabulky o něco kratší:

```
<th><a n:href="list 'datum DESC'">Datum</a></th>
```

Zde se provede přesměrování na metodu **list** v aktuálním presenteru (Akce) a jako parametr (\$order) se předá řetězec **'datum DESC'**.

Červenou barvou je v kódu zvýrazněn tzv. filtr (více <https://latte.nette.org/cs/filters>). Filtry nějakým způsobem upravují vložený údaj. V tomto konkrétním případě `{ $akce->datum | date: 'j. n. Y' }` filtr zajistí úpravu datového údaje do běžně používaného tvaru den. měsíc. rok.

Dva filtry jsou použity i v šabloně **detail.latte**.

app\Presenters\templates\Akce\detail.latte

```
{* Šablona pro detail akce *}
{block content}
<h2 class="text-center bg-secondary text-white p-3">Detail akce</h2>
<h3>{$akce->nazev|upper}</h3>
<p class="text-secondary">Kategorie: {$akce->kategorie}, datum konání: {$akce->datum|date:'j. n. Y'}</p>
<p>{$akce->popis}</p>
<p>Hodnocení: {$akce->hodnoceni}</p>
<hr>
<p><a n:href="list">Seznam akcí</a></p>
{/block}

{block scripts}
{include parent}
{/block}

{block head}
{/block}
```

Nyní provedeme další commit do repozitářů.

9. Tvorba formuláře

Jednou z výhod použití webových frameworků je významné urychlení tvorby bezpečných formulářů. Nette v tomto ohledu nezůstává pozadu a nabízí vývojářům knihovnu **Nette Forms** i podrobný návod jak s ní pracovat. Na stránce <https://doc.nette.org/cs/3.0/forms> najdeme hned na začátku jednoduchý příklad jak do presenteru začlenit registrační formulář. Můžeme ho využít i v naší úloze a upravit ho pro účely vkládání informací o akcích. Doporučuji začít s minimálním počtem prvků a postupně do formuláře přidávat další komponenty, abychom vše mohli průběžně sledovat i testovat.

Tady je "minimalistická" verze formuláře začleněného do souboru **AkcePresenter**:

Nejprve na začátku presenteru připojíme knihovnu UI, jejíž součástí je i třída **Form**.

```
/* Použití jmenného prostoru pro přístup ke komponentám UI - uživatelského rozhraní (formuláře) */
use Nette\Application\UI;
```

Poté do třídy **AkcePresenter** vložíme dvě nové metody:

```
protected function createComponentAkceForm(): UI\Form {
    $form = new UI\Form;
    $form->addText('nazev', 'Název akce:');
    $form->addSubmit('submit', 'Potvrdit');
    $form->onSuccess[] = [$this, 'akceFormSucceeded'];
    return $form;
}

// volá se po úspěšném odeslání formuláře
public function akceFormSucceeded(UI\Form $form, \stdClass $values): void {
    Debugger::barDump($values);
    $this->flashMessage('Formulář byl odeslán','success');
    $this->redirect('Akce:list');
}
```

Úkolem první metody **createComponentAkceForm()** je vytvořit komponentu ("továrnu") s objektem formuláře. Objekt je vytvořen jako instance třídy **Form** a uložen do proměnné **\$form**. Do tohoto objektu jsou postupně přidávány jednotlivé formulářové prvky pomocí metod, které začínají slovem **add** (např. **addText()**, **addTextArea()**, **addSelect()**, **addSubmit()** apod.). Do atributu **onSuccess** (typ pole) jsou poté uloženy informace o metodě, která bude zavolána po úspěšném vyplnění a odeslání formuláře. Ukazatel **\$this** odkazuje na "tento" presenter (tj. **AkcePresenter**) a ošetřující metoda se nazývá **akceFormSucceeded**.

Metoda **akceFormSucceeded()** přijímá ke zpracování dva parametry - objekt formuláře (**\$form**) a asociativní pole **\$values** obsahující data zadaná uživatelem do jednotlivých formulářových prvků. Během postupného vývoje formuláře je dobré sledovat, co obě proměnné obsahují, proto použijeme spodní lištu Laděnky a proměnnou **\$values** si zde budeme vypisovat (**Debugger::barDump(\$values)**).

Příkaz **\$this->flashMessage('Formulář byl odeslán','success')** zase zajistí zobrazení "bleskových" informačních zpráv hned pod záhlavím webové stránky. Druhý parametr rozhoduje o barvě pozadí bloku (alertu), v němž se bleskové zprávy objevují. Můžeme k tomu využít běžné barevné schéma frameworku Bootstrap.

Nakonec dojde k přesměrování na stránku se seznamem akcí.

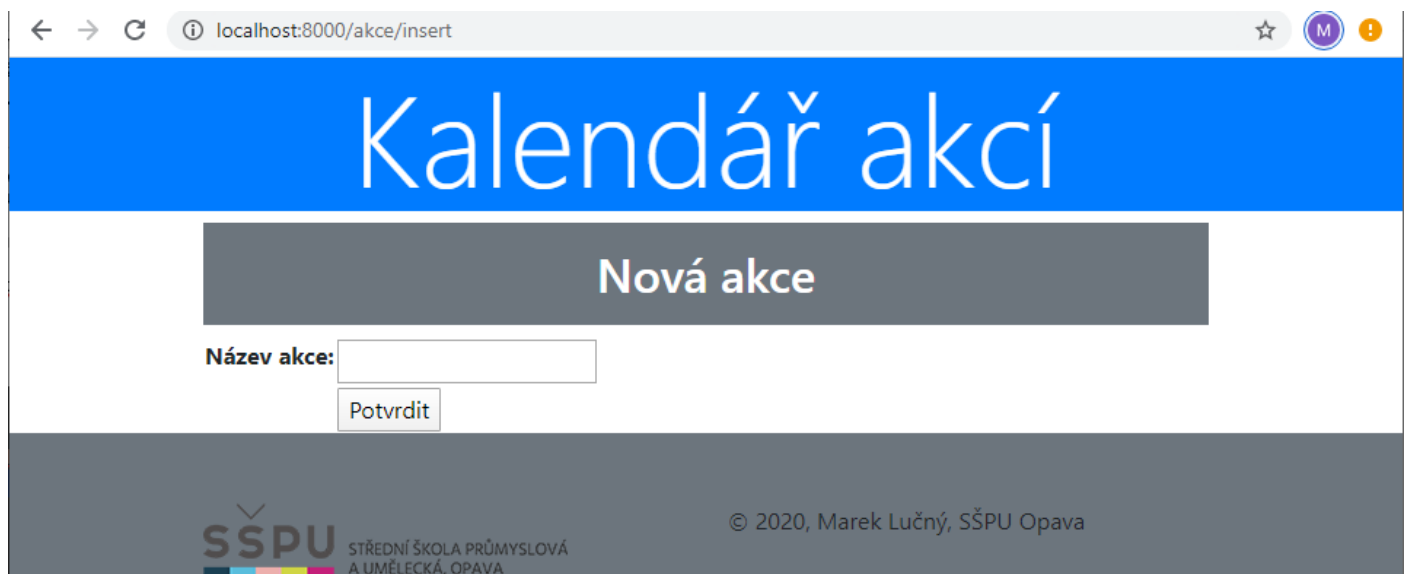
Nyní minimální formulář začleníme do šablon **insert.latte** a **update.latte**. V obou případech stačí použít jednoduché makro **{control akceForm}**

```
{* Šablona pro vložení nové akce *}
{block content}
<h2 class="text-center bg-secondary text-white p-3">Nová akce</h2>
{control akceForm}
{/block}

{block scripts}
{include parent}
{/block}

{block head}
{/block}
```

Po kliknutí na tlačítko Nová akce na stránce Seznam akcí (list.latte) by se měl objevit první formulář:



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/akce/insert'. The page has a blue header with the text 'Kalendář akcí'. Below the header is a grey box with the text 'Nová akce'. Underneath this box is a form with a label 'Název akce:' followed by a text input field. Below the input field is a button labeled 'Potvrdit'. At the bottom of the page, there is a footer with the logo 'SŠPU' and the text 'STŘEDNÍ ŠKOLA PRŮMYSLOVÁ A UMĚLECKÁ, OPAVA' on the left, and '© 2020, Marek Lučný, SŠPU Opava' on the right.

Můžeme zkusit zadat nějaký pokusný text do vstupního pole a odeslat tlačítkem Potvrdit.

Měla by se nám zobrazit stránka Seznam akcí a uvidíme v ní jednak bleskovou zprávu (flashMessage), jednak obsah proměnné \$values (Dumps).

The screenshot shows a web browser at localhost:8000/akce/list?_fid=6wvi. The page has a blue header with the title "Kalendář akcí". Below the header, a green message box says "Akce byla uložena", indicated by a red arrow. The main content is a table titled "Seznam akcí" with columns: Název, Datum, Kategorie, Hodnocení, and Operace. The table contains three rows of "Testovací akce" and one row of "Hudba severských autorů". Each row has "Editovat" and "Smazat" buttons. A red arrow points to the "Editovat" button of the third "Testovací akce" row. A "Dumps" window is open, showing the variable "nazev" with the value "Pokus". The bottom status bar shows performance metrics for "Akce:list" and "Akce:insert".

Název	Datum	Kategorie	Hodnocení	Operace
Testovací akce	1. 4. 2020	výstava	3.5	Editovat Smazat
Testovací akce	1. 4. 2020	výstava	3.5	Editovat Smazat
Testovací akce	1. 4. 2020	výstava	3.5	Editovat Smazat
Hudba severských autorů	14. 4. 2009	hudba		

Ted' už můžeme do formuláře přidávat další prvky a vhodně nastavovat jejich vlastnosti:

```
/* Komponenta ("továrna") formuláře */

protected function createComponentAkceForm(): UI\Form {
    $form = new UI\Form;

    $form->addText('nazev', 'Název akce:')
        /* Pravidlo je definováno pomocí regulárního výrazu */
        ->addRule(UI\Form::PATTERN, 'Musí obsahovat aspoň 5 znaků', '{5,}')
        /* Údaj je povinný */
        ->setRequired(true);

    $form->addText('datum', 'Datum konání akce:')
        /* Nastavení typu vstupního pole */
        ->setHtmlType('date')
        ->setRequired(true);

    $form->addTextArea('popis', 'Stručný popis:')
        /* Vložení atributu do formulářového prvku */
        ->setHtmlAttribute('rows', '6')
        ->setRequired(true);

    /* Asociativní pole definuje předvolby, které je možné použít u prvků
```

```

    * select nebo radio */
    $kategorie = [
        'divadlo' => 'divadlo',
        'hudba' => 'hudba',
        'kino' => 'kino',
        'výstava' => 'výstava'
    ];
    $form->addSelect('kategorie', 'Kategorie:', $kategorie);

    $form->addText('hodnoceni', 'Hodnocení:')
        ->setHtmlType('number')
        ->setHtmlAttribute('min', '1.0')
        ->setHtmlAttribute('max', '5.0')
        ->setHtmlAttribute('step', '0.1')
        ->setHtmlAttribute('title', 'Zadejte hodnocení v rozsahu 1.0 až 5.0')
        /* Validační pravidlo nastavuje rozsah platných hodnot */
        ->addRule(UI\Form::RANGE, 'Hodnocení musí být v rozsahu od 1.0 do 5.0',
[1.0, 5.0]);

    $form->addSubmit('submit', 'Potvrdit');

    $form->onSuccess[] = [$this, 'akceFormSucceeded'];
    return $form;
}

```

Podrobný přehled o formulářových prvcích v Nette a možnostech jejich nastavení najdete zde:

<https://doc.nette.org/cs/3.0/form-fields>. Pomocí komentářů v kódu se můžete lépe zorientovat, jak je formulář vytvořen. Prvky do formuláře přidávejte postupně a kontrolujte jejich chování na stránce, kde je formulář zobrazen:

The screenshot shows a web browser window with the address bar displaying 'localhost:8000/akce/insert'. The page has a blue header with the text 'Kalendář akcí'. Below the header is a grey box with the title 'Nová akce'. The form contains the following elements:

- Název akce:** A text input field containing 'Filmový festival'.
- Datum konání akce:** A date input field containing '13.01.2020'.
- Stručný popis:** A large text area for a brief description, currently empty.
- Kategorie:** A dropdown menu. A validation error message 'Vyplňte prosím toto pole.' is displayed next to it.
- Hodnocení:** A number input field containing '3,0'.
- Potvrdit:** A button to submit the form.

10. Vkládání a editace záznamů

Po vytvoření formuláře musíme zajistit, aby se v něm při jeho použití v různých akcích (insert, update) objevovala správná přednastavená data.

V případě akce **insert** to může být velmi jednoduché, protože na začátku může být k dispozici prázdný formulář. Chceme-li přece jen do některých prvků nastavit nějaká výchozí data, učiníme tak v metodě **actionInsert()**, která je součástí presenteru **AkcePresenter**:

```
/* Metoda zajistí přípravu vložení nové akce a poté vyrendruje stránku s formulářem */

public function actionInsert(): void {
    Debugger::log('Vložen nový záznam');
    /* Vytvoření objektu s aktuálním dato/časovým údajem */
    $dnes = new DateTime();
    /* Nastavení výchozích hodnot do některých formulářových prvků */
    $this['akceForm']['datum']->setDefaultValue($dnes->format('Y-m-d'));
    $this['akceForm']['hodnoceni']->setDefaultValue('3.0');
    $this['akceForm']['kategorie']->setDefaultValue('kino');
}
```

Prvky objektu formuláře tvoří asociativní pole a přistupujeme k nim pomocí názvů (v ideálním případě odpovídají názvům sloupců v tabulce). Metodě **setDefaultValue()** musíme ovšem předat přípustnou hodnotu. Chceme-li tedy například zobrazit jako výchozí aktuální datum, připravíme si nejprve pomocí utility **Nette\Utils\DateTime** nový objekt (**\$dnes = new DateTime()**) a metodou **format('Y-m-d')** datum ve správném tvaru předáme do formuláře.

Abychom třídu **DateTime** mohli takto využít, nesmíme ji zapomenout připojit na začátku presenteru:

```
/* Použití jmenného prostoru pro přístup k utilitě DateTime */
use Nette\Utils\DateTime;
```

V případě akce **update** je příprava výchozích hodnot do formuláře nezbytná - chceme přece jen aktualizovat už existující záznam. Může to vypadat například takto:

```
/* Metoda zajistí přípravu aktualizace akce a poté vyrendruje stránku s formulářem */

public function actionUpdate($id): void {
    Debugger::log('Aktualizován záznam ' . $id);
    /* Načte záznam o dané akci (podle id) a převede ho na asociativní pole */
    $data = $this->akceManager->getById($id)->toArray();
    /* Převede datum z formátu DateTime na datum ve tvaru rok-měsíc-den */
    $data['datum'] = $data['datum']->format('Y-m-d');
    Debugger::barDump($data);
    /* Nastaví výchozí hodnoty ve formuláři podle předaných dat */
    $this['akceForm']->setDefaults($data);
}
```

Poznámky popisují jednotlivé kroky. Zdůrazníme jen, že do proměnné **\$data** je pomocí metody **AkceManageru** načten záznam podle id a hned je metodou **toArray()** převeden do podoby asociativního pole. Díky tomu můžeme ještě před samotným předáním dat do formuláře (**\$this['akceForm']->setDefaults(\$data)**) provést žádoucí úpravu formátu data (**\$data['datum'] = \$data['datum']->format('Y-m-d')**).

Vše si můžeme průběžně kontrolovat pomocí ladicích prostředků - zde např. **Debugger::barDump(\$data)**;

Výsledek může i s pomocnými informacemi z Laděnky na stránce vypadat takto:

← → ↻ localhost:8000/akce/update/4 ☆ M Pozastaveno

Kalendář akcí

Aktualizace akce

Název akce: Abonentní koncert České

Datum konání akce: 09.04.2009

Stručný popis: Již 7. koncert abonentních cyklů E a F České filharmonie bude ve dnech 9. a 10. dubna v Rudolfinu

Kategorie: hudba ▼

Hodnocení: 2

Potvrdit

SSPU STŘE AUM

Dumps

```
array (6) ▼
  id_akce => 4
  nazev => "Abonentní koncert České filharmonie"
  datum => "2009-04-09"
  popis => "Již 7. koncert abonentních cyklů E a F České filharmonie bude ve dnech 9. a 10. dubna v F"
  kategorie => "hudba"
  hodnoceni => 2.0
```

TRACY 100.5 ms DIC Akce:update 0.4 ms/2 dumps

V tomto okamžiku se ještě musíme postarat o to nejdůležitější: o samotné uložení dat z formuláře do databáze. Musíme tedy změnit kód v již připravené metodě **akceFormSucceeded()**, která má tento úkol na starosti:

```
/* Volá se po úspěšném odeslání formuláře */

public function akceFormSucceeded(UI\Form $form, $values): void {
    Debugger::barDump($values);
    /* Do proměnné $akceId uložíme informaci o id, které je v případě update předáváno
ve formě parametru */
    $akceId = $this->getParameter('id');

    /* Pokud proměnná $akceId obsahuje nějaký údaj (není tedy false)... */
    if ($akceId) {
        /* předpokládáme akci update a použijeme připravenou metodu update, která je
součástí AkceManager... */
        $akce = $this->akceManager->update($akceId, $values);
    } else {
        /* ...v opačném případě hodláme vložit nový záznam metodou insert */
        $akce = $this->akceManager->insert($values);
    }
    /* Jestliže návratová hodnota uložená v proměnné $akce byla true */
    if ($akce) {
        /* zobrazíme "úspěšnou" zprávu na zeleném pozadí... */
    }
}
```



```

        $this->flashMessage('Akce byla úspěšně uložena', 'success');
    } else {
        /* ...jinak se zobrazí chybová zpráva na červeném pozadí */
        $this->flashMessage('Došlo k nějaké chybě při ukládání do databáze',
'danger');
    }
    /* Přesměrování na stránku se seznamem akcí - tj. presenter Akce, metoda list */
    $this->redirect('Akce:list');
}

```

Jednotlivé kroky jsou opět podrobně popsány v komentářích.

Ověřme si na příkladu vložení nového záznamu a jeho následné editaci funkčnost obou operací.

11. Vylepšení vzhledu formuláře

V naší aplikaci využíváme **Bootstrap 4** a bylo by proto vhodné, aby i formulář odpovídal standardnímu vzhledu, který tento framework nabízí, a zároveň byl responzivní.

K tomu potřebujeme do pomocné šablony `app\Presenters\templates\components\form.latte` přidat následující blok kódu:

```

{* for Bootstrap v4 *}
{define bootstrap4-form $formName}
    <form n:name=$formName>
        {if $form->ownErrors}
            <div class="alert alert-danger">
                <ul class="error n:ifcontent">
                    <li n:foreach="$form->ownErrors as $error">{$error}</li>
                </ul>
            </div>
        {/if}
        <div n:foreach="$form->controls as $name => $input"
            n:if="!$input->getOption(rendered) && $input->getOption(type) !== hidden"
            n:class="form-group, row, $input->required ? required, $input->error ? has-error">
            <div class="col-sm-3 control-label">{label $input}</div>
            <div n:class="$input->getOption(type) === button ? offset-sm-3, col-sm-9">
                {if $input->getOption(type) in [text, select, textarea]}
                    {input $input class => form-control}
                {elseif $input->getOption(type) === button}
                    {input $input class => "btn btn-primary"}
                {elseif $input->getOption(type) === checkbox}
                    <div class="checkbox">{input $input}</div>
                {elseif $input->getOption(type) === radio}
                    <div class="radio">{input $input}</div>
                {else}
                    {input $input}
                {/if}

                <span class="help-block n:ifcontent">{$input->error ?: $input-
>getOption(description)}</span>
            </div>
        </div>
    </form>
{enddefine}

```

Jedná se o "předpis" (definici), jak má systém Latté renderovat jednotlivé formulářové prvky.

Nyní ještě musíme změnit způsob zobrazování formuláře v šablonách **insert.latte** a **update.latte**. Zde je příklad úpravy souboru **insert.latte**:

```
{* Šablona pro vložení nové akce *}
{block content}
<h2 class="text-center bg-secondary text-white p-3">Nová akce</h2>
{* Použití šablony form.latte pro zobrazení formuláře akceForm pomocí
   renderovací definice bootstrap4-form *}
{import '../components/form.latte'}
{include bootstrap4-form akceForm}
{* control akceForm *}
{/block}

{block scripts}
{include parent}
{/block}

{block head}
{/block}
```

Výsledek pak vypadá takto:

The screenshot shows a web browser window with the address bar displaying 'localhost:8000/akce/update/15'. The page has a blue header with the title 'Kalendář akcí'. Below the header, there is a form titled 'Aktualizace akce' (Update event). The form contains the following fields:

- Název akce: Hrdý Budžes (I. Dousková)
- Datum konání akce: 10.04.2009
- Stručný popis: Host Divadlo A. Dvořáka Příbram - v hlavní roli Barbora Hrzánová Skvělá tragikomedie o prvních letech husákovské normalizace „očima“ zvědavé žákyně druhé třídy ZDŠ, Heleny Součkové. Hrají: B. Hrzánová (oceněna za tuto roli Thalií 2003), J. Vlčková/M. Šíková, L. Jeník. Divadlo Bez zábradlí od 20 hodin
- Kategorie: divadlo
- Hodnocení: 3

At the bottom of the form is a blue button labeled 'Potvrdit' (Confirm).

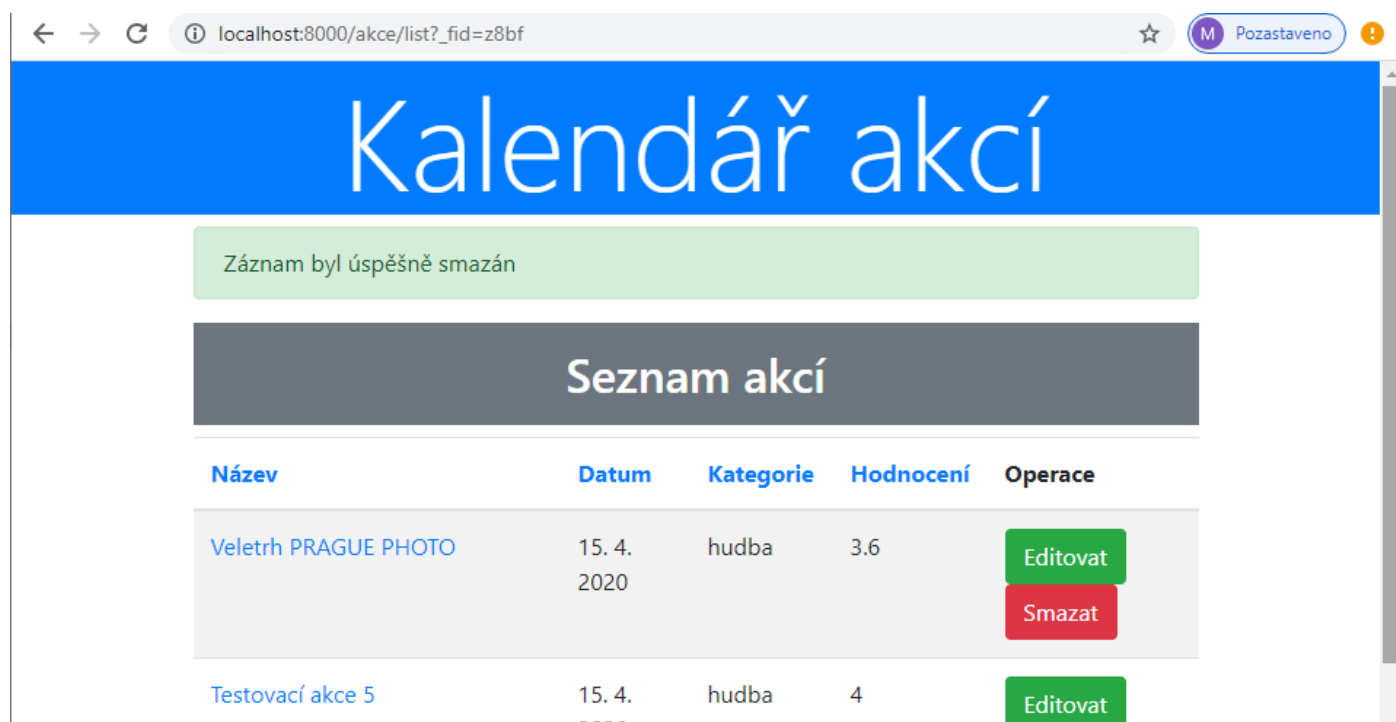
12. Vymazání záznamu

Poslední akcí, kterou musíme ošetřit, je odstranění záznamu z databáze. Doplníme několik řádků kódu do metody **actionDelete()** v **AkcePresenter**:

```
/* Metoda provede smazání záznamu o akci a poté přesměruje zpět na seznam akcí */

public function actionDelete($id): void {
    Debugger::log('Odstraněn záznam ' . $id);
    if ($this->akceManager->delete($id)) {
        $this->flashMessage('Záznam byl úspěšně smazán', 'success');
    } else {
        $this->flashMessage('Došlo k nějaké chybě při mazání záznamu', 'danger');
    }
    $this->redirect('list');
}
```

Parametr **\$id**, který identifikuje odstraňovaný záznam, použijeme v metodě **delete()** připravené v **AkceManager** a pomocí bleskové zprávy informujeme uživatele o úspěchu či neúspěchu operace.



The screenshot shows a web browser at `localhost:8000/akce/list?_fid=z8bf`. The page has a blue header with the title "Kalendář akcí". Below the header, a green message box states "Záznam byl úspěšně smazán". Underneath is a grey header for "Seznam akcí". A table lists two events:

Název	Datum	Kategorie	Hodnocení	Operace
Veletřh PRAGUE PHOTO	15. 4. 2020	hudba	3.6	Editovat Smazat
Testovací akce 5	15. 4. 2020	hudba	4	Editovat

13. Konečně hotovo!

Hotovou a funkční aplikaci můžeme nyní opět uložit na GitHub.