Workflow

index.php & bootstrap.php

Všechny příchozí požadavky od prohlížeče zpracovává soubor `index.php`, v kterém se neděje nic zajímavého. Definují se zde konstanty pro adresářovou strukturu aplikace a předávají se zde požadavky na stránky souboru `bootstrap.php`, který je v adresáři `app/`. Soubor `bootstrap.php` nastaví pravidla pro routování adres (tím že proběhne require souboru `app/routes.php`). Na konci `bootstrap.php` proběhne spuštění aplikace.

```
$container->run();
```

Každý požadavek se přes `index.php` a `bootsrap.php` dostane až do objektu `$container`, který je instancí `Nette\Application\Application`. Tento objekt si požadavky převezme zavoláním metody `run()`.Objekt neumí pracovat s HTTP požadavky, proto požádá objekt `RouteList` o překlad.

routes.php

Zde se vytvoří instance classy `RouteList`, které poté přidáváme pravidla pro cesty, v podobě instancí classy `Route`, tak že do `$routeru` setujeme položky jako by jsme zacházeli s polem. Toto je možné díky tomu, že `RouteList` implementuje `ArrayAccess`.

```
$router = new RouteList();
$router[] = new Route('projekt/<id>[-<name>][!.html]', 'Project:detail);
```

První parametr konstruktoru `Route` je předpis pro relativní URL. Ten funguje tak, že pokud nám přijde request např. na `http://www.volnanabidka.cz/projekt` obslouží ho akce `detail`, presenteru `Project` (viz druhý parametr `Route`), o presenterech si povíme později. Obecně druhý parametr signalizuje před dvojtečkou presenter a za dvojtečkou jeho akci. Další čeho si můžeme všimnout v předpisu pro relativní url je <id>. To je povinný parametr, který se předá akci `detail`. Dále zde máme [-<name>]. To je druhý, **nepovinný** parametr (to co je v hranatých závorkách je nepovinné) pro akci `detail` tj. do URL ho zapsat nemusíme, ale metoda `detail` presenteru `Project` ho v parametrech mít musí. Pomlčka zde signalizuje pouze separátor, nejde o žádný metaznak. A nakonec [!.html]. To říká, že nakonec URL se vždy doplní, ať už to klient zadá, nebo ne, .html. Konečná adresa tedy může mít tvar např.
`http://www.volnanabidka.cz/projekt/5-test.html`

Presentery

Presenter je obdoba controlleru, která má na starosti vyrenderování stránky.

Nejdříve si budeme muset vysvětlit životní cyklus presenteru. Tyto metody se volají postupně za sebou. Názvy ve špičatých závorkách např. action<Action>, se jmenují podle akce presenteru, když

uvedeme náš příklad, např metoda action se tedy bude jmenovat `actionDetail()`. Stejně to platí i pro ostatní akce presenteru.

1. `startup()`

Pokud tato metoda v presenteru existuje, zavolá se hned po instanciování presenteru. V této fázi neprobíhá výpis html. Zde například můžeme rozhodovat, zda má uživatel na stránku přístup.

2. `action<Action>()`

Tuto metodu jsme si nastínili v kapitole routes.php. Přijde-li nám požadavek např. na `http://www.volnanabidka.cz/projekt/5-test.html` a obsluhující presenter a jeho akce jsou `Project:detail`, zavolá se metoda, **pokud existuje**, `actionDetail($id, $name)`, presenteru `Project`, kde v proměnné $id, bude číslo 5 a v proměnné $name bude string test (proměnné se jmenují podle parametrů v cestě, ale nemusejí). V této fázi ještě neprobíhá výpis html. Tedy může zde probíhat redirect na jiný presenter atp.

3. `handle<Signal>()`

Metoda, která obsluhuje akce, které nepotřebují svoje view, nejčastěji tím bývají AJAXové requesty.

4. `render<View>()`

V této metodě už není možný redirect, protože už začal samotný výpis šablony. V našem příkladě by se zavolala metoda `renderDetail($id, $name)` presenteru `Project`. Tato metoda má ve většině případů setnout proměnné do šablony `detail.latte`, která se renderuje. Opět proměnné $id a $name obsahují stejné hodnoty jako v `actionDetail()`.<View> se jmenuje stejně jako akce presenteru, tedy detail.