Serwisy REST w Haskellu – framework Yesod

Przemysław Kamiński

6 czerwca 2013
HUG Warsaw

Yesod – wstęp

Yesod (hebr. “fundament”, “podstawa”) – jedna z kabalistycznych sefir, łącząca świat materialny z duchowym

Yesod – wstęp

Yesod (hebr. “fundament”, “podstawa”) – jedna z kabalistycznych sefir, łącząca świat materialny z duchowym ... oraz Web Framework napisany w Haskellu

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

```
#> cabal install yesod-platform yesod-bin yesod-test
yesod-static cabal-dev
```

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

```
#> cabal install yesod-platform yesod-bin yesod-test
yesod-static cabal-dev
#> yesod init      # project 'rest' with sqlite
```

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

```
#> cabal install yesod-platform yesod-bin yesod-test
yesod-static cabal-dev
#> yesod init      # project 'rest' with sqlite
#> cd rest && cabal-dev install && yesod --dev devel
```

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

```
#> cabal install yesod-platform yesod-bin yesod-test
yesod-static cabal-dev
#> yesod init      # project 'rest' with sqlite
#> cd rest && cabal-dev install && yesod --dev devel
```

$\longrightarrow$ `http://localhost:3000`

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

```
#> cabal install yesod-platform yesod-bin yesod-test
yesod-static cabal-dev
#> yesod init      # project 'rest' with sqlite
#> cd rest && cabal-dev install && yesod --dev devel
```

$\longrightarrow$ `http://localhost:3000`

Serwer produkcyjny:

```
cabal-dev -fproduction configure && cabal-dev build &&
./cabal-dev/bin/rest Production
```

Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji >= 1.2):

```
#> cabal install yesod-platform yesod-bin yesod-test
yesod-static cabal-dev
#> yesod init      # project 'rest' with sqlite
#> cd rest && cabal-dev install && yesod --dev devel
```

$\longrightarrow$ `http://localhost:3000`

Serwer produkcyjny:

```
cabal-dev -fproduction configure && cabal-dev build &&
          ./cabal-dev/bin/rest Production
```

```
#> yesod test
```

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:
`./config/routes`

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:
`./config/routes`
`./Handler/`

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

`./config/routes`

`./Handler/`

`./templates/`

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

```
./config/routes
./Handler/
./templates/ *.{hamlet(HTML)}
```

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

```
./config/routes
./Handler/
./templates/ *.{hamlet , lucius(CSS)}
```

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

```
./config/routes
./Handler/
./templates/ *.{hamlet , lucius , julius(JS)}
```

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

```
./config/routes
./Handler/
./templates/ *.{hamlet , lucius , julius}
./config/models
```

Yesod – podstawy

Zmiana w pliku $\Rightarrow$ rekompilacja w locie przez serwer

Katalogi:

```
./config/routes
./Handler/
./templates/ *.{hamlet , lucius , julius}
./config/models
```

Zróbmy sobie `git init` i zacommitujmy tą wersję.

Yesod – Echo!

Request GET pod `/echo/{str}` zwraca HTML z `<h1>{str}</h1>`.

Yesod – Echo!

Request GET pod /echo/{str} zwraca HTML z <h1>{str}</h1>.

```
#> yesod add-handler
name:  Echo, pattern:  /echo/#String, request:  GET
```

Yesod – Echo!

Request GET pod /echo/{str} zwraca HTML z <h1>{str}</h1>.

```
#> yesod add-handler
name:  Echo, pattern:  /echo/#String, request:  GET

git status...
```

Yesod – Echo! c.d.

Edytujemy `./Handler/Echo.hs`:

```
getEchoR theText = defaultLayout
                   [whamlet|<h1>#{theText}|]
```

Yesod – Echo! c.d.

Edytujemy ./Handler/Echo.hs:

```
getEchoR theText = defaultLayout
                     [whamlet|<h1>#{theText}|]
```

template Haskell (expression quotation)

Yesod – Echo! c.d.

Edytujemy `./Handler/Echo.hs`:

```
getEchoR theText = defaultLayout
                     [whamlet|<h1>#{theText}|]
```

template Haskell (expression quotation)

Script injection protection:

`/echo/%3Cscript%3Ealert%28%22foo%22%29%3C%2Fscript%3E`

Yesod – Echo! c.d.

Edytujemy `./Handler/Echo.hs`:

```
getEchoR theText = defaultLayout
                     [whamlet|<h1>#{theText}|]
```

template Haskell (expression quotation)

Script injection protection:
`/echo/%3Cscript%3Ealert%28%22foo%22%29%3C%2Fscript%3E`

Wszystko dzięki systemowi typów, który konwertuje typ `URL` na `String` (*The web, by its very nature, is not type safe. Even the simplest case of distinguishing between an integer and string is impossible: all data on the web is transferred as raw bytes, evading our best efforts at type safety. Every app writer is left with the task of validating all input.*, Yesod Book)

Yesod – Echo! c.d.

Aby zamiast typu `String` używać typu `Data.Text`, dodajemy jako ostatni import w pliku `./Foundations.hs`:

```
import Data.Text
```

a w pliku `./config/routes` zamieniamy `#String` na `#Text`. Modyfikujemy jeszcze syngnaturę funkcji `getEchoR` w pliku `./Handler/Echo.hs`.

Yesod – system szablonów

hamlet – biblioteka do renderowania HTML z szablonów

`http://hackage.haskell.org/package/hamlet-1.1.7`

Yesod – system szablonów

hamlet – biblioteka do renderowania HTML z szablonów
`http://hackage.haskell.org/package/hamlet-1.1.7`

Dodajemy plik `./templates/echo.hamlet` z zawartością:

```
<h1>#{theText}
```

a w pliku `./Handler/Echo.hs` dajemy

```
getEchoR theText = defaultLayout $(widgetFile "echo")
```

Yesod – system szablonów

hamlet – biblioteka do renderowania HTML z szablonów
`http://hackage.haskell.org/package/hamlet-1.1.7`

Dodajemy plik `./templates/echo.hamlet` z zawartością:

```
<h1>#{theText}
```

a w pliku `./Handler/Echo.hs` dajemy

```
getEchoR theText = defaultLayout $(widgetFile "echo")
```

template Haskell (splice)

Yesod – lustro

```
#> yesod add-handler
name:  Mirror, pattern:  /mirror, request:  GET POST
```

Yesod – lustro

```
#> yesod add-handler
name:  Mirror, pattern:  /mirror, request:  GET POST
```

Edytujemy `./Handler/Mirror.hs`...

Yesod – blog

```
#> yesod add-handler
name:  Blog, pattern:  /blog, request:  GET POST
```

Yesod – blog

```
#> yesod add-handler
name:  Blog, pattern:  /blog, request:  GET POST
#> yesod add-handler
name:  Article, pattern:  /blog/#ArticleId, request:
GET
```

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models`...

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models...` w konsoli z serwerem widzimy komunikat

```
Migrating: CREATE TABLE “article”(“id” INTEGER
PRIMARY KEY,“title” VARCHAR NOT NULL,“content” VARCHAR
NOT NULL)
```

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models...` w konsoli z serwerem widzimy komunikat

```
Migrating: CREATE TABLE ‘‘article’’(‘‘id’’ INTEGER
PRIMARY KEY,‘‘title’’ VARCHAR NOT NULL,‘‘content’’ VARCHAR
NOT NULL)
```

Edytujemy `./Handler/Blog.hs...`

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models...` w konsoli z serwerem widzimy komunikat

```
Migrating: CREATE TABLE ‘‘article’’(‘‘id’’ INTEGER
PRIMARY KEY,‘‘title’’ VARCHAR NOT NULL,‘‘content’’ VARCHAR
NOT NULL)
```

Edytujemy `./Handler/Blog.hs...`
`./templates/articles.hamlet...`

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models...` w konsoli z serwerem widzimy komunikat

```
Migrating: CREATE TABLE “article”(“id” INTEGER
PRIMARY KEY,“title” VARCHAR NOT NULL,“content” VARCHAR
NOT NULL)
```

Edytujemy `./Handler/Blog.hs...`

`./templates/articles.hamlet...`

`./templates/articleError.hamlet...`

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models...` w konsoli z serwerem widzimy komunikat

```
Migrating: CREATE TABLE ‘‘article’’(‘‘id’’ INTEGER
PRIMARY KEY,‘‘title’’ VARCHAR NOT NULL,‘‘content’’ VARCHAR
NOT NULL)
```

Edytujemy `./Handler/Blog.hs...`

- `./templates/articles.hamlet...`
- `./templates/articleError.hamlet...`
- `./Handler/Article.hs...`

Yesod – blog

```
#> yesod add-handler
name: Blog, pattern: /blog, request: GET POST
#> yesod add-handler
name: Article, pattern: /blog/#ArticleId, request:
GET
```

Edytujemy `./config/models...` w konsoli z serwerem widzimy komunikat

```
Migrating: CREATE TABLE ‘‘article’’(‘‘id’’ INTEGER
PRIMARY KEY,‘‘title’’ VARCHAR NOT NULL,‘‘content’’ VARCHAR
NOT NULL)
```

Edytujemy `./Handler/Blog.hs...`

- `./templates/articles.hamlet...`
- `./templates/articleError.hamlet...`
- `./Handler/Article.hs...`
- `./templates/article.hamlet...`

Yesod – Hamlet

- sterowany indentacją

Yesod – Hamlet

- sterowany indentacją $\rightarrow$ bez tagów zamykających!

Yesod – Hamlet

- sterowany indentacją $\rightarrow$ bez tagów zamykających!
  `<h1>Tytuł`

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
      lub
<h1>Dłuższy #
     tytuł
```

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

lub

```
<h1>Dłuższy #
    tytuł
```

- nie trzeba domykać tagów, jeśli nic za nimi nie będzie w danej linii:

```
<p
    <a href=@{MyHandler}>My Handler
```

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

lub

```
<h1>Dłuższy #
    tytuł
```

- nie trzeba domykać tagów, jeśli nic za nimi nie będzie w danej linii:

```
<p
    <a href=@{MyHandler}>My Handler
```

- Zmienne:

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

lub

```
<h1>Dłuższy #
    tytuł
```

- nie trzeba domykać tagów, jeśli nic za nimi nie będzie w danej linii:

```
<p
    <a href=@{MyHandler}>My Handler
```

- Zmienne:
  - `#{...}` – zwykła zmienna

    np.:

```
import qualified Data.Text as T
getTestR = do
    var <- return $ toHtml $ T.pack "<script>"
defaultLayout [whamlet|
    Variable:  #{var}, Reversed:  #{T.reverse var}
|]
```

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

lub

```
<h1>Dłuższy #
    tytuł
```

- nie trzeba domykać tagów, jeśli nic za nimi nie będzie w danej linii:

```
<p
    <a href=@{MyHandler}>My Handler
```

- Zmienne:
  - `#{...}` – zwykła zmienna
  - `@{...}` – interpolacja URL (type-safe)

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

lub

```
<h1>Dłuższy #
    tytuł
```

- nie trzeba domykać tagów, jeśli nic za nimi nie będzie w danej linii:

```
<p
    <a href=@{MyHandler}>My Handler
```

- Zmienne:
  - `#{...}` – zwykła zmienna
  - `@{...}` – interpolacja URL (type-safe)
  - `^{...}` – wklejanie innych szablonów

Yesod – Hamlet

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

lub

```
<h1>Dłuższy #
    tytuł
```

- nie trzeba domykać tagów, jeśli nic za nimi nie będzie w danej linii:

```
<p
    <a href=@{MyHandler}>My Handler
```

- Zmienne:
  - `#{...}` – zwykła zmienna
  - `@{...}` – interpolacja URL (type-safe)
  - `^{...}` – wklejanie innych szablonów
  - `@?{...}` – URL z parametrami GET

Yesod – Hamlet c.d.

- `newIdent` – automatycznie generuje ID dla elementu; potencjalnie redukuje to ilość błędów w komunikacji HTML↔CSS↔JS

Yesod – Hamlet c.d.

- `newIdent` – automatycznie generuje ID dla elementu; potencjalnie redukuje to ilość błędów w komunikacji HTML↔CSS↔JS
- `$if/$elseif/$else`

Yesod – Hamlet c.d.

- `newIdent` – automatycznie generuje ID dla elementu; potencjalnie redukuje to ilość błędów w komunikacji HTML↔CSS↔JS
- `$if/$elseif/$else`
- `$forall`

Yesod – Hamlet c.d.

- `newIdent` – automatycznie generuje ID dla elementu; potencjalnie redukuje to ilość błędów w komunikacji HTML↔CSS↔JS
- `$if/$elseif/$else`
- `$forall`
- `$maybe/$nothing` (można zastąpić `$if, isJust, fromJust`)

```
$maybe name <- maybeName
    <p>Your name is #{name}
$nothing
    <p>I don't know your name.
```

Yesod – Hamlet c.d. 2

- warunkowe atrybuty:

```
<input type=checkbox :isSelected:selected=true
:isCurrent:.current>
```

Yesod – Hamlet c.d. 2

- warunkowe atrybuty:
  ```
  <input type=checkbox :isSelected:selected=true
  :isCurrent:.current>
  ```
- Skróty:

Yesod – Hamlet c.d. 2

- warunkowe atrybuty:
  `<input type=checkbox :isSelected:selected=true :isCurrent:.current>`
- Skróty:
  `<p #someId>` $\Leftrightarrow$ `<p id=''someId''>`

Yesod – Hamlet c.d. 2

- warunkowe atrybuty:
  `<input type=checkbox :isSelected:selected=true :isCurrent:.current>`
- Skróty:
  `<p #someId>` ⇔ `<p id=''someId''>`
  `<p ##{var}>` ⇔ `<p id=''var_value''>`

Yesod – Hamlet c.d. 2

- warunkowe atrybuty:
  `<input type=checkbox :isSelected:selected=true :isCurrent:.current>`
- Skróty:
  `<p #someId>` ⇔ `<p id=''someId''>`
  `<p ##{var}>` ⇔ `<p id=''var_value''>`
  `<p .someClass>` ⇔ `<p class=''someClass''>`

Yesod – Julius, Cassius

CSS – *Cassius* (lub *Lucius* – nadzbiór języka CSS)

Yesod – Julius, Cassius

CSS – *Cassius* (lub *Lucius* – nadzbiór języka CSS)
JS – *Julius*

Yesod – Julius, Cassius

CSS – *Cassius* (lub *Lucius* – nadzbiór języka CSS)
JS – *Julius*

Przykład: `./Handlers/Shakespeare.hs`...

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS
- zewnętrznych JS

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS
- zewnętrznych JS
- wewnętrznych CSS

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS
- zewnętrznych JS
- wewnętrznych CSS
- wewnętrznych JS

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS
- zewnętrznych JS
- wewnętrznych CSS
- wewnętrznych JS
- dowonlej wstawki do `HEAD`

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS
- zewnętrznych JS
- wewnętrznych CSS
- wewnętrznych JS
- dowonlej wstawki do `HEAD`
- dowonlej wstawki do `BODY`

Yesod – Widgety

widget – "klej" łączący Hamlet/Cassius/Julius; małe, spójne fragmenty HTML-CSS-JS, nadające się do wielokrotnego użycia

Dzięki widgetom CSS-y zawsze trafiają do `HEAD`, a JS-y albo do `HEAD` albo bezpośrednio do `BODY`.

Widget składa się z:

- tytułu
- zewnętrznych CSS
- zewnętrznych JS
- wewnętrznych CSS
- wewnętrznych JS
- dowonlej wstawki do `HEAD`
- dowonlej wstawki do `BODY`

Zewnętrzne CSS/JS powinny pojawić się na stronie tylko raz

Yesod – Widgety c.d.

Widgety można łączyć:

```
myWidget = do
    myWidget1
    myWidget2
```

Yesod – Widgety c.d.

Widgety można łączyć:

```
myWidget = do
    myWidget1
    myWidget2

whamlet -- ./whamlet.hs
```

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów. Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów.
Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.
Routing może składać się z 3 części:

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów.
Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.
Routing może składać się z 3 części:

- *statycznej* – stały string w URL, np. `/echo`

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów.
Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.
Routing może składać się z 3 części:

- *statycznej* – stały string w URL, np. `/echo`
- *pojedynczej dynamicznej* – `#<Type>`, np. `/page/#Int` (`Type` – instancja `SinglePiece`)

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów.
Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.
Routing może składać się z 3 części:

- *statycznej* – stały string w URL, np. `/echo`
- *pojedynczej dynamicznej* – `#<Type>`, np. `/page/#Int` (`Type` – instancja `SinglePiece`)
- *multi-dynamicznej* – `<MultiType>`, np. `/page/#Int` (`MultiType` – instancja `MultiPiece`)

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów.
Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.
Routing może składać się z 3 części:

- *statycznej* – stały string w URL, np. `/echo`
- *pojedynczej dynamicznej* – `#<Type>`, np. `/page/#Int` (`Type` – instancja `SinglePiece`)
- *multi-dynamicznej* – `<MultiType>`, np. `/page/#Int` (`MultiType` – instancja `MultiPiece`)
  Typ multi musi występować zawsze na końcu URL.

Yesod – system routingu

Yesod konwertuje do/z URL-i podanych w tabeli routingów.
Wszystkie niekanoniczne URl-e kończą się redirectem do URL kanonicznego.
Routing może składać się z 3 części:

- *statycznej* – stały string w URL, np. `/echo`
- *pojedynczej dynamicznej* – `#<Type>`, np. `/page/#Int` (`Type` – instancja `SinglePiece`)
- *multi-dynamicznej* – `<MultiType>`, np. `/page/#Int` (`MultiType` – instancja `MultiPiece`)
  Typ multi musi występować zawsze na końcu URL.
  Przydaje się do implementowania URL-i o strukturze drzewiastej, np. katalog plików albo wikipedia.

Yesod – REST + JSON (AESON)

Uwaga: od wersji 1.2 `yesod-json` został włączony do `yesod-core` i zamiast zmienił namespace z `Yesod.Json` na `Jesod.Core.Json`.

```
#> yesod add-handler
name:  Json, pattern:  /json, request:  GET
```

Yesod – REST + JSON (AESON)

Uwaga: od wersji 1.2 `yesod-json` został włączony do `yesod-core` i zamiast zmienił namespace z `Yesod.Json` na `Jesod.Core.Json`.

```
#> yesod add-handler
name:  Json, pattern:  /json, request:  GET

./Handler/Json.hs...
```

Dziękuję