

# Serwisy REST w Haskellu – framework Yesod

Przemysław Kamiński

6 czerwca 2013  
HUG Warsaw

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

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

Szybki wstęp (dla wersji  $\geq 1.2$ ):

Szybki wstęp (dla wersji  $\geq 1.2$ ):

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

# Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji  $\geq 1.2$ ):

```
#> 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  $\geq 1.2$ ):

```
#> 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
```

Szybki wstęp (dla wersji  $\geq 1.2$ ):

```
#> 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
```

→ <http://localhost:3000>



# Yesod – konfiguracja prostej aplikacji

Szybki wstęp (dla wersji  $\geq 1.2$ ):

```
#> 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
```

→ <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  $\geq 1.2$ ):

```
#> 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
```

→ <http://localhost:3000>

Serwer produkcyjny:

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

```
#> yesod test
```

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

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

Katalogi:

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

Katalogi:  
`./config/routes`

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

Katalogi:

`./config/routes`

`./Handler/`

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

Katalogi:

`./config/routes`

`./Handler/`

`./templates/`

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

Katalogi:

`./config/routes`

`./Handler/`

`./templates/ *.{hamlet(HTML)}`



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

Katalogi:

`./config/routes`

`./Handler/`

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

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

Katalogi:

`./config/routes`

`./Handler/`

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

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

Katalogi:

`./config/routes`

`./Handler/`

`./templates/ *.{hamlet , lucius , julius}`

`./config/models`

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.

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

Request GET pod /echo/{str} zwraca HTML z

<h1>{str}</h1>.

```
#> yesod add-handler
```

```
name: Echo, pattern: /echo/#String, request: GET
```

Request GET pod /echo/{str} zwraca HTML z

```
<h1>{str}</h1>.
```

```
#> yesod add-handler
```

```
name:  Echo, pattern:  /echo/#String, request:  GET
```

```
git status...
```

Edytujemy ./Handler/Echo.hs:

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



Edytujemy ./Handler/Echo.hs:

```
getEchoR theText = defaultLayout
    [whamlet|<h1>#{theText}|]
    template Haskell (expression quotation)
```

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
```

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)

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 sygnaturę funkcji `getEchoR` w pliku `./Handler/Echo.hs`.

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

**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")
```

**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 add-handler  
name: Mirror, pattern: /mirror, request: GET POST
```



```
#> yesod add-handler
```

```
name: Mirror, pattern: /mirror, request: GET POST
```

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

```
#> yesod add-handler
```

```
name:  Blog, pattern:  /blog, request:  GET POST
```

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

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

Edytujemy ./config/models...

```
#> 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 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 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 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 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 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...

- sterowany indentacją

- sterowany indentacją → bez tagów zamykających!

- sterowany indentacją → bez tagów zamykających!  
`<h1>Tytuł`

- sterowany indentacją → bez tagów zamykających!

```
<h1>Tytuł
```

```
    lub
```

```
<h1>Dłuższy #
```

```
    tytuł
```

- 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
```

- 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:



- 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:  #{reverse var}
```

```
|]
```

- 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)

- 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

- 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

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

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

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

- `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.
```



- warunkowe atrybuty:

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

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

- warunkowe atrybuty:

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

- Skróty:

```
<p #someId> ⇔ <p id=""someId"">
```

- warunkowe atrybuty:

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

- Skróty:

```
<p #someId> ⇔ <p id=""someId"">  
<p ##{var}> ⇔ <p id=""var_value"">
```

- 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"">
```

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

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

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

JS – *Julius*

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



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

*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* – “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:

*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

*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

*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

*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

*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



*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
- dowlonlej wstawki do HEAD

*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

*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

Widgety można łączyć:

```
myWidget = do  
    myWidget1  
    myWidget2
```

Widgety można łączyć:

```
myWidget = do  
    myWidget1  
    myWidget2
```

```
whamlet - ./whamlet.hs
```

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

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

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

Routing może składać się z 3 części:

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



Yesod konwertuje do/z URL-i podanych w tabeli routingów. Wszystkie niekanoniczne URI-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 konwertuje do/z URL-i podanych w tabeli routingów. Wszystkie niekanoniczne URI-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 konwertuje do/z URL-i podanych w tabeli routingów. Wszystkie niekanoniczne URI-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 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.

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

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

Dziękuję