CHAPTER 1

TinyBlog: Building a Web Interface with Seaside

1.1 Previous Week Solution

You can load the solution of the previous week using the following snippet:

```
Metacello new
    smalltalkhubUser: 'PharoMooc' project: 'TinyBlog';
    version: #week3solution;
    configuration: 'TinyBlog';
    load
```

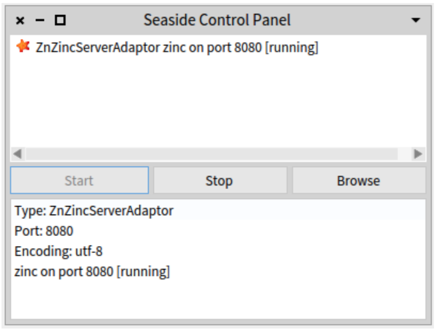
Make sure that you Teapot server is stopped.

```
    TBTeapotWebApp stop
```

1.2 A Web UI for TinyBlog with Seaside

The work presented in this exercise does not require the one on Voyage and MongoDB.

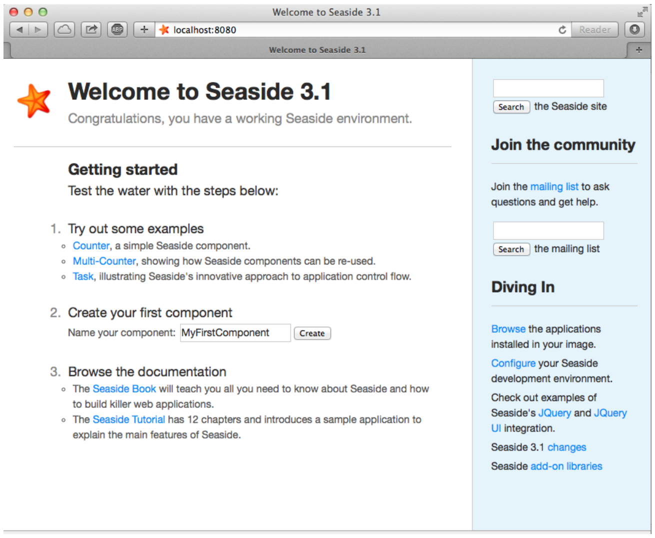
This week, we start with a web UI dedicated to display posts to users. Next week, we will develop another web UI for the blog owner to admnistrate the posts. In both cases, we will define Seaside components http://www.seaside.st which has a freely available book online here: http://book.seaside.st.



ZnZincServerAdaptor zinc on port 8080 [running]

Type: ZnZincServerAdaptor
Port: 8080
Encoding: utf-8
zinc on port 8080 [running]

Figure 1.1 Start Seaside Server.



Welcome to Seaside 3.1
Congratulations, you have a working Seaside environment.

Getting started
Test the water with the steps below:

1. Try out some examples
   - Counter, a simple Seaside component.
   - Multi-Counter, showing how Seaside components can be re-used.
   - Task, illustrating Seaside's innovative approach to application control flow.
2. Create your first component
   Name your component: MyFirstComponent
3. Browse the documentation
   - The Seaside Book will teach you all you need to know about Seaside and how to build killer web applications.
   - The Seaside Tutorial has 12 chapters and introduces a sample application to explain the main features of Seaside.

Search the Seaside site

Join the community
Join the mailing list to ask questions and get help.
Search the mailing list

Diving In
Browse the applications installed in your image.
Configure your Seaside development environment.
Check out examples of Seaside's JQuery and JQuery UI integration.
Seaside 3.1 changes
Seaside add-on libraries

Figure 1.2 Accessing Seaside Home Page.

1.3 Start Seaside

There are two ways to start Seaside. The first one consists in executing the following code:

```
ZnZincServerAdaptor startOn: 8080.
```

The second one uses the graphical tool named ”Seaside Control Panel” (World Menu>Tools>Seaside Control Panel). In the contextual menu (right clic) of this tool, select ”add adaptor...” and add a server of type `ZnZincServerAdaptor`, then define the port number (e.g. 8080) it should run on (cf. figure 1.1). By opening a web browser on the URL http://localhost:8080, you should see the Seaside home page as displayed on Figure 1.2.

1.4 Entry Point for TinyBlog Web UI

Create a class named `TBApplicationRootComponent` which will be the entry point of the application.

```
WAComponent subclass: #TBApplicationRootComponent
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'TinyBlog-Components'
```

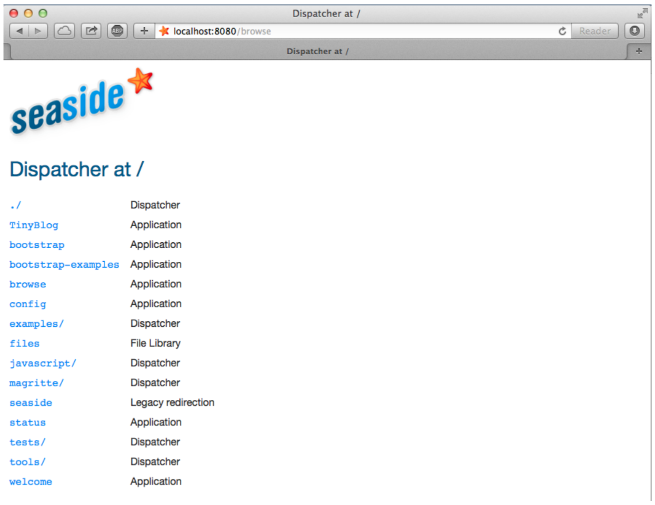
On class-side, we implement the `initialize` method in the `'initialization'` protocol to declare the application to Seaside. We also integrate dependencies to the Bootstrap framework (CSS and JS files will be embedded in the application).

```
TBApplicationRootComponent class >> initialize
   "self initialize"
   | app |
   app := WAAdmin register: self asApplicationAt: 'TinyBlog'.
   app
      addLibrary: JQDeploymentLibrary;
      addLibrary: JQUiDeploymentLibrary;
      addLibrary: TBSDeploymentLibrary
```

Once declared, you should execute this method with `TBApplicationRootComponent initialize`. Indeed, class-side `initialize` methods are executed at loading-time of a class but since the class already exists, we must execute it by hand.

We also add a method named `canBeRoot` to specify that `TBApplicationRootComponent` is not a simple Seaside component but a complete application. This component will be automatically instantiated when a user connects to the application.

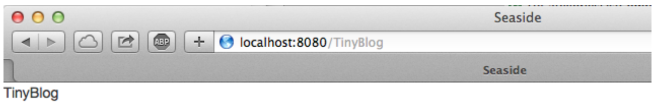
```
TBApplicationRootComponent class >> canBeRoot
   ^ true
```



Dispatcher at /

./	Dispatcher
TinyBlog	Application
bootstrap	Application
bootstrap-examples	Application
browse	Application
config	Application
examples/	Dispatcher
files	File Library
javascript/	Dispatcher
magritte/	Dispatcher
seaside	Legacy redirection
status	Application
tests/	Dispatcher
tools/	Dispatcher
welcome	Application

Figure 1.3 TinyBlog is a Registered Seaside Application.



TinyBlog

Figure 1.4 Your First Seaside Web Page.

You can verify that your application is correctly registered by Seaside by connecting to the Seaside server through your web browser, click on ”Browse the applications installed in your image” and then see that TinyBlog appears in the list. Alternatively, you can visit http://localhost:8080/TinyBlog.

1.5 First Simple Rendering

Let’s add an instance method named `renderContentOn:` in `rendering` protocol to make our application displaying something.

```
TBApplicationRootComponent >> renderContentOn: html
    html text: 'TinyBlog'
```

If you open http://localhost:8080/TinyBlog in your web browser, the page should look like the one on figure 1.4.

You can customize the web page header and declare it as HTML 5 compliant by redefining the `updateRoot:` method.

```
TBApplicationRootComponent >> updateRoot: anHtmlRoot
   super updateRoot: anHtmlRoot.
   anHtmlRoot beHtml5.
   anHtmlRoot title: 'TinyBlog'.
```

The `title:` message is responsible for setting the page title, as can be seen in your web browser's title bar. The `TBApplicationRootComponent` component is the root component of our application. It will not display a lot of things but will contain and display other components. For example, a component to display posts to the blog readers, a component to administrate the blog and its posts, ... Therefore, we decided that the `TBApplicationRootComponent` component will contain components inheriting from the abstract class named `TBScreenComponent` that we will now define.

1.6 Visual Components for TinyBlog

We are now ready to define multiple visual components for our application. The first chapters of http://book.seaside.st can help you and give more details than this tutorial if needed.

Figure 1.5 shows the different components to develop and where they are taking place.

TBScreenComponent Component

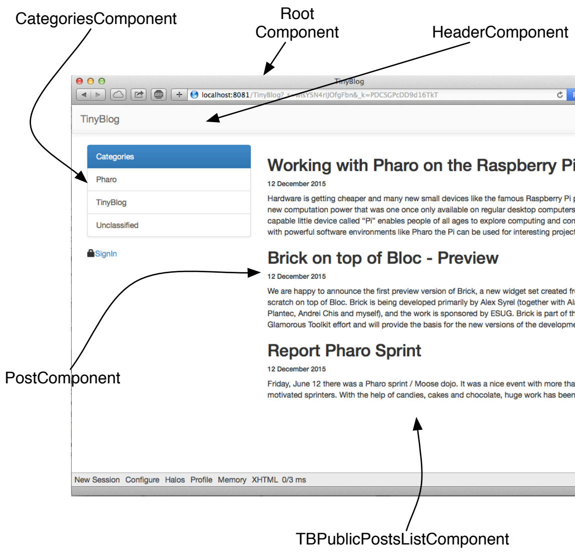
All components contained in `TBApplicationRootComponent` will be subclasses of the abstract class `TBScreenComponent`. This class allows us to factorize shared behaviors between all our components.

```
WAComponent subclass: #TBScreenComponent
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'TinyBlog-Components'
```

All components need to access the model of our application. Therefore, in the 'accessing' protocol, we add a `blog` method that returns an instance of `TBBlog` (here the singleton).

```
TBScreenComponent >> blog
   "Return the current blog. In the future we will ask the
   session to return the blog of the currently logged in user."
   ^ TBBlog current
```

Inspect now the blog object returned by `TBBlog current` and verify that it contains some posts. If it does not, execute: `TBBlog createDemoPosts`.


CategoriesComponent
Root Component
HeaderComponent
PostComponent
TBPublicPostsListComponent
TinyBlog
localhost:8081/TinyBlog?_s=wisYSN4rIJOfgFbn&_k=PDCSGPcDD9d16TkT
Reader
Categories
Pharo
TinyBlog
Unclassified
SignIn
New Session Configure Halos Profile Memory XHTML 0/3 ms

Working with Pharo on the Raspberry Pi

12 December 2015

Hardware is getting cheaper and many new small devices like the famous Raspberry Pi provide new computation power that was one once only available on regular desktop computers. This capable little device called "Pi" enables people of all ages to explore computing and combined with powerful software environments like Pharo the Pi can be used for interesting projects.

Brick on top of Bloc - Preview

12 December 2015

We are happy to announce the first preview version of Brick, a new widget set created from scratch on top of Bloc. Brick is being developed primarily by Alex Syrel (together with Alain Plantec, Andrei Chis and myself), and the work is sponsored by ESUG. Brick is part of the Glamorous Toolkit effort and will provide the basis for the new versions of the development tools.

Report Pharo Sprint

12 December 2015

Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

Figure 1.5 TinyBlog Components.

In the future, if you want to manage multiple blogs, you will need to modify this method and use information stored in the active session to retrieve the current user/blog (cf. `TBSession` later on).

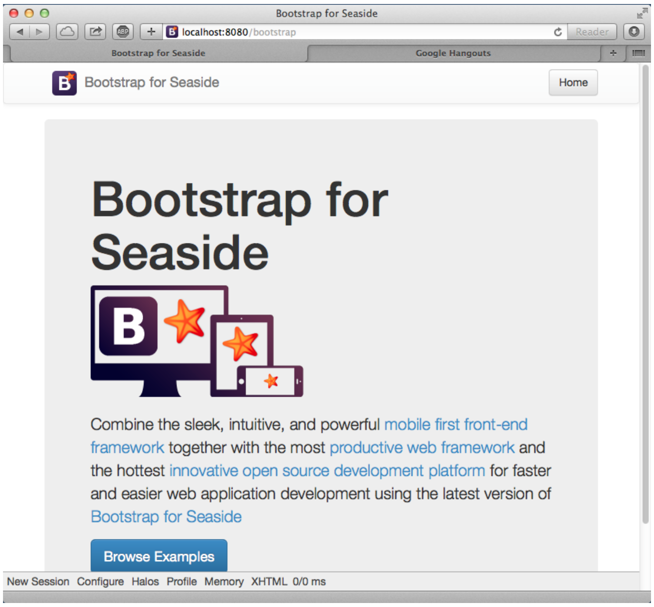
1.7 Bootstrap for Seaside

The Bootstrap library is accessible from Seaside as we will see. If you access the Seaside Bootstrap application in your web browser (**bootstrap** link in the list of Seaside application or directly through http://localhost:8080/bootstrap) you can see several examples as shown in figure 1.6.

If you click on the **Examples** link at the bottom of the web page you can see some graphical elements and their respective code to integrate them in your application (cf. figure 1.7).

Bootstrap

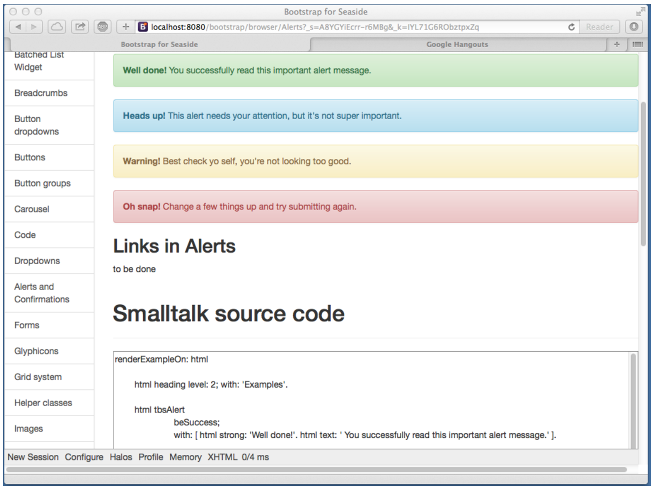
The Seaside Bootstrap library code and documentation is available here: http://smalltalkhub.com/#!/~TorstenBergmann/Bootstrap. There is also an online demo here: http://pharo.pharocloud.com/bootstrap. This library is already loaded in the PharoWeb image used in this tutorial.



Bootstrap for Seaside

Combine the sleek, intuitive, and powerful mobile first front-end framework together with the most productive web framework and the hottest innovative open source development platform for faster and easier web application development using the latest version of Bootstrap for Seaside

Figure 1.6 Bootstrap Library.



Well done! You successfully read this important alert message.

Heads up! This alert needs your attention, but it's not super important.

Warning! Best check yo self, you're not looking too good.

Oh snap! Change a few things up and try submitting again.

Links in Alerts

to be done

Smalltalk source code

```
renderExampleOn: html

    html heading level: 2; with: 'Examples'.

    html tbsAlert
            beSuccess;
            with: [ html strong: 'Well done!'. html text: ' You successfully read this important alert message.' ].
```

Figure 1.7 Bootstrap Elements and their code.

1.8 Definition of TBHeaderComponent

Let's define a component to display the top of all components in our application.

```
WAComponent subclass: #TBHeaderComponent
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'TinyBlog-Components'
```

The 'rendering' protocol contains the `renderContentOn:` method that displays our application header.

```
TBHeaderComponent >> renderContentOn: html
   html tbsNavbar beDefault with: [
      html tbsNavbarBrand
         url: '#';
         with: 'TinyBlog' ]
```

Our header is a simple navigation bar of Bootstrap (See Figure 1.8)

By default, in Bootstrap navigation bars, there is a link on tbsNavbarBrand. Since we consider that useless in our application, we used a `'#'` anchor. Now, when the user clicks on the title link, nothing happens.

Usually, clicking on an application title brings back the user to the web application home page.

1.9 Header Use

It is not desirable to instantiate the `TBHeaderComponent` each time a component is called. So, we add an instance variable named `header` in the `TBScreenComponent` component and initializes it in its `initialize` method.

```
WAComponent subclass: #TBScreenComponent
   instanceVariableNames: 'header'
   classVariableNames: ''
   package: 'TinyBlog'

TBScreenComponent >> initialize
   super initialize.
   header := TBHeaderComponent new.
```

Composite-Component relationship

In Seaside, subcomponents of a component must be returned by the composite when sending it the `children` message. So, we must define that the `TBHeaderComponent` instance is a children of the `TBScreenComponent` component in the Seaside component hierachy (and not in the Pharo classes hierarchy). We do so by specializing the method `children`.

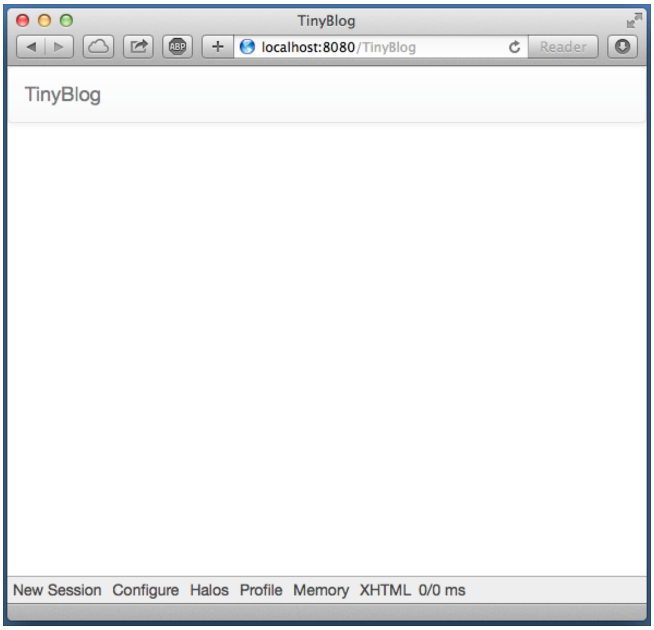


Figure 1.8 TinyBlog Application with a Header Navigation Bar.

```
TBScreenComponent >> children
   ^ OrderedCollection with: header
```

In the `renderContentOn:` method ('rendering' protocol), we can now display the subcomponent (the header):

```
TBScreenComponent >> renderContentOn: html
   html render: header
```

1.10 Using the Screen Component in our Application

Temporarily, we will directly use the `TBScreenComponent` while we develop other components. So, we instantiate it in `initialize` method and store it in the instance variable `main`.

```
TBApplicationRootComponent >> initialize
   super initialize.
   main := TBScreenComponent new.
```

```
TBApplicationRootComponent >> renderContentOn: html
   html render: main
```

We declare the containment relationship by returning `main` as children of `TBApplicationRootComponent`.

```
TBApplicationRootComponent >> children
   ^ { main }
```

If you refresh your application in a web browser, you should obtain what is depicted on Figure 1.8.

Possible Enhancements

The blog name should be customizable using an instance vairable in the `TBBlog` class and the application header component should display this title.

1.11 List of Posts

We will now display the list of all posts - which is the primary goal in fact. Remember that we speak about the public access to the blog here and not the administration interface that will be developped later.

Let's create a `TBPostsListComponent` inheriting from `TBScreenComponent`:

```
TBScreenComponent subclass: #TBPostsListComponent
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'TinyBlog-Components'
```

Add a temporary `renderContentOn:` method (in the 'rendering' protocole) to test during development (cf. Figure 1.9).

```
TBPostsListComponent >> renderContentOn: html
   super renderContentOn: html.
   html text: 'Blog Posts here !!!'
```

Add this new component in the root application component:

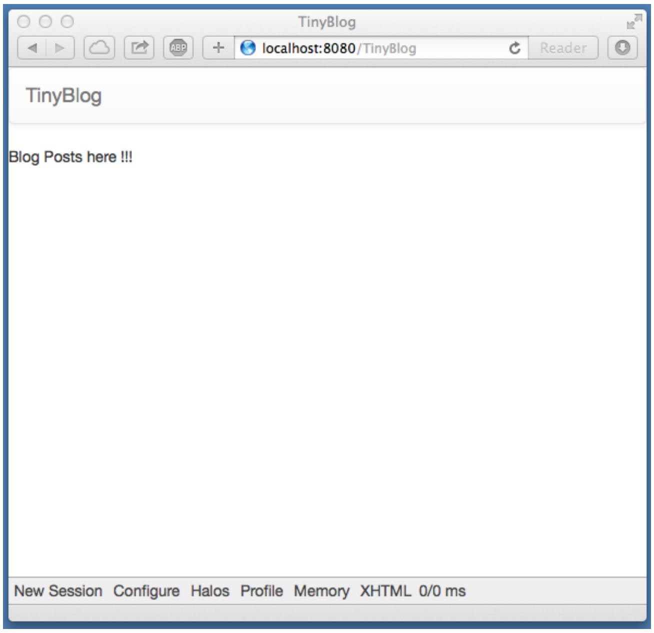
```
TBApplicationRootComponent >> initialize
   super initialize.
   main := TBPostsListComponent new.
```

Modifying this method is not a good practice so we add a setter method to dynamically change the component to display in the future. But, we keep that by default the `TBPostsListComponent` component will be displayed.

```
TBApplicationRootComponent >> main: aComponent
   main := aComponent
```

1.12 A Post Component

Now we will define `TBPostComponent` to display the details of a post. Each post will be graphically displayed by an instance of `TBPostComponent` which will show the post title, its date and its content.



Blog Posts here !!!

Figure 1.9 TinyBlog with a First List of Posts Component.

```
WAComponent subclass: #TBPostComponent
   instanceVariableNames: 'post'
   classVariableNames: ''
   package: 'TinyBlog-Components'
```

```
TBPostComponent >> initialize
      super initialize.
      post := TBPost new.
```

```
TBPostComponent >> title
   ^ post title
```

```
TBPostComponent >> text
   ^ post text
```

```
TBPostComponent >> date
   ^ post date
```

The `renderContentOn:` method defines the HTML rendering of a post.

```
TBPostComponent >> renderContentOn: html
   html heading level: 2; with: self title.
   html heading level: 6; with: self date.
   html text: self text
```

About HTML Forms

Next week, we will develop the administration web UI using Magritte and this will demonstrate that it is not common at all to manually develop components as we explained above. Indeed, Magritte is used to describe model data and then offers automatic Seaside component generators. As we will see, all the code of the previous section could be done just with:

```
TBPostComponent >> renderContentOn: html
   "DON'T WRITE THIS YET"
   html render: post asComponent
```

1.13 Display Posts

To display all visible posts in the database, we just need to modify the `TBPostsListComponent >> renderContentOn:` method:

```
TBPostsListComponent >> renderContentOn: html
   super renderContentOn: html.
   self blog allVisibleBlogPosts do: [ :p |
      html render: (TBPostComponent new post: p) ]
```

Refresh you web browser and you should get an error.

1.14 Debugging Errors

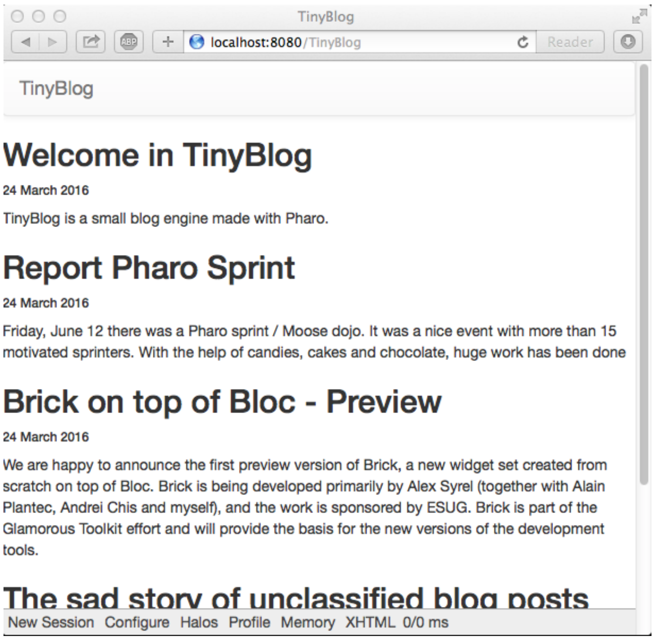
By default, when an error occurs in a web application, Seaside returns an HTML page with the error message. You can change this message or during development, you can configure Seaside to open a debugger directly in Pharo IDE. To configure Seaside, just execute the following snippet:

```
(WAAdmin defaultDispatcher handlerAt: 'TinyBlog')
    exceptionHandler: WADebugErrorHandler
```

Now, if you refresh the web page in your browser, a debugger should open on Pharo side. If you analyze the stack, you should see that we forgot to define the following method:

```
TBPostComponent >> post: aPost
    post := aPost
```

You can define this method in the debugger using the `Create` button. After that, press the `Proceed` button. The web application should now correctly renders what is shown in Figure 1.10.



Welcome in TinyBlog
24 March 2016
TinyBlog is a small blog engine made with Pharo.

Report Pharo Sprint
24 March 2016
Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

Brick on top of Bloc - Preview
24 March 2016
We are happy to announce the first preview version of Brick, a new widget set created from scratch on top of Bloc. Brick is being developed primarily by Alex Syrel (together with Alain Plantec, Andrei Chis and myself), and the work is sponsored by ESUG. Brick is part of the Glamorous Toolkit effort and will provide the basis for the new versions of the development tools.

The sad story of unclassified blog posts

Figure 1.10 TinyBlog with a List of Posts.

1.15 Displaying the List of Posts with Bootstrap

Let's use Bootstrap to make the list of posts more beautiful using a Bootstrap container.

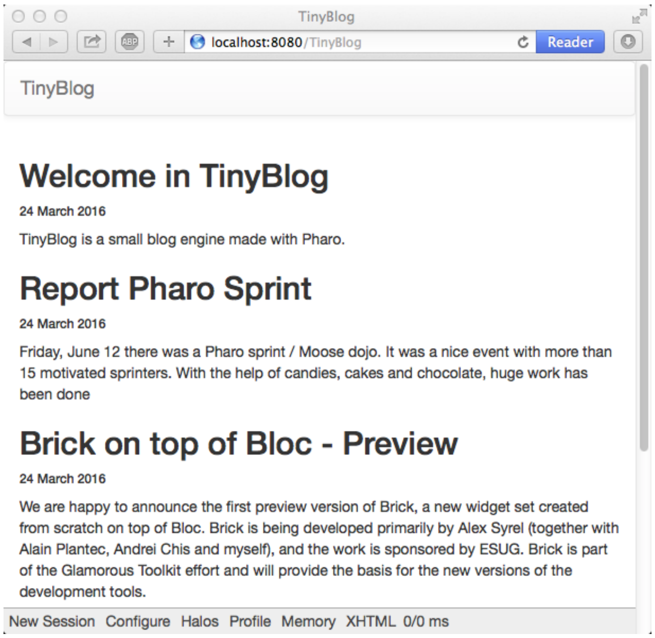
```
TBPostsListComponent >> renderContentOn: html
   super renderContentOn: html.
   html tbsContainer: [
   self blog allVisibleBlogPosts do: [ :p |
      html render: (TBPostComponent new post: p) ] ]
```

Your web application should look like Figure 1.11.

1.16 Displaying Posts by Category

Posts are classified into categories. By default, a post is classified into a special category named "Unclassified" if nothing is specified.

We will now create a new component named `TBCategoriesComponent` to manage the list of categories.



Welcome in TinyBlog

24 March 2016

TinyBlog is a small blog engine made with Pharo.

Report Pharo Sprint

24 March 2016

Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

Brick on top of Bloc - Preview

24 March 2016

We are happy to announce the first preview version of Brick, a new widget set created from scratch on top of Bloc. Brick is being developed primarily by Alex Syrel (together with Alain Plantec, Andrei Chis and myself), and the work is sponsored by ESUG. Brick is part of the Glamorous Toolkit effort and will provide the basis for the new versions of the development tools.

Figure 1.11 TinyBlog with a Bootstrap List of Posts.

Category Component Definition

`TBCategoriesComponent` is responsible to display the list of all categories available in the database (the model) and also to select one category. This component should be able to communicate with the `TBPostsListComponent` to pass it the currently selected category.

```
WAComponent subclass: #TBCategoriesComponent
   instanceVariableNames: 'categories postsList'
   classVariableNames: ''
   package: 'TinyBlog-Components'

TBCategoriesComponent >> categories
   ^ categories

TBCategoriesComponent >> categories: aCollection
   categories := aCollection

TBCategoriesComponent >> postsList: aComponent
      postsList := aComponent

TBCategoriesComponent >> postsList
   ^ postsList
```

On class-side, we define a creation method.

```
TBCategoriesComponent class >> categories: aCollectionOfCategories
    postsList: aTBScreen
   ^ self new categories: aCollectionOfCategories; postsList:
    aTBScreen
```

The `selectCategory:` method (protocol 'action') communicates the currently selected category to the `TBPostsListComponent`.

```
TBCategoriesComponent >> selectCategory: aCategory
   postsList currentCategory: aCategory
```

In `TBPostsListComponent`, we now add an instance variable to store the current category.

```
TBScreenComponent subclass: #TBPostsListComponent
   instanceVariableNames: 'currentCategory'
   classVariableNames: ''
   package: 'TinyBlog-Components'
```

```
TBScreenComponent >> currentCategory
   ^ currentCategory

TBScreenComponent >> currentCategory: anObject
   currentCategory := anObject
```

1.17 Category Rendering

We add a rendering method ('rendering' protocol) to display one category. A category is rendered as a link and that makes this category the current one if the user clicks on it.

```
TBCategoriesComponent >> renderCategoryLinkOn: html with: aCategory
   html tbsLinkifyListGroupItem
      callback: [ self selectCategory: aCategory ];
      with: aCategory
```

Finally, the `renderContentOn:` method of the `TBCategoriesComponent` is now straighforward, it just iterates over all categories and renders them using bootstrap brushes:

```
TBCategoriesComponent >> renderContentOn: html
   html tbsListGroup: [
      html tbsListGroupItem
         with: [  html strong: 'Categories' ].
      categories do: [ :cat |
         self renderCategoryLinkOn: html with: cat ]]
```

It remains to display the list of categories in the root application component and refresh the list of displayed posts regarding the currently selected category.

1.18 Refreshing the List of Posts

We have to modify some methods of `TBPostsListComponent`.

First, the `readSelectedPosts` method returns the posts to display. If the current category is `nil`, it means that the user did not select any category yet and all visible posts of the database are displayed. If the user has selected a category, only posts of this category are displayed.

```
TBPostsListComponent >> readSelectedPosts
   ^ self currentCategory
      ifNil: [ self blog allVisibleBlogPosts ]
      ifNotNil: [ self blog allVisibleBlogPostsFromCategory: self
    currentCategory ]
```

Then, the method that renders the list of posts can be modified as follows:

```
TBPostsListComponent >> renderContentOn: html
   super renderContentOn: html.
   html render: (TBCategoriesComponent
               categories: (self blog allCategories)
               postsList: self).
   html tbsContainer: [
      self readSelectedPosts do: [ :p |
         html render: (TBPostComponent new post: p) ] ]
```

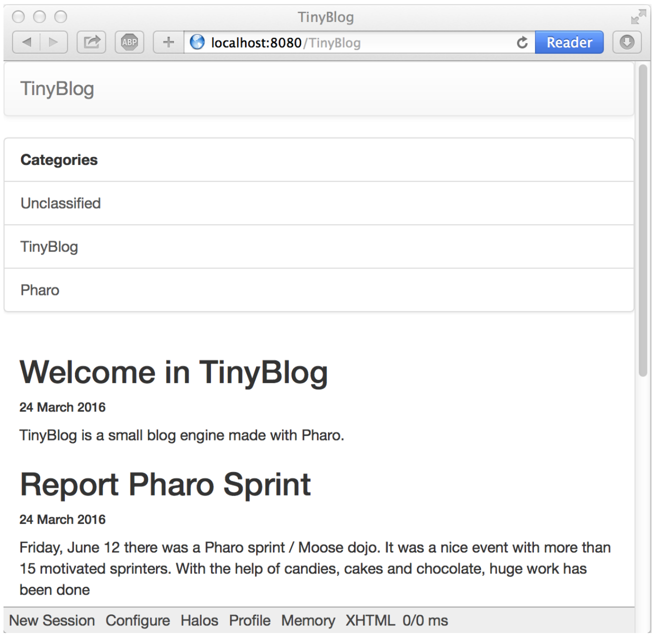
An instance of `TBCategoriesComponent` is rendered on the web page and allows users to select a category (See Figure 1.12).

1.19 Look and Layout

We will improve the layout of `TBPostsListComponent` using a repsonsive design for the list of posts. It means that the CSS will adapt the component size and placement based on the available space.

Components are displayed inside two Bootstrap containers in a row and two columns. Column width is determined according to the resolution (viewport) of the displaying device. The 12 columns of the Bootstrap grid are splitted between the list of categories and list of posts. If a low resolution is used, the list of categories will be above the list of posts (each lists will occupy 100% of the width of the container).

```
TBPostsListComponent >> renderContentOn: html
   super renderContentOn: html.
   html tbsContainer: [
      html tbsRow showGrid;
         with: [
            html tbsColumn
               extraSmallSize: 12;
               smallSize: 2;
```



TinyBlog

Categories
Unclassified
TinyBlog
Pharo

Welcome in TinyBlog
24 March 2016
TinyBlog is a small blog engine made with Pharo.

Report Pharo Sprint
24 March 2016
Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

New Session Configure Halos Profile Memory XHTML 0/0 ms

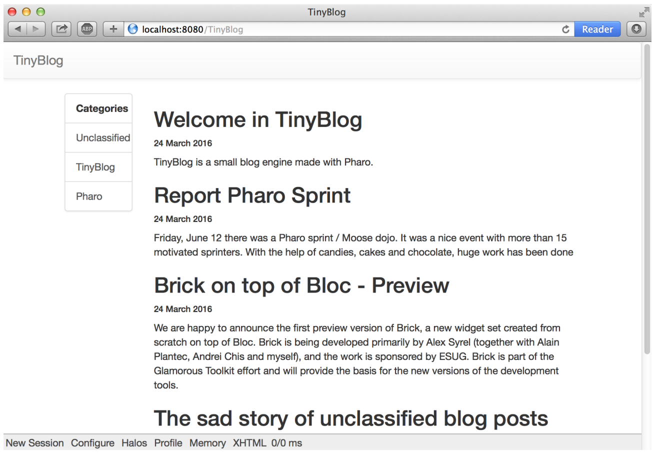
Figure 1.12 Select a Category to Filter Posts.

```
                mediumSize:  4;
                with: [
                   html render: (TBCategoriesComponent
                     categories: (self blog allCategories)
                     postsList: self) ].
       html tbsColumn
                extraSmallSize: 12;
                smallSize: 10;
                mediumSize: 8;
                with: [
          self readSelectedPosts do: [ :p |
              html render: (TBPostComponent new post: p) ] ] ] ]
```

You should now obtain the same look as in Figure 1.13.

When you select a category, the list of posts is correctly refreshed. However, the current category is not highlighted. To introduce this feature, we modify the following method:

```
TBCategoriesComponent >> renderCategoryLinkOn: html with: aCategory
   html tbsLinkifyListGroupItem
      class: 'active' if: aCategory = self postsList currentCategory
     ;
      callback: [ self selectCategory: aCategory ];
      with: aCategory
```



TinyBlog

Categories
Unclassified
TinyBlog
Pharo

Welcome in TinyBlog

24 March 2016

TinyBlog is a small blog engine made with Pharo.

Report Pharo Sprint

24 March 2016

Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

Brick on top of Bloc - Preview

24 March 2016

We are happy to announce the first preview version of Brick, a new widget set created from scratch on top of Bloc. Brick is being developed primarily by Alex Syrel (together with Alain Plantec, Andrei Chis and myself), and the work is sponsored by ESUG. Brick is part of the Glamorous Toolkit effort and will provide the basis for the new versions of the development tools.

The sad story of unclassified blog posts

Figure 1.13 TinyBlog with a Responsive Design.

The application works well but you must not keep the current implementation of `TBPostsListComponent >> renderContentOn:`. This method is too long and cannot be easily reused. Propose a solution.

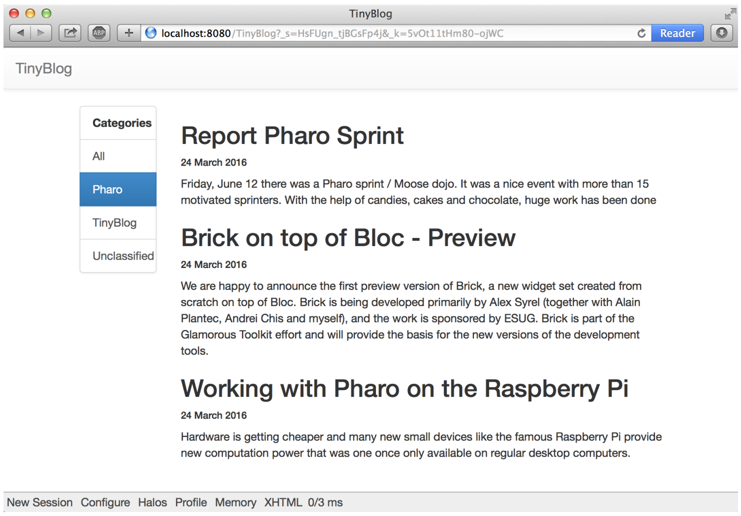
1.20 Our Solution: Small Methods

```
TBPostsListComponent >> renderContentOn: html
   super renderContentOn: html.
   html
      tbsContainer: [
         html tbsRow
            showGrid;
            with: [ self renderCategoryColumnOn: html.
                  self renderPostColumnOn: html ] ]

TBPostsListComponent >> renderCategoryColumnOn: html
   html tbsColumn
      extraSmallSize: 12;
      smallSize: 2;
      mediumSize: 4;
      with: [ self basicRenderCategoriesOn: html ]

TBPostsListComponent >> basicRenderCategoriesOn: html
   ^ html render: (TBCategoriesComponent
         categories: self blog allCategories
         postsList: self)

TBPostsListComponent >> renderPostColumnOn: html
   html tbsColumn
```


TinyBlog

localhost:8080/TinyBlog?_s=HsFUgn_tjBGsFp4j&_k=5vOt11tHm80-ojWC

TinyBlog

Categories
- All
- Pharo
- TinyBlog
- Unclassified

Report Pharo Sprint

24 March 2016

Friday, June 12 there was a Pharo sprint / Moose dojo. It was a nice event with more than 15 motivated sprinters. With the help of candies, cakes and chocolate, huge work has been done

Brick on top of Bloc - Preview

24 March 2016

We are happy to announce the first preview version of Brick, a new widget set created from scratch on top of Bloc. Brick is being developed primarily by Alex Syrel (together with Alain Plantec, Andrei Chis and myself), and the work is sponsored by ESUG. Brick is part of the Glamorous Toolkit effort and will provide the basis for the new versions of the development tools.

Working with Pharo on the Raspberry Pi

24 March 2016

Hardware is getting cheaper and many new small devices like the famous Raspberry Pi provide new computation power that was one once only available on regular desktop computers.

New Session Configure Halos Profile Memory XHTML 0/3 ms

Figure 1.14 Final TinyBlog Public UI.

```
          extraSmallSize: 12;
          smallSize: 10;
          mediumSize: 8;
          with: [ self basicRenderPostsOn: html ]

TBPostsListComponent >> basicRenderPostsOn: html
   ^ self readSelectedPosts do: [ :p |
         html render: (TBPostComponent new post: p) ]
```

We are now ready to define a administrative UI for TinyBlog.

1.21 Possible Enhancements

For example, you can:

- sort categories by name
- add a link named 'All' in the list of categories to display all visible posts regardless of their category

Figure 1.14 shows the final application you may have developped.