VAADIN TRAINING

Router API

Defining Routes with @Route

No more views, any component can be a route target, as long as it has @Route annotation.

- @Route("") defines a default route target.
- @Route also takes arbitrary path as parameter, e.g. @Route("some/path")

@RouteAlias

Additional route paths to a view can be defined with @RouteAlias, so that have multiple paths for the same view, e.g. you might want your HomeView to be accessible with both localhost:8080/home and localhost:8080,

JAVA

```java
@Route(value = "home")
@RouteAlias("")
public class HomeView
```

URL Parameters for Navigation Target

By implementing HasUrlParameters interface, a component can take url parameters. In the example below, when you go to yourdomain.com/greeting/vaadin, you will get Welcome, Vaadin!

JAVA

```java
@Route(value = "greet")
public class GreetingComponent extends Div implements HasUrlParameter<String> {
    @Override
    public void setParameter(BeforeNavigationEvent event,
            String parameter) {
        setText(String.format("Welcome %s.", parameter));
    }
}
```

URL parameters can be optional, by using the @OptionalParameter.

JAVA

```java
@Route(value = "greet")
public class OptionalComponent extends Div implements HasUrlParameter<String> {
    @Override
    public void setParameter(BeforeNavigationEvent event,
            @OptionalParameter String parameter) {
        if(parameter == null) {
            setText("Welcome anonymous.");
        } else {
            setText(String.format("Welcome %s.", parameter));
        }
    }
}
```

In case more parameters are wanted, e.g. yourdomain.com/greeting/one/two/three, the URL parameter can also be annotated with @WildcardParameter. Note that the parameter for Wildcard parameter can never be null.

JAVA

```java
@Route(value = "greet")
public class WildCardComponent extends Div implements HasUrlParameter<String> {
    @Override
    public void setParameter(BeforeNavigationEvent event,
            @WildcardParameter String parameter) {
        if(parameter.isEmpty) {
            setText("Welcome anonymous.");
        } else {
            setText(String.format("Handling parameter %s.", parameter));
        }
    }
}
```

More specific paths will have precedence over optional and wildcard target. In the example below, TrainingComponent will be matched instead of OptionalComponent or WildcardComponent, when user go to location yourdomain.com/greet/training

JAVA

```java
@Route(value = "greet/training")
public class TrainingComponent extends Div{
    public GreetingComponent(){
        setText("Welcome to online training"));
    }
}
```

Router exception Handling

- You can define some navigation targets that are intended to show user some error message when something went wrong.
- They would work the same way as normal navigation targets except that they usually do not have any specific @Route
- Such error navigation targets can be defined by implementing HasErrorParameter<T extends Exception>
- Exception matching will run first by exception cause and then by exception super type
- Note: One exception may only have one exception handler

JAVA

```java
@Tag("div")
public class RouteNotFoundError extends Component
        implements HasErrorParameter<NotFoundException> {

    @Override
    public int setErrorParameter(BeforeNavigationEvent event,
            ErrorParameter<NotFoundException> parameter) {
        getElement().setText(
                "Could not navigate to '" + event.getLocation().getPath() + "'");
        return 404;
    }
}
```

Navigation

The easiest way to trigger a navigation is to use RouterLink

JAVA
```
Div menu = new Div();
menu.add(new RouterLink("Home", HomeView.class))
```

In the browser, a router link is just a normal html link with "router-link" attribute. The difference is that a normal link will cause the page to reload while a RouterLink will be handled by Vaadin Framework and won't reload the page.

Note that RouterLink only works for navigation inside a website or web application, for external navigation, you can use Anchor.

JAVA
```
new RouterLink("https://vaadin.com", "Vaadin");
```

Navigation can also be triggered with UI.navigate(), it accepts String or navigation target class as parameter.

JAVA
```
Button button = new Button("Navigate to company");
button.addClickListener( e-> {
     button.getUI().ifPresent(ui -> ui.navigate(“company”));
     //OR
     button.getUI().ifPresent(ui -> ui.navigate(CompanyView.class));
});
```

URL generation

Router exposes methods to get navigation URL for registered navigation targets. You can get the Router from UI by using UI.getCurrent().getRouter(). Note that it only returns the URL after domain name, context path.

JAVA
```
@Route("path")
public class PathComponent extends Div {…}
//returns ‘path’
String route = UI.getCurrent().getRouter().get().getUrl(PathComponent.class);
```

It also works with parameters

JAVA
```
@Route("path")
public class PathComponent extends Div implements HasUrlParameter{…}
//returns ‘path/test’
String route = UI.getCurrent().getRouter().get().getUrl(PathComponent.class,"test");
```

To get the full url, you can use VaadinServletRequest.

JAVA
```
StringBuffer url = VaadinServletRequest.getCurrent().getrequestURL();
```

Page Title

Static page title can be configured with @PageTitle.

JAVA

```java
@PageTitle("home")
class HomeView extends Div {

  HomeView(){
    setText("This is the home view");
  }
}
```

Page title can also be dynamically updated by implementing HasDynamicTitle interface.

JAVA

```java
@Route(value = "blog")
class BlogPost extends Component implements HasDynamicTitle, HasUrlParameter<Long> {
  private String title = "";

  @Override
  public String getPageTitle() {
    return title;
  }

  @Override
  public void setParameter(BeforeNavigationEvent event, @OptionalParameter Long
parameter) {
    if (parameter != null) {
      title = "Blog Post #" + parameter;
    } else {
      title = "Blog Home";
    }
  }
}
```

Note that previous two approached only update the page title upon navigation. To change the page tile at run time, use

JAVA

```java
UI.getCurrent().getPage().setTitle();
```

Application Layout

When defining route using @Route("path"), the component will by default be rendered inside <body> element of the page. Most of the time this is not what you want, normally you will have an application layout, e.g. a header on the top, a footer at the bottom, a menu bar on the left and child view on the right, and the route target should be rendered inside the child view.

You can define a parent layout with the "layout" attribute in the @Route annotation. When navigating between components that use the same parent, the parent will be re-used and won't update during the navigation.

JAVA

```java
@Route(value="company", layout=MainLayout.class)
public class CompanyComponent extends Component {
}
```

Each layout used as a parent layout must implement the RouterLayout interface.

JAVA

```java
public class MainLayout extends Div implements RouterLayout{
}
```

RouterLayout has a default method showRouterLayoutContent(), which will append the content as its child. Override the showRouterLayoutContent() according to your need.

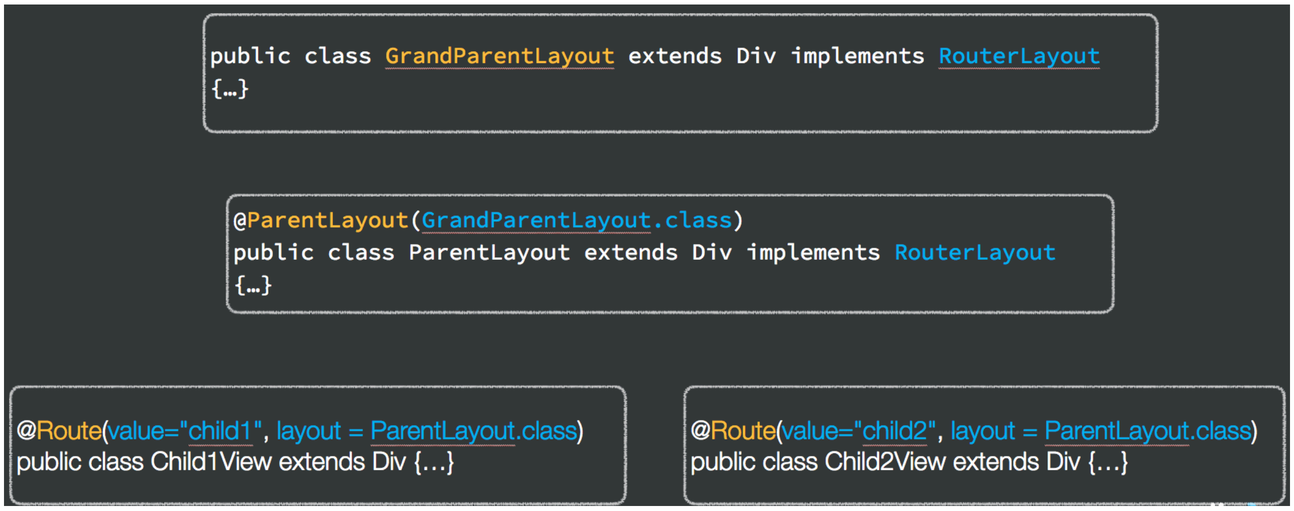
JAVA

```java
public interface RouterLayout extends HasElement {

   default void showRouterLayoutContent(HasElement content) {
       if (content != null) {
           getElement().appendChild(Objects.requireNonNull(content.getElement()));
       }
   }
}
```

JAVA

```java
public class MainLayout extends VerticalLayout implements RouterLayout {
   private Div childWrapper = new Div();

   @Override
   public void showRouterLayoutContent(HasElement content) {
       childWrapper.getElement().appendChild(content.getElement());
   }
}
```

Multiple layers of nested layout is also supported, The intermediate layers can be annotated with @ParentLayout to indicate the hierarchy. There is no restrictions on the amount of nested layouts. The class that has @ParentLayout annotation should be a RouterLayout.



```java
public class GrandParentLayout extends Div implements RouterLayout
{…}
```

```java
@ParentLayout(GrandParentLayout.class)
public class ParentLayout extends Div implements RouterLayout
{…}
```

```java
@Route(value="child1", layout = ParentLayout.class)
public class Child1View extends Div {…}
```

```java
@Route(value="child2", layout = ParentLayout.class)
public class Child2View extends Div {…}
```

Navigation Lifecycle

There are 3 events can be observed during navigation lifecycle:

- BeforeEnterEvent
- BeforeLeaveEvent
- AfterNavigationEvent

BeforeEnterEvent

- BeforeEnterEvent is fired when a component is being attached.
- Can be observed with BeforeEnterObserver
- Can be used to reroute dynamically.

JAVA

```java
@Route("no-items")
public class NoItemsView extends Div {
  public NoItemsView() {
    setText("No items found.");
  }
}
@Route("blog")
public class BlogList extends Div implements BeforeEnterObserver {
  @Override
  public void beforeEnter(BeforeEnterEvent event) {
    if (getItem() == null) {
      event.rerouteTo(NoItemsView.class);
    }
  }
}
```

You can also reroute to an error view registered for an exception from BeforeEnterEvent and BeforeLeaveEvent.

JAVA

```java
public class AuthenticationHandler implements BeforeNavigationObserver {
    @Override
    public void beforeEnter(BeforeEnterEvent event) {
        Class<?> target = event.getNavigationTarget();
        if (!currentUserMayEnter(target)) {
            event.rerouteToError(AccessDeniedException.class);
        }
    }
}
```

BeforeLeaveEvent

- BeforeLeaveEvent is fired when a component is being detached.
- Can be observed with BeforeLeaveObserver
- Can be used for postpone purpose.

JAVA

```
public class SignupForm extends Div implements BeforeLeaveObserver {
    @Override
    public void beforeLeave(BeforeLeaveEvent event) {
        if (this.hasChanges()) {
            ContinueNavigationAction action = event.postpone();
            Dialog confirmDialog = new Dialog();
            Button confirmButton = new Button("Confirm", e -> {
                   action.proceed();
                   confirmDialog.close(); });
            confirmDialog.open();
        }
    }
}
```

AfterNavigationEvent

- AfterNavigationEvent can be observed with AfterNavigationObserver.
- No further redirects, you can safely use the location returned by the AfterNavigationEvent.
- A good use case is when you need to update the menu selections.

JAVA

```
public class SideMenu extends Div implements AfterNavigationObserver {
    Anchor blog = new Anchor("blog", "Blog");

    @Override
    public void afterNavigation(AfterNavigationEvent event) {
        boolean active = event.getLocation().getFirstSegment()
                .equals(blog.getHref());
        blog.getElement().getClassList().set("active", active);
    }
}
```

Query Parameters

You can also use query parameters in Vaadin. You navigation target doesn't need to implement anything, To navigate to you navigation target with query parameters, you can use QueryParameters object together with the location. e.g.

JAVA

```java
Button button = new Button("Navigate to my view");
button.addClickListener( e-> {
  Map<String, List<String>> parameters = new HashMap();
  parameters.put("param", "value");
  QueryParameters queryParameters = new QueryParameters(parameters);
  button.getUI().ifPresent(ui -> ui.navigate("myView", queryParameters));
});
```

Or you can also generate the location url from Router in order to avoid string literals.

JAVA

```java
Button button = new Button("Navigate to my view");
button.addClickListener( e-> {
  Map<String, List<String>> parameters = new HashMap();
  parameters.put("param", Arrays.asList("value"));
  QueryParameters queryParameters = new QueryParameters(parameters);
  String viewUrl = UI.getCurrent().getRouter().getUrl(MyView.class);
  button.getUI().ifPresent(ui -> ui.navigate(viewUrl, queryParameters));
});
```

Now the only question left is how to receive the query parameters. You can safely get the query parameters from AfterNavigationEvent, as there won't be url redirections anymore.

JAVA

```java
public class MyView extends Div implements AfterNavigationObserver {

    @Override
    public void afterNavigation(AfterNavigationEvent event) {
        QueryParameters queryParameters = event.getLocation().getQueryParameters();
        …
    }
}
```