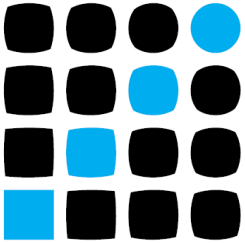


adaptTo()

APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 28-30 SEPTEMBER 2015

Making Sling Grunt

Or How to Integrate Modern Front-End Development with Sling

Philip Hornig (Publicis Pixelpark), Michael Sunaric (Netcentric)

Topics

- Modern front-end development and why we need it.
- How to integrate modern front-end development with the Sling development stack.
- The tools necessary to achieve the integration.

Modern Front-End Development

Complexity is increasing

- Complex layouts and responsive design
- Client side apps
- Modular and object oriented design
- Automated testing
- Documentation

Agile development

- Rapid prototyping
- Design in HTML

- Prototyping engine with node.js, express.js

Setup

Scaffolding

Yeoman, Middleman, …

Libraries

jQuery, Bootstrap, Modernizr, Bourbon

Frameworks

Ember.js, AngularJS, Backbone.js,
ExtJS, Dojo, ...

Develop

Watch

CSS (Sass, Less, Stylus)
Javascript (CoffeeScript, TypeScript, ECMAScript 6)
HTML (Jade, Haml, Handlebars)

Lint

CSS (csslint, sasslint, styluslint)
Javascript (jshint, jscs)

Refresh

LiveReload

Prototype

Node.js, Express

Test

Function

PhantomJS, CasperJS, Selenium

Form

Selenium, BackstopJS

Build

1. Code linting
2. Compile
3. Unit tests
4. Concatenate
5. Minify
6. Generate icons/iconfonts
7. Optimize images
8. Measure performance
9. Prototype
10. Integration tests
11. Deploy

Build Tools and Dependency Mgmt.

- NPM, Bower, Gulp, Grunt, Broccoli

Gruntfile.js

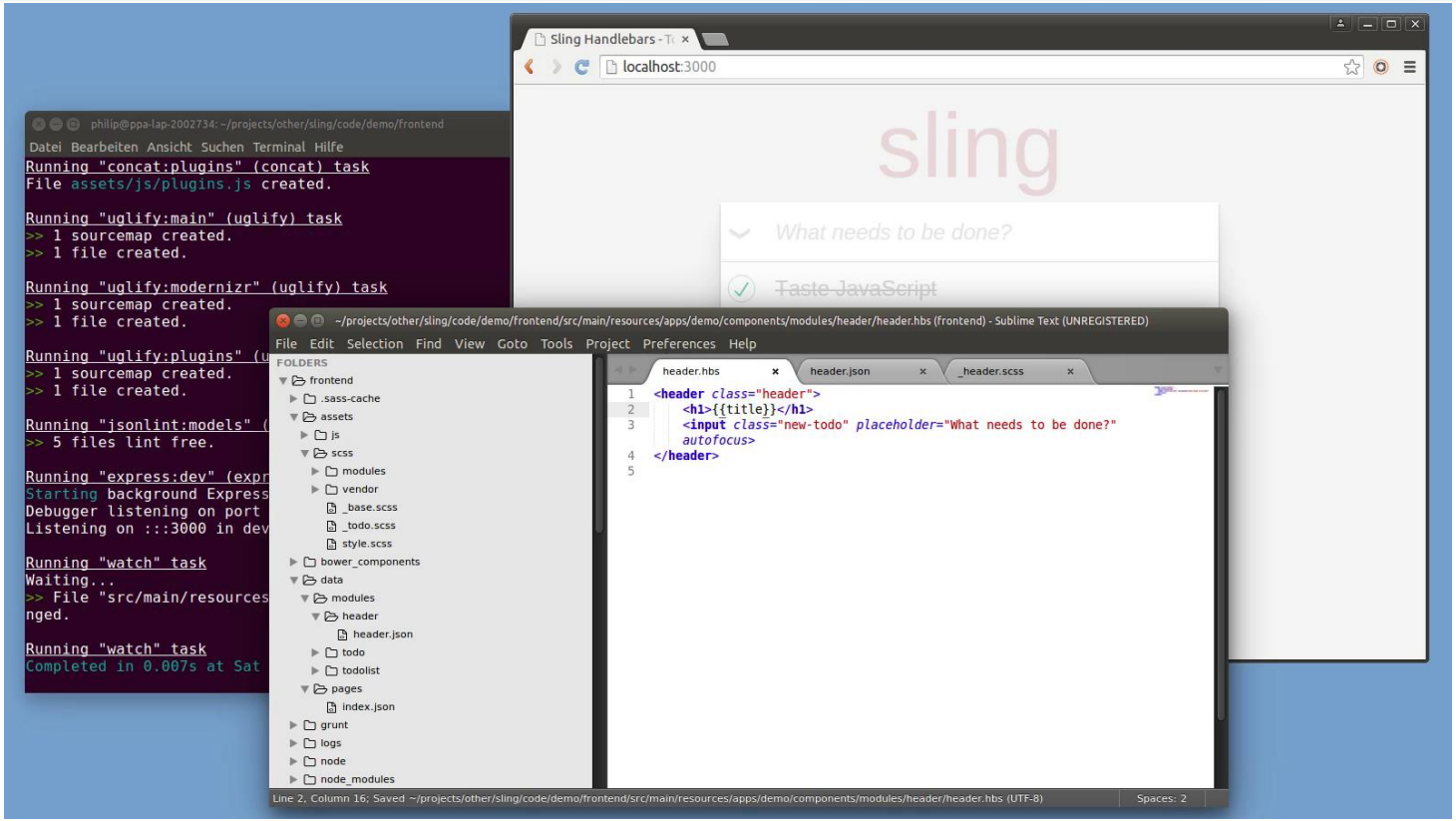
```
module.exports = function (grunt) {
    "use strict";
    require("load-grunt-config")(grunt, {
        init: true,
        data: {
            assetsSrc: "assets",
            assetsDist: "src/main/resources/apps/demo/assets"
        }
    });

    grunt.registerTask("default", ["clean", "bowercopy", "scss", "js", "jsonlint:models"]);
    grunt.registerTask("dev", ["default", "express:dev", "watch"]);
    grunt.registerTask("scss", ["scsslint", "sass"]);
    grunt.registerTask("js", ["jscs", "jshint", "concat", "uglify"]);
};
```

Sass Task

```
module.exports = function (grunt) {
    "use strict";
    return {
        options: {
            quiet: true,
            sourcemap: "inline",
            unixNewlines: true,
            trace: true
        },
        all: {
            options: {
                style: "expanded"
            },
            src: "<%= assetsSrc %>/scss/style.scss",
            dest: "<%= assetsDist %>/css/style.css"
        }
    };
};
```

Working with Grunt



How to integrate with backend development

How integration used to be

A) Code duplication

Front-end delivers HTML prototype, back-end developers recreate templates in back-end (e.g. jsp)

B) Back-end tool chain only

Front-end developer works with back-end tools/development environment to create front-end code.

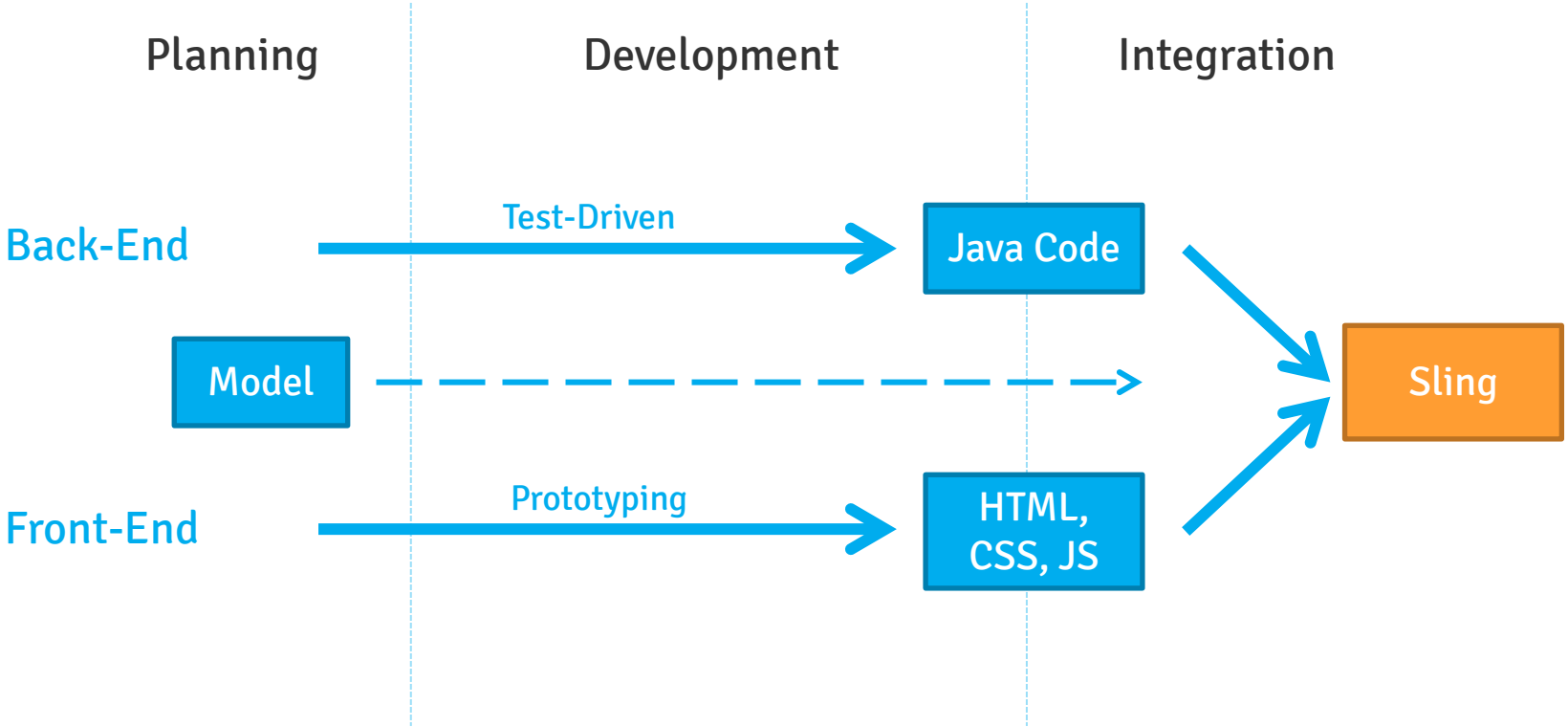
Pitfalls of “code duplication”

- Extra work, more code to maintain, error prone
- Eventually front-end code and back-end code diverge.

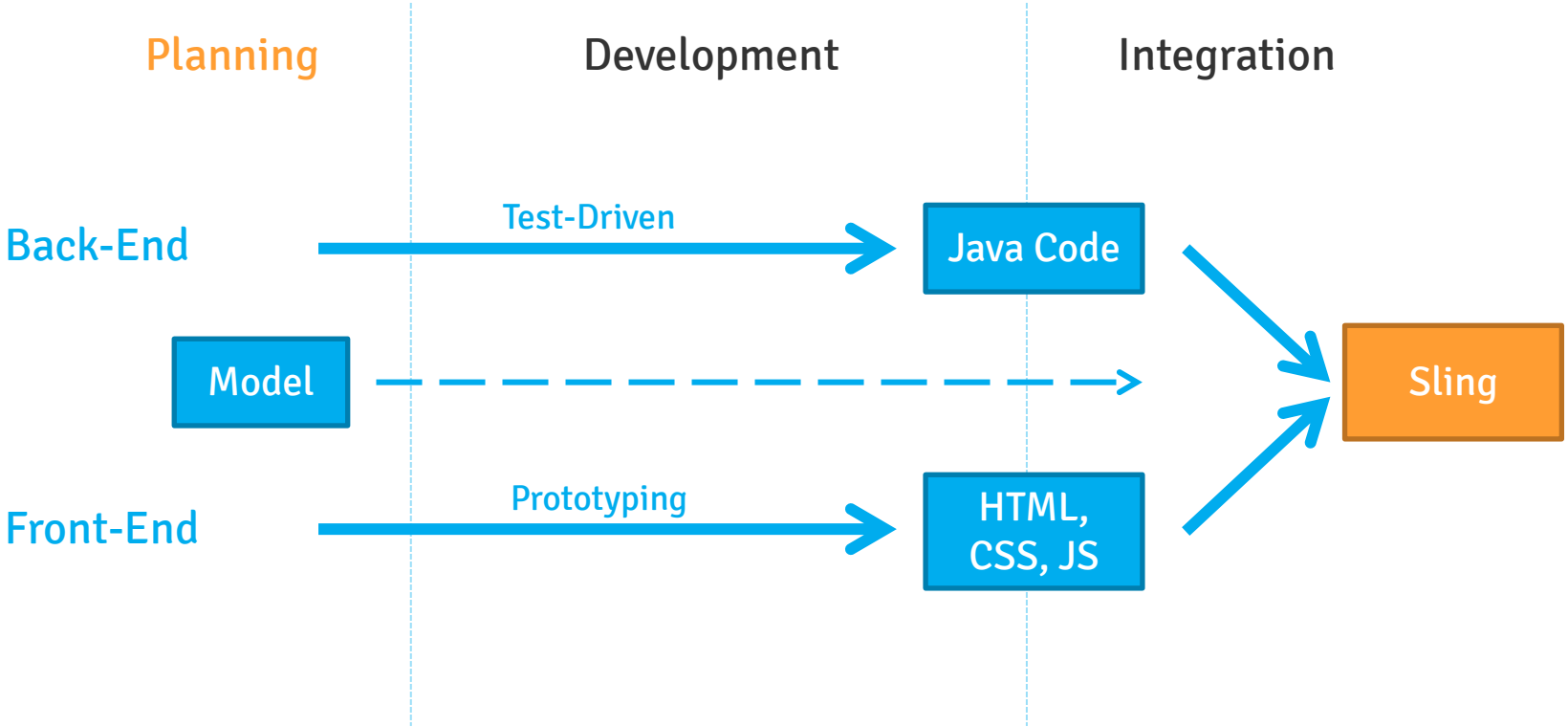
Pitfalls of “back-end tool chain only”

- Not compatible with modern front-end stack
- Each discipline requires more specialized knowledge.
- Usage of full back-end stack for front-end development is time consuming.

How we would like the process to be like



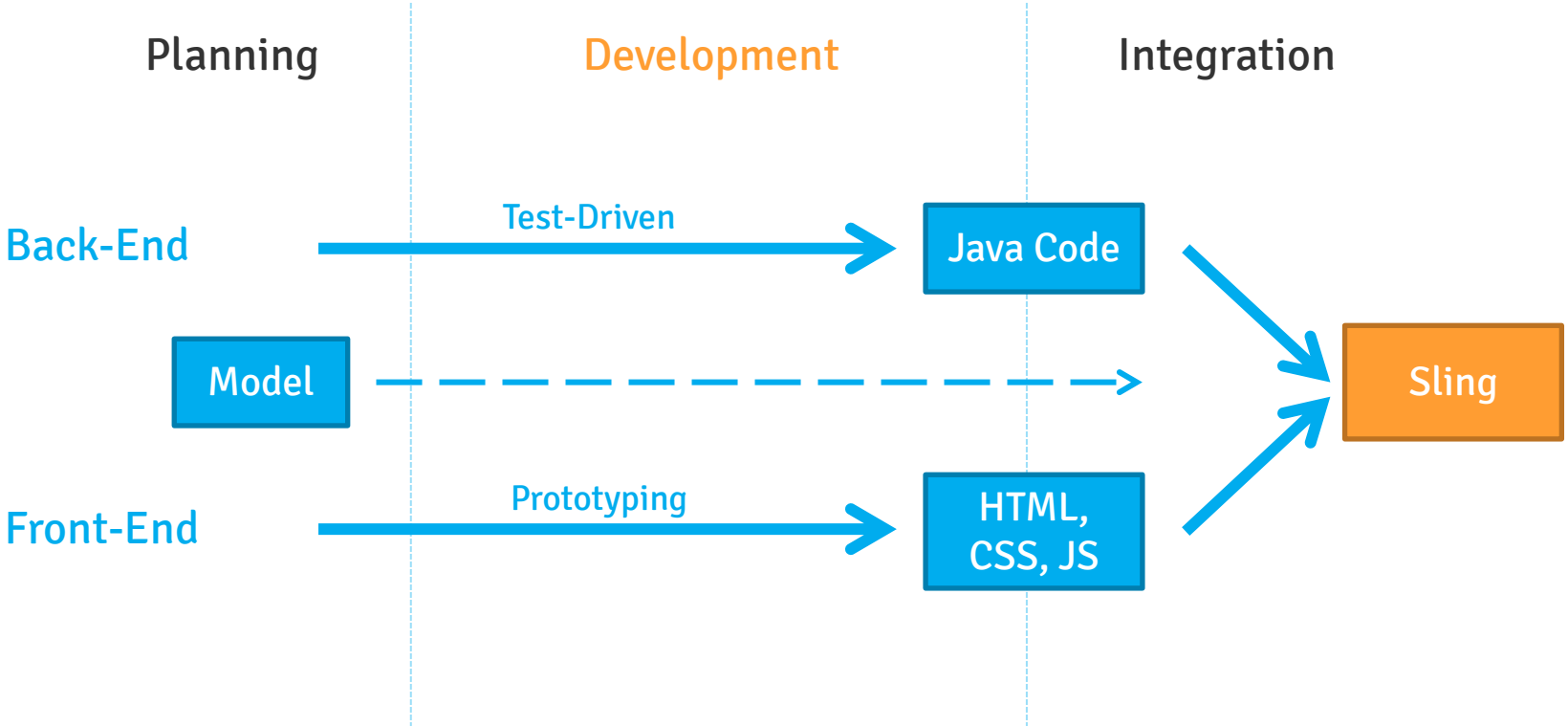
How we would like the process to be like



Planning Phase

- The interface between front-end and back-end is the data model
- The data model is defined in a JSON file
- The JSON file serves as documentation of the interface

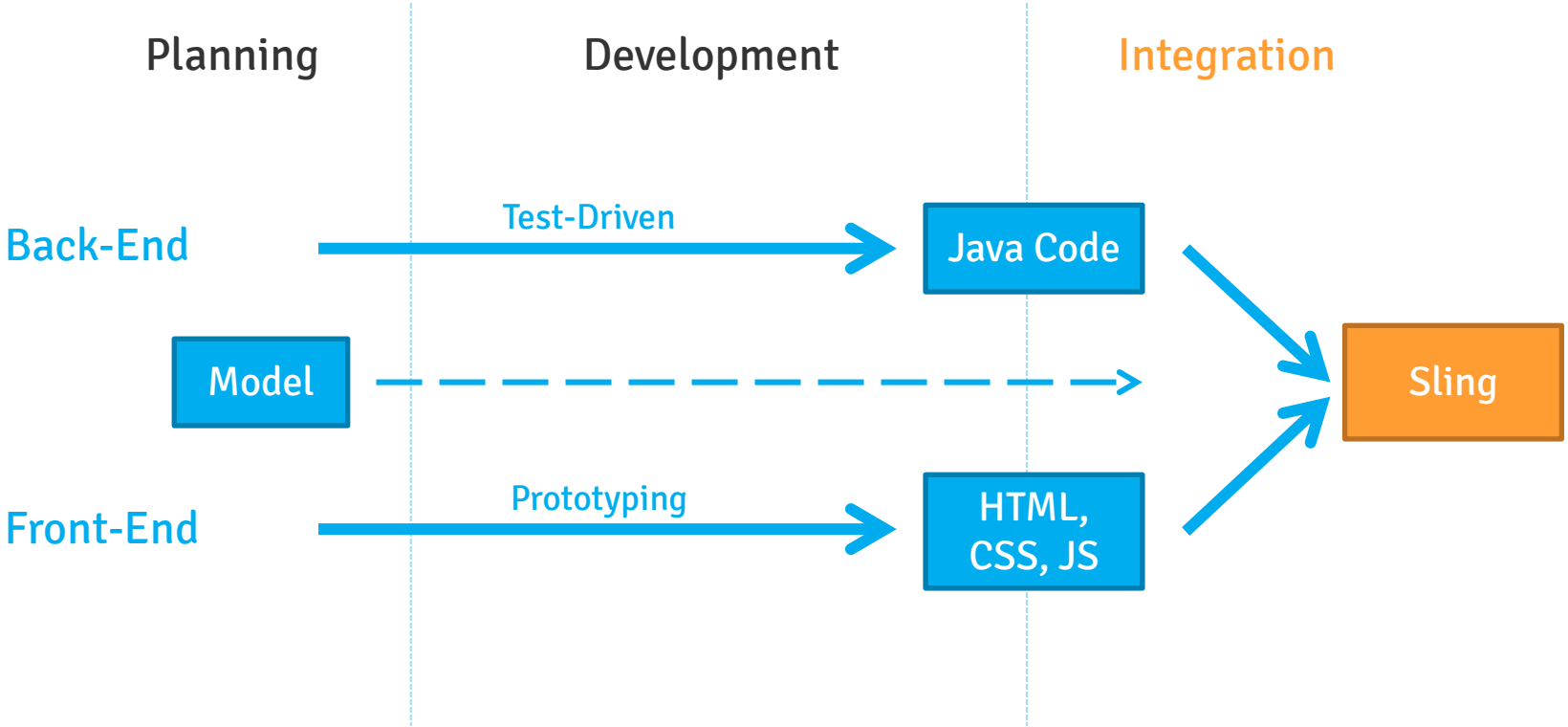
How we would like the process to be like



Development Phase

- Now front-end and back-end development can start in parallel
- The front-end uses the JSON file in development for the HTML prototype
- The back-end develops test-driven against the data model

How we would like the process to be like



Integration Phase

- In Sling front-end and back-end code is deployed and works together seamlessly.

The Crux

- Touchpoint between front-end and back-end is the template engine

  …which merges the template delivered by the front-end and the data model delivered by the back-end

We need a common template engine!

Unified Template Source

- Template development is front-end driven.
- Templates must be part of the front-end stack.
- Templates must contain only presentation logic.
- Templates must be renderable in Sling.

Our solution

- Front-end stack with **Handlebars** as template language
- Sling uses a **Handlebars Sling Script Engine**

Evaluated Handlebars Sling Script Engines

- AEM Social Component Framework (https://docs.adobe.com/docs/en/aem/6-1/develop/communities/scf.html)
- Handlebars Sling Script Engine by Ian Boston (http://svn.apache.org/repos/asf/sling/whiteboard/ieb/handlebars, see https://issues.apache.org/jira/browse/SLING-2919).

Both use the Handlebars Java port Handlebars.java (https://github.com/jknack/handlebars.java).

AEM Social Component Framework

- Closed source, support for our purpose unclear.
- API and implementation is specific to Social Communities.

Handlebars Script Engine by Ian Boston

- Uses JCR structure as data model.
- For complex applications, the JCR structure will not match the presentation model.

Handlebars Sling Script Engine with Context Generators

Script Engine with Context Generators

- Add “context generators”.
- Context generators create a specific presentation model for each template.

Script Engine with Context Generators - 2

- We implemented two context generators:
  - SimpleContextGenerator
    - Exposes JCR structure as model
  - PresenterContextGenerator

PresenterContextGenerator

- Delegates model generation to a Sling Model bean.
- Bean is assigned by `Presenter` annotation to a `sling:resourceType`.

Template

- Handlebars template

```
<!doctype html>
<html lang="en">
<head>
  <title>{{title}}</title>
  <link rel="stylesheet" href="/demo/assets/css/style.css">
</head>
<body>
  {{include this template="body"}}
  {{include this path="footer" resourceType="demo/components/modules/footer"}}
</body>
</html>
```

Model

- Provide presentation model using Sling Models

```
@Model(adaptables = Resource.class)
@Presenter(resourceTypes = {"demo/components/page"})
public class PagePresenter {
    @ValueMapValue
    @XSSProtection(strategy = XSSProtection.Strategy.HTML_FILTER)
    private String title;

    public String getTitle() {
        return title + " presented by PagePresenter";
    }
}
```

Model generation

- Find the right Presenter and convert it to a Map

```
public Map<String, Object> createModel(ScriptContext scriptContext) {
    Resource resource = new
    ScriptContextAdapter(scriptContext).getResource();
    String resourceType = resource.getResourceType();
    Class<?> presenterType =
    presenterBundleListener.getPresenters().get(resourceType);
    Object presenter = null;
    if (presenterType != null) {
        presenter = resource.adaptTo(presenterType);
    }
    return beanToMapSerializer.convertToMap(presenter);
}
```

Server-Side Rendering

- Render the template with Handlebars.java

```
public Object eval(Reader templateReader, ScriptContext scriptContext)…{
    Handlebars handlebars = new Handlebars();
    …
    Template template = getTemplate(templateReader, scriptContext,
    handlebars);
    Context context = createContext(scriptContext);
    if (context != null) {
      template.apply(context, scriptContext.getWriter());
    }
    …
```

Features

- Handlebars.java HTML-escapes by default.
- Additional, customizable XSS protection by `XSSProtection` annotation.
- Template languages can be mixed (e.g. `body.hbs` may include `parsys.jsp`).

How to use it

Download

https://github.com/phornig/sling-handlebars

Demo

https://github.com/phornig/sling-handlebars-demo

Outlook

- Adding additional Handlebars helper via a Service Factory
- AEM helper plugin (i18N, WCMMode)
- Presenter lookup by Sling selectors

Thank you

Questions?

Download https://github.com/phornig/sling-handlebars

Demo https://github.com/phornig/sling-handlebars-demo

Contact

Michael Sunaric, Netcentric

michael.sunaric@netcentric.biz

Philip Hornig, Publicis Pixelpark

phornig@gmail.com