Georgi Yanev

Product Selector

F-Secure project for the public web

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

30 April 2016

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Georgi Yanev Product Selector: F-Secure project for the public web 34 pages + 1 appendix 30 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kimmo Sauren, Senior Lecturer

The purpose of this thesis is to document and describe in detail the process of developing a medium scale web application. The Product Selector application aims to help customers find the security and privacy product they need in F-Secure's portfolio.

The application takes the user through a series of questions that help identify their security needs. To do that, the Product Selector application makes use of HTML5, CSS3, JavaScript, jQuery, XML, Velocity Macro, Liferay and has been developed with the help of frameworks such as Grunt.js, Node.js, NPM, Atom and many more plug-ins and packages to speed up the workflow.

As a result the Product Selector application development process was smooth and the product was published live to help F-Secure customers find what they need. The feedback on the application has been largely positive and work on potential future extensions continues.

Keywords	Web development, web application, SPA, JavaScript, jQuery, HTML5, CSS3, Velocity Macro, XML, Liferay, Git

Contents

Abbreviations and Terms

API	Application Programming Interface
CSS	Cascading Style Sheets
CTA	Call to action
DOM	Document Object Model
GUI	Graphical User Inferface
HTML	HyperText Markup Language
SPA	Single Page Application
JS	JavaScript Language
JSON	JavaScript Object Notation
NPM	Node package manager
POC	Proof of concept
PS	Product Selector
URL	Uniform Resource Locator
UI	User Interface
WYSIWYG	What you see is what you get

1 Introduction

F-Secure is a Finnish security company with over 27 years of experience in information security. The F-Secure consumer business provides security and privacy solutions to end users. The portfolio includes products such as F-Secure SAFE – a multiplatform, multidevice internet security, shopping and banking protection with digital parenting features. F-Secure Freedome VPN is a multiplatform privacy solution that allows hiding or changing computer and mobile devices' IP address and also blocks trackers on the internet. F-Secure Booster is a performance optimizer for PC and Android. F-Secure Key is a password manager that stores safely all passwords and helps generate very strong random ones on demand.

There are a number of free tools that are also offered on the F-Secure consumer web. Those include F-Secure Search, a tool that provides prescreened search results over encrypted connection; F-Secure Online Scanner is a free online tool that scans and cleans the user's PC; F-Secure AdBlocker is an iOS tool that provides advertisement blocking capabilities to iOS's Safari browser.

The product selector is being developed to help visitors on the F-Secure Consumer Web select the right F-Secure product for them, based on their privacy, security or performance needs.

2 Designing a product selector tool

2.1 Goals

The main goal is to help the visitors navigate amongst F-Secure products and at the same time promote and market those products. The tool should provide relevant information and highlight the most accurate product based on the customer's use case or need, a tool to easily answer the question – "What security do I need, based on my internet and device usage".

The Product Selector tool is also intended to raise awareness of the possible needs that customers might have but have not realized before. Of course, last but not least, it should

serve consumers' needs and help promote F-Secure privacy and security solutions and products.

2.2 Examples

At the time of finalizing the product selector tool and this report, we finally managed to find a somewhat closer selector tool (based on layout and functionality), to what we are building [1]. However when the project was kicked off, the following couple of examples were examined.

The Samsung phone selector was among the first interesting examples for a similar selector tool, but still quite far off of what we were going for [2]. It featured a very complicated UI with inconsistent transitions and not a very user-friendly application flow.

Another interesting example that was inspected was the deck selector [3]. This one was easier to understand with smoother transitions but rather long to complete, sitting at ten total steps.

The product selector was designed to be simple, easy and efficient to provide quick and hassle free help.

2.3 Meetings and brainstorming

To facilitate easier development a number of meetings with involved team members were scheduled. The syncs and meetings provided direction for implementation ideas and useful feedback on the visuals, techniques and layout.

2.3.1 Initial Brainstorming

The first brainstorming session focused on what the needs for such a tool really were and how to go from there. Figure 1 shows the POC suggestions for the different application views. Divided in multiple steps, the application should guide the user through a selection of easy to understand at a glance questions. The choices in these steps provide data for us in order to do the calculation and suggest products to the user.

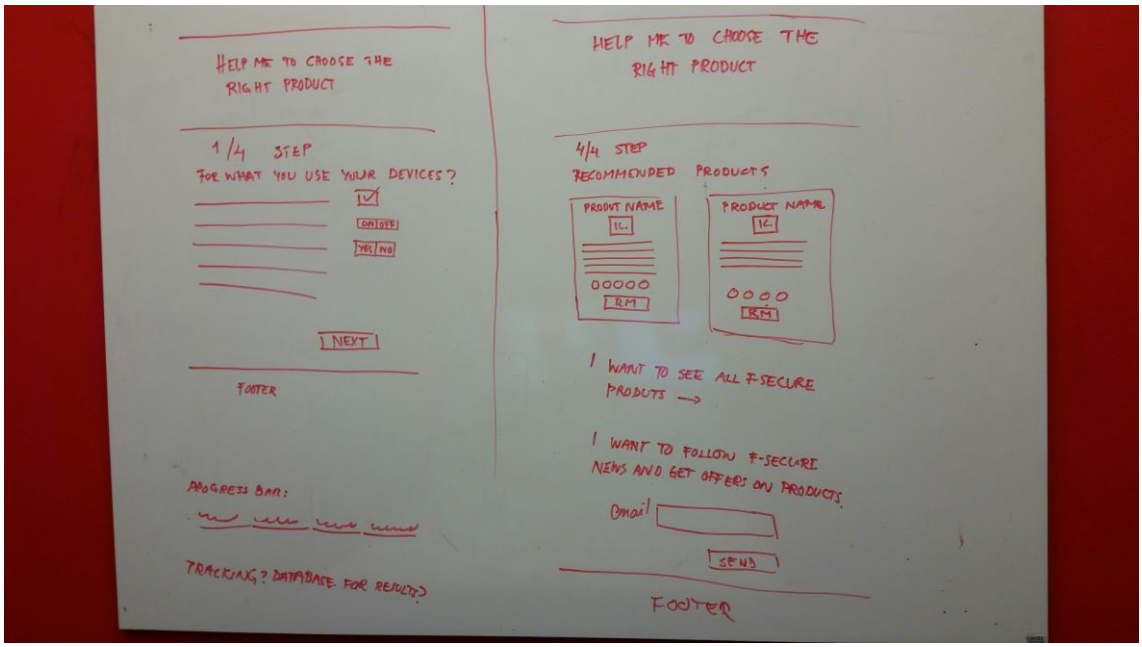

Figure 1. Initial planning of the views.

The initial idea for the application was for a 3-4 step process, asking the user questions like the ones below, in order to acquire information about services and device usage, so that calculations could be done.

- What do you use your devices for?
- What do you want the product to do?
-How many devices do you have?

Over time as the project progressed, the questions were changed, refined and the requirements shifted as well. “How many devices do you have?” was removed as an unnecessary question, since it did not provide any additional value in the calculations and would have created unnecessary complexity for the tool. The answers to the questions were designed in such a way that they held a specific product value that could then be used when implementing the application logic.

2.3.2 Follow-up syncs and feedback

In follow-up meetings the main focus was on the development process, potential problems, issues and how to handle them. Minor project requirements changes have occurred, many bugs were reported and fixed as well as good suggestions implemented.

3 Tools, scaffolding and initial set-up

3.1 Tools and workflow

The used tools were Atom as an IDE, GitHub for version control. Atom is an up and coming completely hackable text editor for the 21st century [4]. To previous users and fans of Sublime Text 2, Atom would immediately appear as a similar type of editor with plenty of customization options, plug-ins and themes and is one of the better choices for an IDE. Built in GitHub integration shows at a glance modified, uncommitted files and file changes. Some of the most essential plug-ins when doing front-end development with Atom are:

Emmet - allows to expand abbreviations with the Tab key, essential for web developers.
JSHint – JavaScript real-time validation.
Minimap – Source code preview add on.
Pigments – Displays color from words or hex values in the editor.
Beautify – Source code formatting tool.
Local server express – Serve current project via Express.

Google coding conventions were utilized when building this project [5].

GitHub was used for source control of the POC phase of the tool, before its implementation in Liferay. Source code commits had descriptive messages, containing incremental work and were regularly pushed to the master branch of the repository. In the background, a Heroku server was set up to then build the source code and deploy from the GitHub repository and publish the application to a sandbox page for testing purposes.

3.2 F-Secure components

As a front-end styling framework, we used the F-Secure Styleguide v4.0.0 which is built on top of the Twitter Bootstrap framework and extends on top of it with the F-Secure specific brand style and helper classes. The style guide is an extensive CSS library helping to speed up development, making it easy to adjust colors, layout, spacing, fonts and covers most if not all styling of elements. There was a learning curve to get familiar with

the code base but that investment paid off greatly further in the development process. Previous Bootstrap base class knowledge helped a great deal with the rapid learning.

HTML-wise most components have been developed from scratch since they did not exist in the style guide premade components and some have been heavily altered, while still holding on to the brand guidelines.

The JavaScript part of the F-Secure style guide was not needed as all components were custom and freshly built from the ground up.

3.3 Additional frameworks

To speed up development and not reinvent the wheel some additional open-source frameworks or parts of them were used.

3.3.1 Node.js and Gulp

For the local development, Node.js was installed and set up, so that Express and Gulp could run servers on top of it [6]. With Node.js installed we can easily install packages such as Gulp using the node package manager (NPM). The –g switch installs the package globally by adding it also to the system path for easier access regardless of the current directory. Below is a snippet of the code we used to install Gulp from the command line, bash or Windows PowerShell.

```
npm install –g gulp
```

This allows us to use the builder and task runner tool. Next we set up a very simple gulpfile.js in the project directory which looks like the snippet in listing 1.

Listing 1. Gulpfile.js contents

In only 15 lines of code, after requiring the necessary packages for the tool to run, we receive the live reload functionality to help with quick repetitive tasks and minimize manual page reloads while working. Using Live Reload for Gulp has vastly helped to automatically watch for file changes in the working directory for .html, .js and .css files. On subsequent saves of files, if there are changes the browsers used for debugging the application are being reloaded. This is just one of the benefits we enjoy from using Gulp as a task runner. Some other use cases include automatic minification of files and concatenation of all style files to a single .css file and all JavaScript files to a single JavaScript file.

3.3.2 Animate.css

Animate.css is an easy to use, open source pure CSS framework (no JavaScript required) [7]. It is a great library that adds great visual effects with little to no effort. In the Product Selector project we used the **slideInLeft** and **slideOutLeft** classes, for the visual transition back and forth between the different application views. Animate.css works by simply applying a class to an element that would fire off a transition. Of course, those classes were added programmatically to appear only when needed to be used.

3.3.3 Hover.css

Just a single snippet was used from the hover.css framework [8]. This open source project focuses mainly on hover effects and although being very extensive, in the development of the Product Selector we ended up using only a single class. That was the `hvr-glow` class which applies a smooth four-sided drop shadow when the element is hovered with the mouse pointer. The effect can be seen in figure 2.

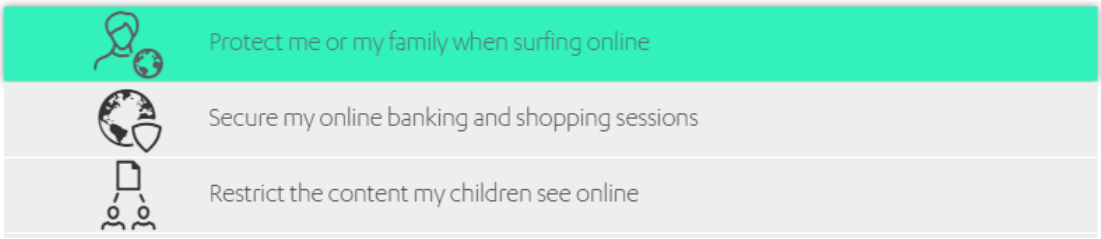

Figure 2. Hvr-glow class of the Hover.css framework

Overall, the hover.css library is easy to include in a project and provides a multitude of useful visuals.

4 Developing the Product Selector

4.1 Designing the layout

In the follow-up discussions after the initial planning of the PS tool, the total number of steps was reduced to two only and a starting view was introduced. The starting view is used as a preparation step for the user, instead of having them thrown directly in the middle of the tool. This smooth transition prepares the user for a test ahead and builds a specific user expectation. Figure 3 shows the starting view of the application, a neat view with a single CTA button.

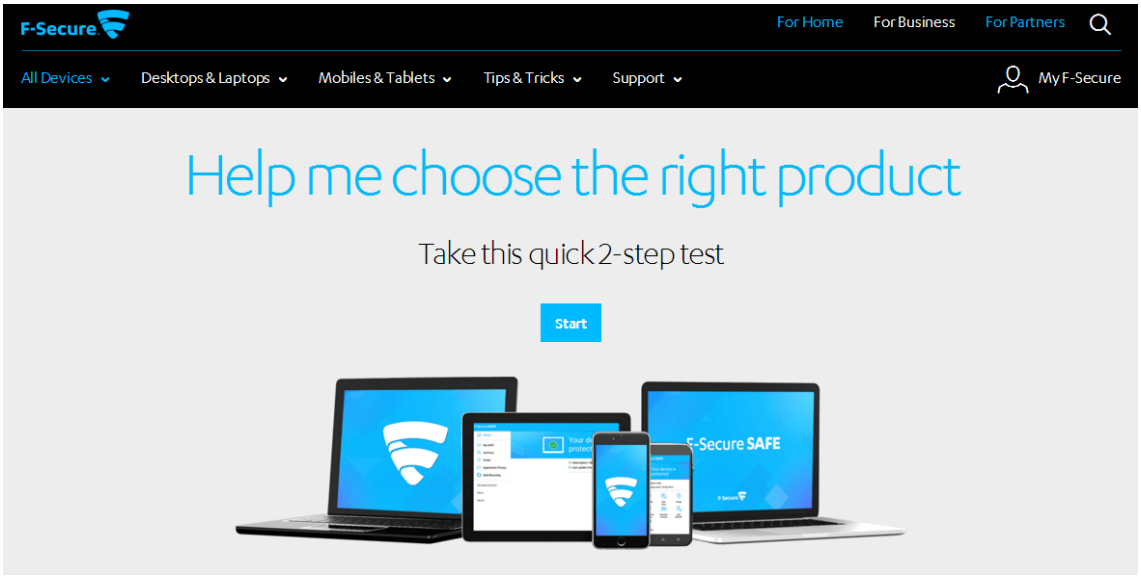

Figure 3. Starting view of the Product Selector application.

Clicking the “Start” button takes the user to the next screen. This is the first step where user interaction is needed to provide data to the application. Figure 4 below shows the second view of the application and the first view where we acquire data from the user. To showcase the hover effect we are using to provide feedback to the user that the elements can be interacted with, the first item has been hovered in the image below.

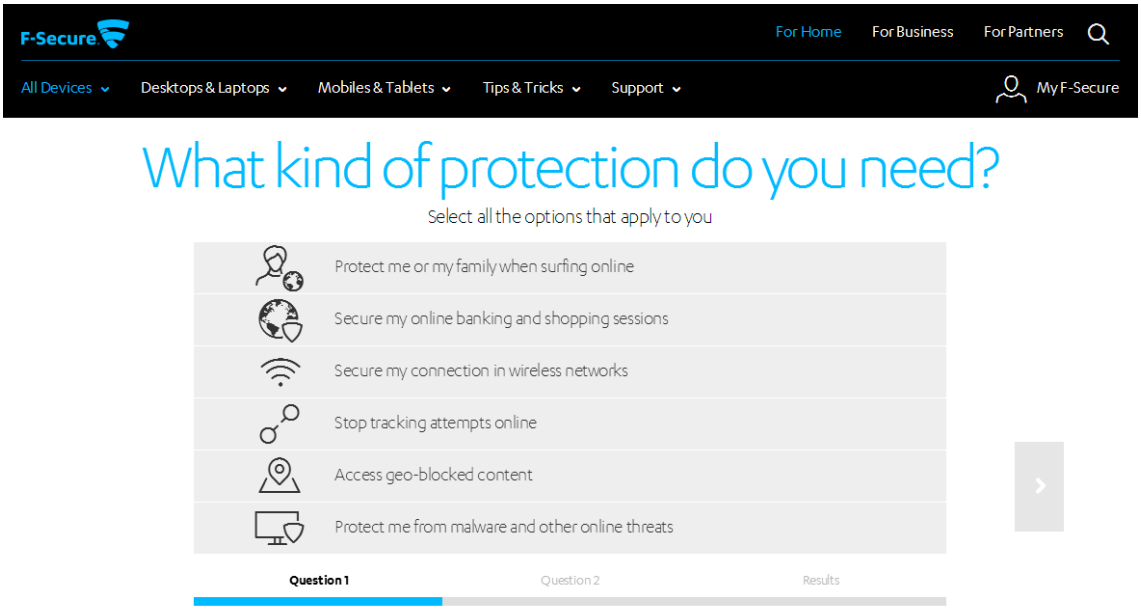

Figure 4. Step 1 view of the PS application.

The main UI elements are the list of questions, progress bar underneath and the arrow buttons. This easy to use UI provides enough information to the user and at the same time is very intuitive. Icons next to the question are giving instant visual feedback on what the selection is about. Smooth hover effects and highlights provide action feedback to the user, making sure the application feels interactive and highlights points of interest. Figure 5 shows how items look when they have been selected/clicked by the user.

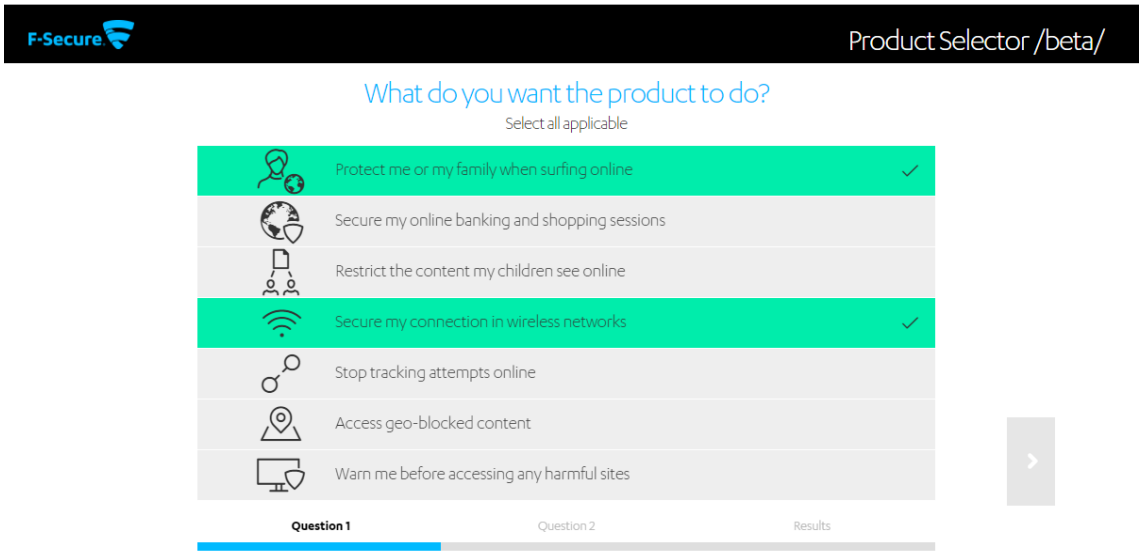

Figure 5. The question view with two selected items to showcase how active elements provide feedback about their status.

The item selected remains with the highlighted color persisting even when the user moves the mouse pointer out of the focus area and a check-mark is added at the right-hand side of the list item. Clicking the next arrow button advances the view to the next step with a smooth side-scrolling transition and updates the progress area by highlighting the current step and advances the progress bar forward.

The second step has the same structure and look like the first step. It again presents questions to the user in the same manner, but answering a different question and with its own different set of answers. Advancing past the second question view brings us to the result view, as shown in figure 6 and figure 7 below.

Figure 6 shows how the suggested product section was designed. This section holds the product selected based on the user's choices. A "left-split" or "right-split" component, with product name, short to-the-point description of product features, "read more" and "buy

now” buttons to allow the user to find out more and / or acquire the product. On the right-hand side the image showcases the product installed on different devices and a blue “play/video” button hints the user additional video material is available right here on this page. Video materials are a great way to additionally advertise the product and highlight core features and advantages. A simple well-constructed video can provide more information and could be more convincing than a wall of text.

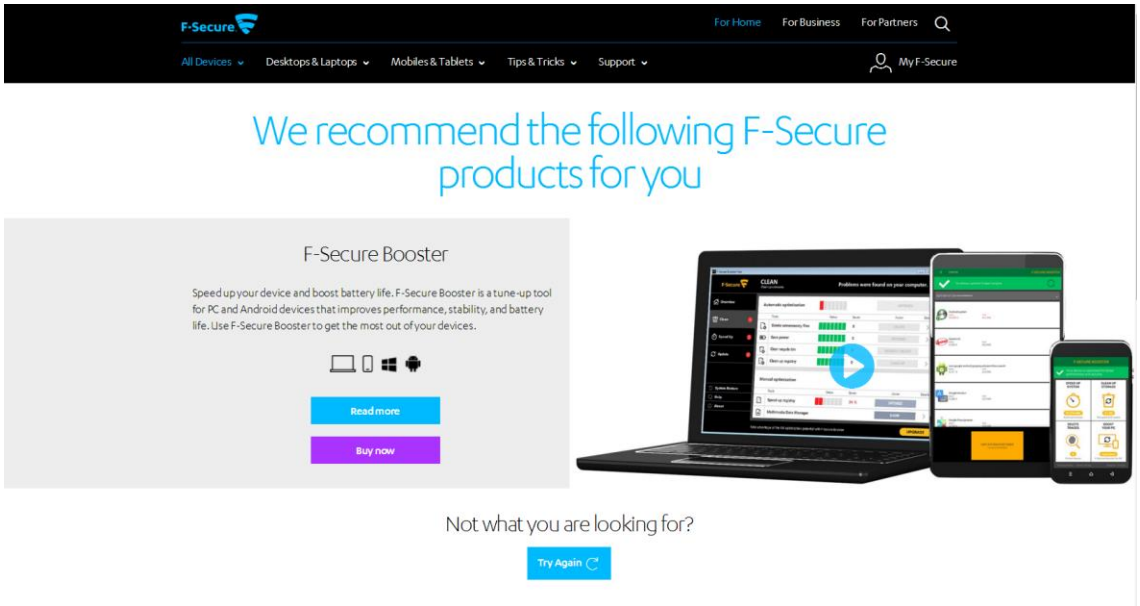

Figure 6. The top part of the result view, presenting the suggested product to the user.

In case the user thinks the suggested product is not what they came here for, they can click the button below and redo the test again. Right after the “retry” button comes the section labeled “more-products”, which contains other relevant products. This section provides information about more F-Secure products, which may be somewhat relevant, but were not selected as a suggested product in the results section at the top of the page. Clicking items in this section provides a link to the product page on the F-Secure consumer website. Last but not least a minimalistic “news and offers” section is laid out to allow users to subscribe to a newsletter and potential future deals.

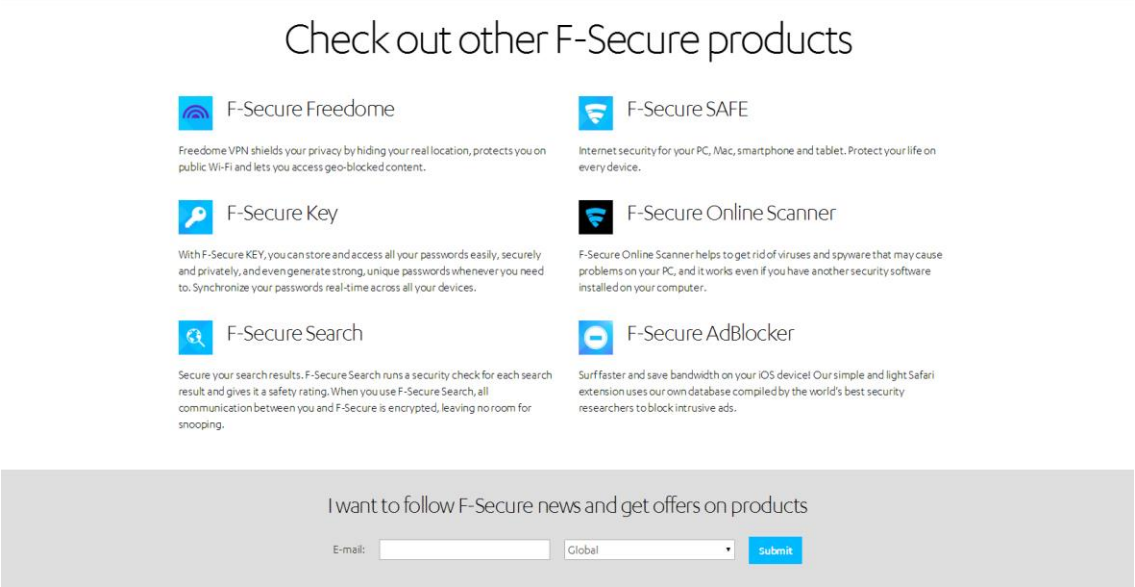

Figure 7. The “more products” section, right after the suggested products section on the result view of the application.

Additionally the results view is a subject to change and improvements as products are updated and gain extra functionality as well as product bundles introduced.

4.2 Building the template

We are using static HTML to layout the different views and UI elements. Every view is composed within a section with an ID and all elements nested inside. As a showcase we are using the HTML code for the starting view of the application and that can be seen in listing 2 below. It is important to notice the section id attribute which we will use to access that DOM element and manipulate it. The helper classes that come from the FS styleguide provide an easy way to quickly style elements – change colors, background colors, add padding and margins, adjust positions and much more. Most of the functionality is done on the front end through JavaScript and jQuery, so action elements are provided unique IDs or specific classes.

```
<!--HERO-->
<section id="step-0" class="fs-section text-center padding-top-medium padding-bottom-mediu
  <div class="container">
    <div class="row">
      <div class="col-xs-12 text-center">
        <h2 class="$view1-heading1-color.getData()">$view1-heading1.getData()</h2>
        <h3 class="margin $view1-heading2-color.getData()">$view1-heading2.getData()</h3>
        <button id="startTest" class="btn btn-secondary">$start-button.getData()</button>
      </div>
    </div>
  </div>
  <div class="container-fluid margin-top-medium">
    <div class="row">
      <div class="col-xs-12 padding-top-x-large padding-bottom-x-large"
           style="background: url('$view1-image.getData()') no-repeat; background-size: co
      </div>
    </div>
  </div>
</section>
```

Listing 2. The HTML code that makes up the starting view of the application.

The question view is created from an unordered list with as many entries as there are questions. The static HTML for that component can be previewed in listing 3. Every list item is centered on the screen, taking 8 columns out of 12 in total in the Bootstrap grid system. Two columns are left off on each side to provide padding around the centered content.

```
<div class="row text-left">
  <ul style="list-style-type: none" class="col-md-8 col-md-offset-2 questions-l
    <li class="lead row question hvr-glow" data-product="s">
      <div class="col-xs-2 text-right"><i class="fsg-icon icon-reseller icon-52
      <div class="col-xs-8 text-left">Protect me or my family when surfing onli
      <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right">
    </li>
    <li class="lead row question hvr-glow" data-product="s">
      <div class="col-xs-2 text-right"><i class="fsg-icon icon-rtpn icon-52"></
      <span class="col-xs-8 text-left">Secure my online banking and shopping se
      <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right">
    </li>
    <li class="lead row question hvr-glow" data-product="s">
      <div class="col-xs-2 text-right"><i class="fsg-icon icon-collaboration ic
      <span class="col-xs-8 text-left">Restrict the content my children see onl
      <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right">
    </li>
    <li class="lead row question hvr-glow" data-product="f">
      <div class="col-xs-2 text-right"><i class="fsg-icon icon-freedome-wifi-se
      <span class="col-xs-8 text-left">Secure my connection in wireless network
      <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right">
    </li>
    <li class="lead row question hvr-glow" data-product="f">
      <div class="col-xs-2 text-right"><i class="fsg-icon icon-freedome-untrace
      <span class="col-xs-8 text-left">Stop tracking attempts online</span>
      <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right">
    </li>
```

Listing 3. The question view HTML code.

In turn, each question is given a helper class for styling and targeting reasons and the very important data-product attribute is assigned a specific value. We will take a look into that a bit later in the logic section below. Every question is given also two icons. The first one, visible at all times, is a functionality icon, there to hint the user, with a simple glance what this question is all about, before they are even able to read it. The second icon, after the question is hidden to begin with, resembles a check mark and appears only after user interaction with the list (in this case a mouse click or finger tap).

Finally, listing 4 shows a code snippet from the results section. Div element with product name for an ID is nested into the section with ID=”results”. Each result is also given the class “card” for styling and targeting in the DOM tree.

```
...
<div id="booster" class="col-md-12 card bg-white container" style="">
  <div class="container">
    <div class="col-xs-12 col-sm-6 half half-booster col-sm-push-6">
      <div class="video-container">
        <div data-youtube-id="wZhE9ogc2OE" class="overlay" data-fullscreen="true" id="wZhE9ogc2OE
          <i class="fsg-icon-thin icon-media-play-overlay font-blue icon-78 play-btn hvr-bounce-i
        </div>
      </div>
    </div>
    <div class="col-xs-12 col-sm-6 padding col-sm-pull-6">
      <div class="row text-center">
        <h3 class="margin-top-none">F-Secure BOOSTER</h3>
      </div>
      <div class="row margin-top">
        <div class="col-xs-12 info text-left">
          Speed up your device and boost battery life. F-Secure Booster is a tune-up tool for PC
           and Android devices that improves performance, stability, and battery life.
            Use F-Secure Booster to get the most out of your devices.
        </div>
      </div>
      <div class="row margin-top">
        <img
        src="https://www.f-secure.com/documents/10192/982446/All+Devices+%28Black%29/9aaaa3cc-616
        426240195000" class="os-icons" alt="os icons">
      </div>
      <div class="row margin-top text-center">
        <button class="btn btn-secondary col-xs-12 col-sm-6 col-sm-offset-3">Read more</button>
        <button class="btn btn-primary col-xs-12 col-sm-6 col-sm-offset-3 margin-top">Buy now</bu
      </div>
    </div>
  </div>
</div>
...
```

Listing 4. F-Secure Booster's results section.

Helper classes again provide an easy way to do small styling adjustments to elements. The section is split 50/50 in the center of its content. One half being available for the image and the overlaying play button for the video, the other half for the product name and descriptive marketing text.

4.3 Implementing the logic

4.3.1 The data-product attribute

The most effective and efficient way of getting a product value was by assigning a product name to each question. Albeit being too simple at a glance this solution also allows us to scale, if for example we decide that a question should provide value for two or more

products. Tables 1 and 2 show the sample questions and their respective data-product attribute values, where table 3 shows the relation between products and value abbreviation.

Table 1. Questions for the first view and their data-product attribute values.

Question 1: What do you want the product to do?	**Data-product attribute**
Protect me or my family when surfing online	s
Secure my online banking and shopping sessions	s
Restrict the content my children see online	s
Secure my connection in a wireless networks	f
Stop tracking attempts online	f
Access to geo blocked content	f

Table 2. Second batch of questions and their data-product attribute values.

Question 2: Have you considered the following security needs?	**Data-product attribute**
Show secure search results	e
Track which apps may threaten my privacy	p
Scan my PC for virus and other harmful malware	o
Maximize my device performance	b
Clean up unnecessary files from my devices	b
Save power on my device	b
Store my passwords and PIN codes securely	k

For each F-Secure product we have assigned a particular value to the data-product attribute. Those values are represented in table 3 below.

Table 3. F-Secure product names and their data-product attribute value by our convention.

Product name	**Data-product attribute**
SAFE	S
Freedome	F
KEY	K
Booster	B
Search	E
Online Scanner	O
AdBlocker	A

Having come up with the data-model, we can take advantage of the HTML5 custom data attributes and create our own, namely “data-product” and assign it the product specific value which in turn is programmatically accessible with JavaScript. Listing 5 shows the result of this implementation.

```
<li class="lead row question hvr-glow" data-product="s">
  <div class="col-xs-2 text-right"><i class="fsg-icon icon-collaboration icon-52"></i
  <span class="col-xs-8 text-left">Restrict the content my children see online</span>
  <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right"></i></div>
</li>
<li class="lead row question hvr-glow" data-product="f">
  <div class="col-xs-2 text-right"><i class="fsg-icon icon-freedome-wifi-security ico
  <span class="col-xs-8 text-left">Secure my connection in wireless networks</span>
  <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right"></i></div>
</li>
```

Listing 5. The “data-product” attribute implementation.

Now that we can get actual values from each question we can proceed with the logic implementation.

4.3.2 Variables, helpers and initialization

Before we implement the core and most interesting part, some more preparation is required. Listing 6 below shows the point object we will use to store the acquired point for every F-Secure product. The points object consists of a key and value pairs. The key being the product name or abbreviation of it and the values are zeroed out at the start of the program. The rest of the snippet shows the usage of jQuery to grab UI elements that will be interacted with or manipulated.

Additionally in this first initializing part of the program we are setting up the initial progress bar animation, so the user sees it responding and starting to fill up as soon as the first question view is reached. We set up the “Start” button transition, using a fadeout and delay of 750 ms, to provide a smooth introduction to the first set of questions. Lastly, the arrow buttons are displayed as soon as the transition is over.

```
16    var $moreProducts = $('#more-products');
17    var $newsAndOffers = $('#news-and-offers');
18    var $startOverBtn = $('#startOverBtn');
19    var $startTestBtn = $('#startTest');
20    var $backBtn = $('#back');
21    var $nextBtn = $('#next');
22    //Init point system
23    var points = {
24      safe: 0,
25      freedome: 0,
26      key: 0,
27      booster: 0,
28      search: 0,
29      ap: 0,
30      ols: 0,
31      sense: 0
32    };
```

Listing 6. The point system object and grabbing UI elements with jQuery for DOM manipulation.

Functions to restart the application, like the following code as seen in listing 7, are implemented and provide restart functionality for the application, wiping the slate clean, resetting points and showing the start screen again.

```
202     //Start over button
203     $startOverBtn.on('click', function() {
204       location.reload();
205     });
```

Listing 7. Restarting the application.

The `maxPts` function is another helper function, which allows us to grab all points from the points object, push them in an array and then use the `Math.max.apply` method to find and return the highest values.

4.3.3 Events and transitions

Our Product Selector application requires most of the click events to be rather straight-forward, and easy to implement. We would only need to attack a click listener with a simple callback that describes what happens. One such case is the question clicking. Listing 8 shows a snippet.

```
76     //Attach click event to all questions
77     $('.question').on('click', function() {
78       $(this).toggleClass("clicked");
79       $(this).find(".checkbox").toggleClass("icon-check-mark");
80     });
81
```

Listing 8. Question click listener and handler.

Because we want to allow the user to deselect an already selected question, should they desire to do so, we need to use the `toggleClass` jQuery method, to make sure our arbitrary “clicked” class goes on and off on every click of the same item. The clicked class applies the green “selected” color to the list item, while we also make sure we add the correct icon class to the icon sibling in order to show the checkmark when the item is selected.

Let’s take a look at the transition function when the arrow is pressed for the first time. So, first things first, we have a click handler on the “Next” button arrow that looks as shown on listing 9.

```
179    //Next button click
180    $nextBtn.on('click', function() {
181      if(StepOne && !StepTwo){
182        StepOne = false;
183        StepTwo = true;
184        stepOneToTwo();
185      } else if(StepTwo && !StepOne){
186        StepTwo = false;
187        stepTwoToResults();
188      }
189    });
190
```

Listing 9. The next button click listener and handler.

Because we mainly want that same button to do two different things, we need to represent some sort of state and in this particular case this is implemented with Boolean variables `StepOne` and `StepTwo`, which check where we are currently in the program flow. Depending on the case a corresponding function is fired and here is how the `stepOneToTwo()` function look like.

```
82    //Transition functions
83    function stepOneToTwo () {
84      //Hide buttons while animating
85      $nextBtn.hide();
86      $backBtn.hide();
87
88      //transition animation
89      $step1.addClass("animated slideOutLeft").one('webkitAnimationEnd mozAnimationEnd MSAnimatior
90        $step1.removeClass("animated slideOutLeft");
91        $step1.hide();
92        $step2.show();
93        $step2.addClass("animated slideInRight").one('webkitAnimationEnd mozAnimationEnd MSAnimati
94          $step2.removeClass("animated slideInRight");
95          $nextBtn.show();
96          $backBtn.show();
97        });
98      });
99
100     //Update question number
101     $(".quest-prog").eq(0).children("i").show();
102     $(".quest-prog").eq(0).children("span").removeClass("text-bold font-black").addClass("font-g
103     $(".quest-prog").eq(1).children("span").addClass("text-bold font-black");
104
105     //Update progress bar with animation
106     $progress.animate({
107         value: 66
108     }, 1200);
109
110   }
111
```

Listing 10. stepOneToTwo transition function.

As indicated on listing 10 above, here is where we implement the animations coming from the Animate.css framework, by only adding the animation name and the string “animated” as classes to the transitioning element. What we do next is to also wait for animation end event and as soon as that fires, we then remove the animation classes and update the UI. The step changes must also be reflected in the progress bar, so the jQuery animate function is passed a settings object as well as a timer, over how much time to play the progress bar filling animation. Those transitions do not hold any data value yet but they are providing UI updates and therefor are rather simple to implement.

4.3.4 Calculations and decision making

Listing 11 shows a sample code from the transitioning to the results section functionality.

```
113    function stepTwoToResults(){
114      //Update UI
115      $nextBtn.hide();
116      $backBtn.hide();
117      //Update question number
118      $(".quest-prog").eq(1).children("i").show();
119      $(".quest-prog").eq(1).children("span").removeClass("text-bold font-black").addClass("font-green
120      $(".quest-prog").eq(2).children("span").addClass("text-bold font-black");
121      setTimeout(
122      function() {
123        $(".quest-prog").eq(2).children("i").show();
124        $(".quest-prog").eq(2).children("span").removeClass("text-bold font-black").addClass("font-gre
125      }, 1000);
126      //Update progress bar
127      $progress.animate({
128          value: 100
129      }, 800);
130      //Hide progress bar in a sec
131      setTimeout(
132      function() {
133        $('#progress').hide();
134        $('#arrows').hide();
135      }, 1500);
136
137      //TRANSITIONS
138      $step2.hide();
139      $loadResults.show();
140
141      //in a few secs get results
142      setTimeout(
143      function() {
144        calculateResults();
145        $loadResults.hide();
146        $results.show();
147        $moreProducts.show();
148        $startOverBtn.parents("section").show();
149      }, 2500);
150    }
```

Listing 11. The final transition from step two to the results views of the Product Selector.

When the arrow is pressed for the second time the `stepTwoToResults` function fires and does mostly the same like the other transition functions but also executes a few additional functions. Listing 12 below, shows a snippet from the result calculation function.

```
285     function calculateResults(){
286       //Loop all user choices
287       $('.question.clicked').each(function(){
288         var attr = $(this).attr("data-product");
289
290         //Error checking if attribute exists so we avoid undefined
291         if (typeof attr !== typeof undefined && attr !== false) {
292           //Increment points accordingly
293           if($(this).attr("data-product").match(/s/g)){
294             points.safe++;
295           }
296           if($(this).attr("data-product").match(/k/g)){
297             points.key++;
298           }
299           if($(this).attr("data-product").match(/f/g)){
300             points.freedome++;
301           }
302           if($(this).attr("data-product").match(/b/g)){
303             points.booster++;
304           }
305           if($(this).attr("data-product").match(/e/g)){
306             points.search++;
307           }
308           if($(this).attr("data-product").match(/p/g)){
309             points.ap++;
310           }
311           if($(this).attr("data-product").match(/o/g)){
312             points.ols++;
313           }
314           if($(this).attr("data-product").match(/n/g)){
315             points.sense++;
316           }
317         }
318       });
319
320       //Suggest products to the user and update the UI
321       showProductSuggestions();
```

Listing 12. Calculating the data-product attribute values.

This function updates the progress bar, and a second and a half after it has completed, it hides that element altogether. The arrows are also being removed at this point in time. A few more UI housekeeping functions are being fired, but the key role here falls on the `calculateResults()` function.

This is where the magic happens. The code goes in a loop through all questions flagged as clicked, meaning the ones the user has actually selected. A check then is issued to verify that the elements exist and they have a “data-product” attribute. If all goes well, the attribute is matched against a regular expression based on the product’s code. Then, points are incremented by one per hit for the corresponding product directly in the points object. Whenever the loop is done, so is the function and the application can move on, firing off the next major function.

4.3.5 Providing the results

Listing 13 shows how the `showProductSuggestions` function is being called right after the product calculation function and takes care of the product selection process. It first calls the `maxPts()` helper to resolve what the product with the most points is.

```
226     //Find the product with the most points
227     function showProductSuggestions(){
228       var max = maxPts();
229
230       //User did not select anything
231       if(max === 0) {
232         //show nothing and case with all 7 products in MP
233         $('#nothing').show();
234       } else {
235         //go through all keys
236         var zig = 0;
237         var mp = 7;
238         for (var item in points) {
239           if (points.hasOwnProperty(item)) {
240             if(points[item] == max) {
241               //show the correct product
242               $('#'+item).show();
243               //hide the same product from more-products section
244               $('#more-'+item).hide();
245               mp--;
246               //pull-right every other product to form a zig-zag pattern
247               if(zig % 2 === 0) {
248                 $('#'+item).find(".half").addClass("col-sm-push-6");
249                 $('#'+item).find(".half").siblings("div").addClass("col-sm-pull-6
250               }
251               zig++;
252             }
253           }
254         }
255       }
256     }
```

Listing 13. showProductSuggestions function to take care of the selection process.

Then a check is performed to tackle the case when a user would not select anything at all but still completes the selector tool. In such a scenario we cannot really offer suggestions but we can help the user restart from the beginning and try again or simply advertise all F-Secure products in a neat two-column layout. If however everything goes well and the user has made some selections and therefore points have been calculated, we will check which product matches the max points value. If a match is found, the current product will be displayed and the same product will be removed from the “more products” marketing section below the results. In the event when we have more than one products, eligible to be shown, an additional check is done and classes set. This takes care of

arranging the content into a “zig zag” pattern, as product sections go down below one another, to alter the half split content to be left to right or right to left.

4.3.6 Optimization and scalability

During writing of the code many functions have been iterated on and improved to be faster and more efficient. The main idea was to build the application with modularity in mind. Each module would act as a standalone function so that potential adjustments would not be too difficult to understand or implement.

For example if we decide to change the pointing system in the future from the somewhat straightforward one point per question selected, we could easily change only the calculation function and the rest of the code would still work. Or if we wanted to add support for decimal point values we could implement that as well with a couple of adjustments in the corresponding functions. Of course the code could still be refactored and improved even further.

5 Liferay Implementation

Implementing content in Liferay undergoes a few closely correlated processes. Every article in Liferay relies on a structure and a template. The structure by default is represented in XML. For the template there is a choice between **FTL** (FreeMarker Template Language), **VM** (Velocity Macro) and **XSL** (Extensible Style Sheet Language). Velocity is a scripting language that allows mixing logic with HTML. This is similar to other scripting languages, such as PHP, though Velocity is much simpler. FreeMarker is a templating language which could be considered a successor to Velocity. It has some advantages over Velocity for which it sacrifices some simplicity, yet it is still easy to use. XSL is used in Liferay templates to transform the underlying XML of a structure into markup suitable for the browser. XSL is still being used, but it is also quite outdated. [9.]

For our development we have chosen Velocity Macro as it provides the tools we need to get the Product Selector up and running in a rather straightforward way.

5.1 Writing the structure

The structure was implemented view by view. The first step was to identify what the necessary fields would be. We then needed to identify what additional article information was required, for example font size, font color, background color. This in turn allowed us to change those settings easy in the future, by only using the Liferay GUI. The work process was mainly test-driven in small increments to avoid potential errors and massive throwbacks.

Listing 14 below shows a code snippet of the beginning of the XML structure. The final version of the structure ended at a bit above 1200 lines of code. Essentially, every element must be represented with a `<dynamic-element>` tag in order to be accessible as a field in the Liferay GUI. The most important attribute that every element must have is the `name` attribute. This allows us to access the element's data programmatically in the template, as we will see later in section 5.2.

Elements can be different types. From simple text fields to WYSIWYG text areas, document library fields, which provide a button to select a file from the files already uploaded to the site documents, and more. The child tag `<meta-data>` allows for the addition of extra functionality to the structure. By specifying through the `<entry>` tag as a child of `<meta-data>` we are able to set fields as required elements (meaning the article will not accept this field if it is empty), specify label text for that element for the Liferay GUI, add tooltips with instructions about the particular field and more.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <dynamic-element index-type="" name="STARTING VIEW" repeatable="false" type="selection_break"/>
  <dynamic-element index-type="" name="view1-image" repeatable="false" type="document_library">
    <meta-data>
      <entry name="displayAsTooltip">
        <![CDATA[true]]>
      </entry>
      <entry name="instructions">
        <![CDATA[]]>
      </entry>
      <entry name="label">
        <![CDATA[Starting View Image]]>
      </entry>
    </meta-data>
  </dynamic-element>
  <dynamic-element index-type="" name="view1-bgcolor" repeatable="false" type="list">
    <meta-data>
      <entry name="label">
        <![CDATA[Starting View: Background Color]]>
      </entry>
    </meta-data>
    <dynamic-element index-type="" name="Black" repeatable="false" type="bg-black"></dynamic-element>
    <dynamic-element index-type="" name="Blue Lightest" repeatable="false" type="bg-blue-lightest"></dyn
    <dynamic-element index-type="" name="Blue Lighter" repeatable="false" type="bg-blue-lighter"></dynam
    <dynamic-element index-type="" name="Blue" repeatable="false" type="bg-blue"></dynamic-element>
    <dynamic-element index-type="" name="Blue Darker" repeatable="false" type="bg-blue-darker"></dynamic
    <dynamic-element index-type="" name="Blue Darkest" repeatable="false" type="bg-blue-darkest"></dynam
    <dynamic-element index-type="" name="Green Lightest" repeatable="false" type="bg-green-lightest"></d
    <dynamic-element index-type="" name="Green Lighter" repeatable="false" type="bg-green-lighter"></dyn
    <dynamic-element index-type="" name="Green" repeatable="false" type="bg-green"></dynamic-element>
    <dynamic-element index-type="" name="Green Darker" repeatable="false" type="bg-green-darker"></dynam
    <dynamic-element index-type="" name="Green Darkest" repeatable="false" type="bg-green-darkest"></dyn
    <dynamic-element index-type="" name="Gray Lightest" repeatable="false" type="bg-gray-lightest"></dyn
    <dynamic-element index-type="" name="Gray Lighter" repeatable="false" type="bg-gray-lighter"></dynam
    <dynamic-element index-type="" name="Gray" repeatable="false" type="bg-gray"></dynamic-element>
    <dynamic-element index-type="" name="Gray Darker" repeatable="false" type="bg-gray-darker"></dynamic
    <dynamic-element index-type="" name="Gray Darkest" repeatable="false" type="bg-gray-darkest"></dynam
    <dynamic-element index-type="" name="Pink Lightest" repeatable="false" type="bg-pink-lightest"></dyn
    <dynamic-element index-type="" name="Pink Lighter" repeatable="false" type="bg-pink-lighter"></dynam
    <dynamic-element index-type="" name="Pink" repeatable="false" type="bg-pink"></dynamic-element>
    <dynamic-element index-type="" name="Pink Darker" repeatable="false" type="bg-pink-darker"></dynamic
    <dynamic-element index-type="" name="Pink Darkest" repeatable="false" type="bg-pink-darkest"></dynam
    <dynamic-element index-type="" name="Purple Lightest" repeatable="false" type="bg-purple-lightest"><
    <dynamic-element index-type="" name="Purple Lighter" repeatable="false" type="bg-purple-lighter"></d
    <dynamic-element index-type="" name="Purple" repeatable="false" type="bg-purple"></dynamic-element>
    <dynamic-element index-type="" name="Purple Darker" repeatable="false" type="bg-purple-darker"></dyn
```

Listing 14. Product Selector XML structure.

Flagging the repeatable attribute as true on the questions dynamic element allows us to have a repeatable field, which is presented once in the XML structure but allows for multiple elements in the content article and can be looped with a “for each” statement in the template. We have also implemented a drop-down menu for the icon type of the repeatable question, so it is much easier to select the correct icon class rather than having to remember icon names by heart or needing to reference the Styleguide. This way

of implementation also avoids typos and generally incorrect icon class names, if they would be entered by hand by another stakeholder. In certain scenarios using the `<entry name=”predefinedValue”>` tag and attribute can help to prefill certain fields in the article. Having a descriptive structure with many dynamic elements to customize, like for example heading font color, section background color and more, makes it very easy to adjust those styles later on, straight from the article without the need of writing code.

5.2 Editing and implementing the template

Having developed the static HTML prototype at first, has paid off generously while working on the Liferay template. Here most of the content was mainly copied and pasted straight from the source HTML and as the structure development progresses, parts of the template are implemented. Using Velocity Macro to represent the logical fields coming from the structure, would allow us to later edit the article content straight from the Liferay GUI and would make work on localized versions of the same article much easier.

Listing 15 below shows how the starting view of the application was implemented in VM template. Velocity Macro templates allows for direct structure data field reference by using $ and the name of the field in the XML structure. For example, $view1-heading1. Every dynamic field has a getData() method that returns the value of the element. Calling that method returns its assigned data, which is then injected in the final rendered HTML.

```
<!--HERO-->
<section id="step-0" class="fs-section text-center padding-top-medium padding-bottom-medium $view1-bgcolor.getData()">
  <div class="container">
    <div class="row">
      <div class="col-xs-12 text-center">
        <h2 class="$view1-heading1-color.getData()">$view1-heading1.getData()</h2>
        <h3 class="margin $view1-heading2-color.getData()">$view1-heading2.getData()</h3>
        <button id="startTest" class="btn btn-secondary">$start-button.getData()</button>
      </div>
    </div>
  </div>
  <div class="container-fluid margin-top-medium">
    <div class="row">
      <div class="col-xs-12 padding-top-x-large padding-bottom-x-large" style="background: url('$view1-image.getData()'
      background-size: contain; background-position: center bottom;">
      </div>
    </div>
  </div>
</section>
```

Listing 15. Starting view VM template

Velocity Macro is simple and straightforward to follow as a templating language. In the first set of questions view template we need to use some additional functionality. As shown in the snippet in listing 18, we utilize the foreach loop to access dynamically all repeatable elements added to the article. The .getSiblings() method returns an array of all those elements. We open the loop with #foreach and close it with and #end statement. Every line of code in between will be repeated a number of times equal to the number of elements in the repeatable element array of siblings. To make it easy to render a question useless or avoid errors, we make sure questions with no actual question value are not rendered. We do this with a simple “if” statement as in the snippet in listing 16 below:

```
#if (q.getData() != ””)
```

Listing 16. Velocity macro if statement example

This reads, if the elements return value is nothing, skip the next actions enclosed between this and the ending #end statement. For repeatable elements that have a set of children fields, we then proceed to grab the data by calling the .get() method and pass the child element field’s name as in the example in listing 17.

```
$q.get(“view2-data-product-value”).getData()
```

Listing 17. Acquiring variable values from the structure into the template.

```
<!--STEP 1-->
<section id="step-1" class="fs-section text-center padding-top" style="display: none">
  <div class="container">
    <div class="row">
      <div class="col-md-8 col-md-offset-2 text-center">
        <h2 class="margin-top-none $view2-heading1-color.getData()">$view2-heading1.getData()</h2>
        <h5 class="$view2-heading2-color.getData()">$view2-heading2.getData()</h5>
      </div>
    </div>
    <div class="row text-left">
      <ul style="list-style-type: none" class="col-md-8 col-md-offset-2 questions-list">

        #set ($v2q = $view2-question.getSiblings())
        #foreach ($q in $v2q)
          #if ($q.getData() != "")
            <li class="lead row question hvr-glow" data-product="$q.get("view2-data-product-value").g
              <div class="col-xs-2 text-right"><i class="fsg-icon $q.get("view2-icon").getData()
              $q.get("view2-icon-size").getData()"></i></div>
              <div class="col-xs-8 text-left">$q.getData()</div>
              <div class="col-xs-2 text-right"><i class="fsg-icon checkbox text-right"></i></div>
            </li>
          #end
        #end

      </ul>
    </div>
  </div>
</section>
```

Listing 18. VM template for the questions view.

In the same manner we keep going until we have replaced all static HTML values from our POC code with their corresponding names from the XML structure. Once done, the template is ready. Applying that template and its structure to the article will yield the final render of the application.

5.3 Adding values to the article

Finally, having a ready structure and template, we can go to the web content section in the Liferay GUI and add a new article. This will hold all the actual application data, customizable, editable and localizable. All fields come directly from the structure and provide corresponding fields as described above: text boxes, drop down menus, WYSIWYGs, document library files. Figure 8 shows a look of the article from the Liferay UI.

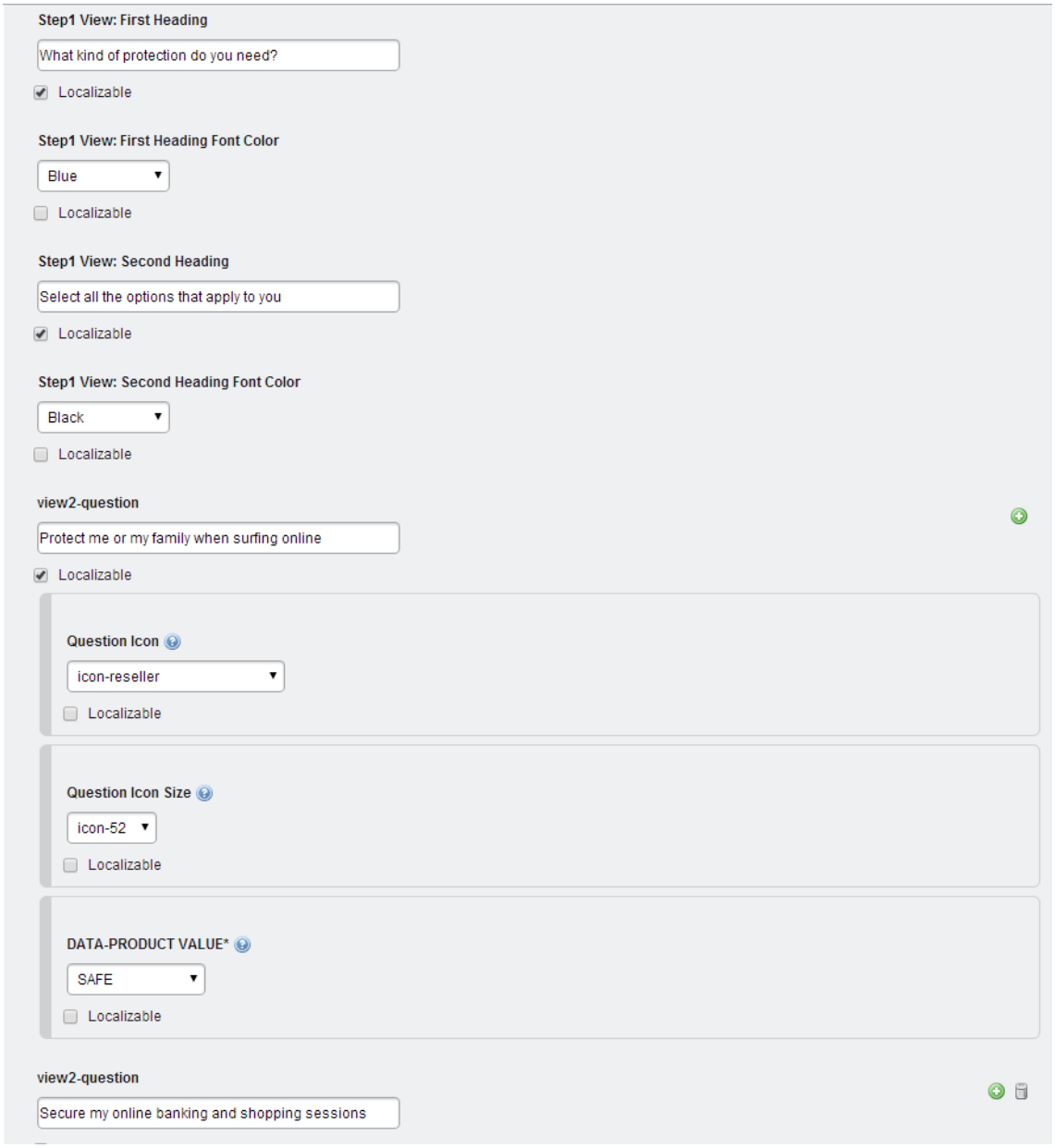

Figure 8. Product Selector article in the Liferay UI.

This part is very straightforward: fill in the blanks. The structure set up also allows us to make style changes easily and test different color combinations for the final look. The one main thing to look out for is to control which fields we want to have the “Localizable” checkbox enabled for. After we fill in the elements with their values and enabling the localizable checkboxes where applicable, we can now go ahead and click “publish” which enables the article to be used on the development and live websites. We can access the article by its unique ID or article name.

5.4 Creating the site page

The creation of the site page is a rather straightforward process. For this particular case we are using the newest F-Secure Styleguide site template, we add all the important settings like which top level navigation and which footer we want and set up the Open-Graph tags and the SEO tags. We make sure that our new page is set up as hidden in the top level navigation, so it would not show up in the menu, as it will be accessible as a link from the home page. Additionally it is worth noting that the page is set up in the staging environment at first and not published immediately live.

5.5 Localizations

When the copy has been finalized and all edits done after revision with team member feedback and stakeholder suggestions, we are ready to send out the content to localizations. Double checking the article for proper flagging of localizable fields is a good practice. After confirming them we go ahead and send the content to localizations. When the localizations department is ready with all the strings for all F-Secure consumer web languages, they can proceed and import them in Liferay. When translations are imported we can once again test the content in different country settings to make sure everything looks as it should, that there are no broken or not localized links.

5.6 Analytics and tracking

Discussion on what would be the potential tracking requirements has commenced in the last month preceding the launch. Of course we would be monitoring page views, unique visitors and other traffic data to see how useful the tool is, does it provide value to end customers and does it generate traffic to product pages. What are the most selected use cases? As discussion is still undergoing on that topic, there is not much to be shared in this report.

5.7 Final issue resolving and publishing

Finally when we have the go ahead to go live with the tool, and we have already checked the article and the localizations, alongside all follow up issues and when all looks good to go, we can proceed to set up the live pages. The page creation as described above

in section 5.4 can now be done and set up for all communities. However as we are going live, the prelaunch_ beginning of the site name will be omitted. Product Selector tool links are added to the home page and the pages pushed live. Those are then being monitored for any issues that could potentially arise in a live environment.

6 Outcomes and conclusion

The implementation of the Product Selector tool has been an interesting and fruitful long-term project. Still the main focus remains for the tool to provide value to the end user and allow for an easy an accurate suggestion of a protection, privacy or performance F-Secure product. Of course in its core this is also a strictly marketing tool, whose purpose is to drive sales and provide additional information about our products.

Constant support from the team members and manager alike has made the work flow smoother and easier without any too great hurdles to overcome. Feedback has been very clear and valuable and weekly syncs and meetings helped clarify the target.

The building of the Product Selector tool has been an interesting challenge and the majority of the goals have been met. There has been a multitude of choices to be made about different parts of the tool's implementation. Certainly there is room for improvement, but nevertheless the current implementation achieves its main mission. From here on statistics and analytics will tell if the tool does as good as we hope it will.

References

1 Android official web site, Phone selector tool at:
https://www.android.com/phones/whichphone/
Accessed on 5th of December 2015.

2 Samsung phone selector tool. Click the compass icon in the top right corner of this URL to access the selector.
http://www.samsung.com/uk/business/
Accessed on 5th December 2015

3 Deck selector tool.
http://timbertech.com/idea-starters/product-selector/
Accessed on 5th December 2015

4 Atom IDE official website
https://atom.io/
Accessed on 5th December 2015

5 Google HTML/CSS and JavaScript Style Guide
https://google.github.io/styleguide/htmlcssguide.xml
https://google.github.io/styleguide/javascriptguide.xml
Accessed on 5th December 2015

6 Node.js official documentation
https://nodejs.org/en/docs/
Accessed on 5th December 2015

7 Animate.css official website and repository
https://daneden.github.io/animate.css/
Accessed on 5th October 2015

8 Hover.css official website and repository
http://ianlunn.github.io/Hover/
Accessed on 5th October 2015

9 Liferay official documentation. Structures and templates.
https://www.liferay.com/documentation/liferay-portal/6.1/user-guide/-/ai/lp-6-1-ugen03-advanced-content-creatio-10
Accessed on 5th October 2

Appendix 1. Git for version control

Version control is a system that records changes to files over time. This allows reverting the project back to a previous state as well as keeping track of updates as incremental state changes. Understanding Git and how it works provides key features to software developers and is crucial nowadays.

Git is a version control system (VCS). In a nutshell, this is a system that allows tracking of changes to the code base by taking “snapshots” of the repository at a specific time. Those different snapshots are saved forever and can be rolled back to easily in case it is needed. On top of that, Git makes it easier to collaborate with others when writing code. Keeping up to date with changes, as well as updating other collaborators when the current user adds new features.

In the Product Selector’s case, the POC project was initialized in a git repository. Git provided useful and easy workflow to commit work and updates, as well as have one common place to push the current live version of the project to, so other stakeholders could preview and provide feedback.

Going over all git commands is beyond the scope of this appendix. The basic workflow with Git includes some of the following commands from the command line:

```
git add readme.txt
```

Then, to commit the file from the staging area write:

```
git commit –m “add readme.txt”
```

With this command we have committed the file with the passed as argument message.

```
git push origin master
```

After adding a remote location for the repository, we can push local changes there as well, making it easier for others to fetch and merge those changes to their respective local repositories.

The basic Git workflow is represented graphically on figure 25 below.

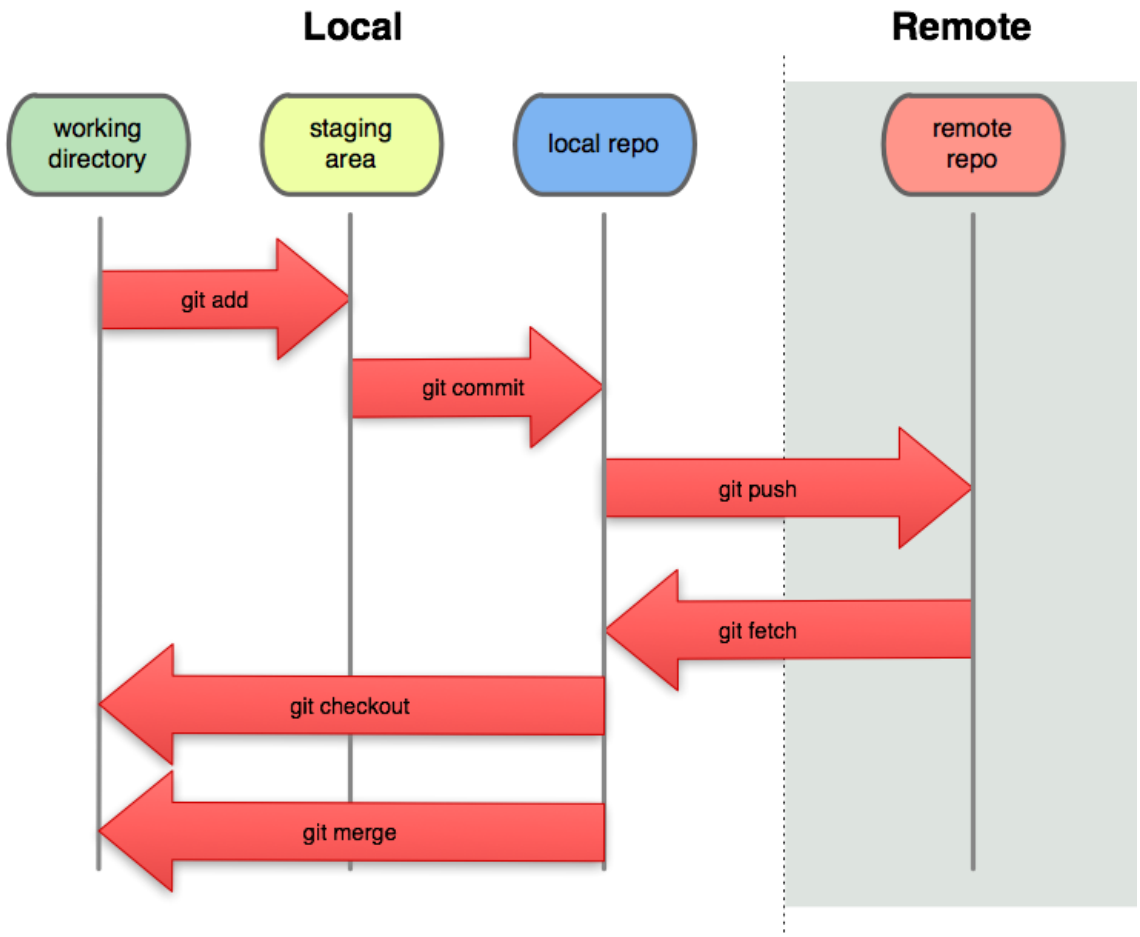

Figure 25. Basic Git workflow

The deployment process is taken care of through a Heroku set up but there are other automatic deployment systems as well. In this case, Heroku takes in and watches for changes in our Github repository and builds and pushes live changes after every commit. This speeds up development and fits the semi-automated workflow. Overall version control tools are essential and Git is definitely a very useful if not even mandatory tool.