

Study and Prototyping the new Bank GUI for Klarna

Christian Rennerskog
Daniel Widgren



UPPSALA
UNIVERSITET

**Teknisk- naturvetenskaplig fakultet
UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
<http://www.teknat.uu.se/student>

Abstract

Study and Prototyping the new Bank GUI for Klarna

Christian Rennerskog och Daniel Widgren

This master thesis is how to go about when creating a Bank application for the community based modern web. Today we do not only need a system that could scale well with into the millions of users interacting with it, but also provide those users with an understandable and extendable interface, and design for every individuals own needs.

Handledare: Per Andersson och Torbjörn Törnkvist
Ämnesgranskare: Iordanis Kavathatzopoulos
Examinator: Anders Jansson
IT 11 032
Tryckt av: Reprocentralen ITC

Contents

1	Populärvetenskaplig sammanfattning	5
2	Acknowledgments	6
I Introduction		7
3	Background	7
4	Problem Formulation	7
5	Purpose	7
6	Limitation	8
II Theory		9
7	Separating the System Logic from the GUI	9
7.1	Model-View-Controller	9
7.2	Event-Driven Frameworks	10
7.3	REST	10
7.3.1	Constraints	11
7.3.2	Principles	11
7.4	Style	12
7.4.1	HyperText Markup Language	12
7.4.2	Cascading Style Sheets	12
7.4.3	CSS frameworks	12
7.5	Interactivity	13
7.5.1	JavaScript	13
7.5.2	JavaScript frameworks/libraries	13
8	Erlang as a Server-side Language	14
8.1	History	14
8.2	Telecommunication and Web applications	14
9	Graphical User Interface Guidelines	15
III Implementation		16
10	Preparation	16

11 Evaluation of Erlang Web Frameworks	16
11.1 Erlang Web	16
11.2 ErlyWeb	17
11.3 BeepBeep	18
11.4 Chicago Boss	18
11.5 Zotonic	19
11.6 Nitrogen	19
11.7 Webmachine	20
11.8 Comparison	21
11.9 Erlang Web Servers	22
12 Choice of Framework	23
12.1 Webmachine	23
12.2 PURE	23
13 Developing the Prototype	24
13.1 Overview	24
13.2 Technical	25
IV Conclusion	26
14 Results	26
15 Discussion	26
16 Future Work	27
17 Problems	28
V Glossary	29
VI Bibliography	31
I Klarna GUI Guidelines	32
A Design Principles	33
A.1 Design principles	33
B Interaction Guidelines	35
B.1 Forms	35
B.2 Application Main Page	37
B.3 Lists	38
B.4 Navigation & Tools	40

B.5	In-page Editing	42
B.6	Drag and Drop	43
B.7	Overlays	44
B.8	Direct Selection	45
B.9	Inlays	46
B.10	Feedback Patterns	47
B.11	Lookup Patterns	48
B.12	Invitations	49
B.13	Transitions	50
B.14	Internationalization	51
B.15	Accessibility	53
B.16	Visual Design	54
C	Building a Pattern Library for Klarna	56
C.1	Documenting Patterns	57
C.2	Best Practices for a Library	57
C.3	Using Patterns	58
II	Instructions	59
D	Running the Proptype	59
D.1	Common Problems	59

1 Populärvetenskaplig sammanfattning

Examensarbetet utfördes hos Klarna AB med syftet att ta fram nya riktlinjer för att skapa en mer konsekvent struktur och utveckling av deras hemsidor. Grundat på studier inom människa-datorinteraktion var även vår uppgift att ta fram en ny prototyp åt företaget som i största möjliga mån använde de etablerade normer som finns inom ämnet. I Klarnas system, så är logiken och koden för att generera presentationslagret integrerad vilket försvårar effektiv utveckling. Därför fick vi även i uppdrag att finna en tydlig metod för att separera koden på bästa möjliga sätt.

Prototypen skulle använda sig av programmeringsspråket Erlang, det var den begränsning vi hade när vi började. Erlang är ett språk som utvecklades av Ericsson AB Computer Science Lab under mitten av 80-talet, gjort för telefonväxlar. Idag är det ett språk som används inom utveckling på flera områden för att kunna dra nytta av samma fördelar som telekomföretag har gjort. En telefonväxel får inte gå ner eller stanna, och dessa egenskaper är även viktiga även i ekonomisystem som Klarnas. Andra önskvärda egenskaper är att det måste vara säkert, och väldigt feltolerant.

I examensarbetet tittade vi närmre på tydliga metoder och verktyg för att separera logik från koden som skapar hemsidor. En teknik vi tittade närmare på var Model-View-Controller, en väletablerad metod för att separera logik från presentation. Vi tittade även närmre på Representational State Transfer (REST) för att kunna styra HTTP-anrop till rätt resurser. I examensarbetet skulle vi även bygga en prototyp med de riktlinjer som vi etablerat. Prototypen skulle vara i formen av Då prototypen skulle vara en bank skulle vi se om vi kunde göra den mer användarvänlig än de internetsidor som banker erbjuder idag. Många banker idag ger intrycket av att vara statiska med mycket överflödigt information för användaren. Därför ville vi skapa en dynamisk användarupplevelse där användaren själv har större kontroll över den information som presenteras.

Användaren ska på ett enkelt sätt kunna anpassa sin tjänst efter egna önskemål och snabbt hitta de tjänster som användaren är intresserad av. Vi tittade därför närmre på gränssnitt som kunde efterlikna de skrivbordsliknande gränssnitt som vi är vana vid i våra fönsterhanterare.

2 Acknowledgments

We want to thank all great people at the Research & Development department at Klarna for their time in helping us doing this master thesis. Special thanks goes out to Torbjörn Törnkvist and Per Andersson which were our supervisors during this period.

Part I

Introduction

3 Background

Klarna AB is a financial institution dealing primarily with high volumes (several millions per year) of small (1.000 SEK) transactions. An extensive Graphical User Interface is used today by Klarnas customer care as well by Klarnas customers (10.000 e-commerce sites).

The Graphical User Interface (web interface) used today by Klarna is extensive but rather crude. The displayed 'view' consists of HTML which is created by the server to be rendered by the browser. The code handling the production of the view this is mixed with the code dealing with the logic. This thesis involves investigating and establishing guidelines for building Graphical User Interfaces. It also includes exploration of methods for separating the GUI from the system logic (e.g. with the Model-View-Controller pattern) and prototyping of established guidelines with a framework built in Erlang. In addition, the thesis include some exploration of the new functionality and possibilities that the GUI could provide.

4 Problem Formulation

Questions and problems that we needed to look at were:

- What guidelines define a good User Interface and how are these incorporated?
- How can we separate logic and GUI?
- What basic functionality the GUI provide to its customers?
- Thinking outside of the box; what new functionality could the GUI provide? (E.g. Klarna Labs: A plugin-based system for enabling experimental functionality.)

5 Purpose

One of the purposes for this thesis is the investigation how suitable different Erlang web frameworks are for building modern graphical user interfaces. Also how suitable they are when separating the production of the view and the code dealing with logic.

Another purpose was to investigate and form guidelines for Klarna to use when building their graphical user interfaces. The reason for this is to create standards within Klarna so that they will provide their customers with a nice and consistent user experience.

6 Limitation

In this thesis we were limited to use Erlang because of the company environment. If time permits other web frameworks would be investigated as well, but the main focus was on frameworks made with Erlang. We were not to create any real implementation, but instead only create prototypes.

Part II

Theory

7 Separating the System Logic from the GUI

In the early days, the web was mostly about sites, not applications. Until a few years back, this was still the case. HTTP and HTML are page-oriented, and scripting languages made it easy to develop dynamic pages. Because of this web developers missed the shift to event-driven programming which recently has become more popular.

For small web applications, the page-oriented paradigm may work fine, but when the application grows, it quickly breaks down. The core of the problem is one of encapsulation. The typical script page consists of two kinds of code, action code and presentational code. These two does not belong together. From an encapsulation point of view, there is no reason why the unit of code that performs an action should know about other navigation changes associated with other buttons on a page. The more complex the navigation becomes, the more apparent it gets. Typically, code blocks in scripting pages look like this:

```
if (pageID = "pageX") then
    //do something
else
    //do something else
end if
```

This example is a form of event-driven programming where each button represents an event. The problem is that the navigation rules and the processing code are all mixed up together. Code like the one above is dependent on the preceding page flow, yet the nature of the dependency is not clearly visible. Hidden dependencies like this makes the code vulnerable.

7.1 Model-View-Controller

Model-View-Controller (MVC) is an architectural pattern that isolates the application logic from the input and presentation of the GUI. Consequentially it permits independent development, testing and maintenance of each. MVC also helps reduce the complexity in architectural design and increase flexibility and reuse of code.

The **model** is the domain-specific representation of the data upon which the application operates. Domain logic adds meaning to raw data. When the model changes state it notifies its associated views so they can act upon it. MVC does not mention the data access layer (often a persistent storage mechanism), since its understood to be beneath or encapsulated by the model.

The **view** renders the model into the interface (where the interaction takes place). Multiple views can exist for a single model for different purposes.

The **controller** receives input and initiates a response by making calls on model objects.

Model-View-Controller has long been considered the architecture to use in Web applications. In Web applications the view consists of HTML or XHTML generated by the application. The controller receives GET or POST input and decides which model it should operate on.

7.2 Event-Driven Frameworks

Event-driven frameworks let you code the Web application more like the GUI application it really is. The programming paradigm lets the flow of the application be determined by events, i.e. user actions (e.g. mouse clicks, key presses) or messages from other programs or threads. These frameworks decouple event handlers (action processing) from view selection (navigation and style). Commonly this is done by using a navigations rules file that specifies all the page transitions independently of any code that handles events. Instead of having dependencies hiding in many places, they are represented explicitly in one place. The code becomes more robust and maintainable.

The event-driven paradigm can be applied in any programming language, although the task is easier in languages that provide high-level abstractions.

7.3 REST

REST or Representational State Transfer is a term defined and introduced in the year 2000 by Roy Fielding in his doctoral dissertation "*Architectural Styles and the Design of Network-based Software Architectures*". Architectures of REST-style consist of clients and servers. Clients initiate requests to servers which in turn process the requests and return appropriate responses. The client can either be in transitions between applications states or at rest. A client in a rest state creates no load on the server or network, while it is still able to interact with its user. All requests and responses are built around the transfer of representations of resources. A representation of a resource can be any coherent and meaningful concept that may be addressed.¹²

Each resource is referenced with a global identifier (e.g., a URI in HTTP). In order to manipulate these resources, components of the network communicate via a standardized interface (e.g. HTTP) and exchange representations of these resources. Any number of connectors can mediate the request, but without seeing past it's own request. Thus, an application can interact with a resource by knowing only the identifier of the resource and the action required. It does not need to care about any intermediaries like caches, proxies, firewalls etc.

¹A *Brief Introduction to REST*. Viewed in 2011-04-10. <http://www.infoq.com/articles/rest-introduction>

²*RESTful Web services: The basics*. Viewed in 2001-04-10. <https://www.ibm.com/developerworks/webservices/library/ws-restful/>

7.3.1 Constraints

The REST style applies the following constraints to the architecture, while the rest of the implementation is free to design:

Client-server Clients are separated from servers by a uniform interface. Clients are not concerned with data storage and servers are not concerned with interface or user state. This leads to a more scalable system and allows for clients and servers to be developed independently, as long as the interface is not altered.

Stateless No client context is stored on the server. Any state is held in the client. This makes servers more reliable in face of network failure and thus improves scalability.

Cachable Responses must define themselves as cacheable. Well-managed caching could in some cases reduce or eliminate client-server interactions, improving performance.

Layered system A client is unaware if its connected to the end server or an intermediary on the way. Intermediary servers may improve security and system scalability.

Code on demand (optional) Servers can transfer logic to the clients which it can execute and thereby extend the functionality of the client (e.g. with for example JavaScript).

Uniform interface A uniform interface between client, implemented with the four principles below in mind decouples and simplifies the architecture. Conforming to these REST constraints is referred to as being RESTful. These constraints all lead to a separation of client and server logic. While the server doesn't need to care about presentation, the client-side application need only to understand the format of the information (representation) returned. There is no need for the client side to worry about storage, low-level logic, or intermediaries to the resource.

7.3.2 Principles

A concrete implementation of a REST Web service follows four basic design principles:

- Use standard HTTP methods explicitly (e.g. POST, GET, PUT, DELETE).
- Communicate stateless.
- Expose directory structure-like URIs.
- Transfer data XML, JavaScript Object Notation, or other valid Internet media types supported by the web service.

7.4 Style

The view of any web page today consists of markup and style sheets. Even here it is important to separate these to improve flexibility when developing a web service. This chapter will give a brief overview of this technique used in almost every modern web page.

7.4.1 HyperText Markup Language

HyperText Markup Language, is the predominant markup language for web pages. HTML elements form the basic building blocks of all websites. HTML is written in the form of HTML elements consisting of tags, enclosed in angle brackets. In between pairs of these tags, text, tables, images and other content is put. A browser uses the HTML tags to interpret the content of the page and compose them into the visual and audible documents.

To define the appearance and layout of content in HTML, web browsers often refer to Cascading Style Sheets.

7.4.2 Cascading Style Sheets

Cascading Style Sheets (CSS) is a style sheet language used to define the look and formatting (the presentation semantics) of a document written in a markup language. In most cases, CSS is used to style web pages written in HTML or XHTML, but can also be applied on any kind of XML document.

CSS is primarily designed to enable the separation of document content (the markup) from layout, colors, fonts and other document presentation. This separation reduces complexity and repetition of structural content, improve content accessibility and provide more control over presentation specifics. CSS also allows for the same markup to be presented in different styles for different rendering methods (e.g. printing). Moreover CSS provides, when used efficiently, site-wide consistency, flexibility and easier page reformatting. By changing a few rules in a global style-sheet, it takes away maintenance that often were time consuming, expensive and difficult. Markup are often linked together with the style-sheet but can be overridden by one that the readers decide.

The syntax of CSS is simple with a number of keywords to specify the various style rules. CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. Properties and weights are calculated and assigned to rules, so that results are predictable.

7.4.3 CSS frameworks

To allow for easier, more standard compliant styling of web pages, you could use a pre-prepared CSS library. They often provide a number of ready-made options for layout and design of a web-page. This often helps when building fast prototypes and such.

7.5 Interactivity

A common way to extend the functionality of the client and thereby reduce server load and give the user a more pleasant experience is to transfer logic to the clients which it can execute. The one we are to focus on, and by far is the most common way is to use client-side JavaScript.

7.5.1 JavaScript

JavaScript is a object-oriented scripting language that has the properties of being dynamic and weakly typed. It is also considered to be a functional language since it has closures and supports higher-order-functions. JavaScript is implemented as a part of a web browser in order to provide enhanced user interfaces and dynamic websites. This enables programmatic access to computational objects within a host environment.

The syntax of JavaScript is strongly influenced by C and Java. Except from names and naming conventions, Java and JavaScript are totally unrelated, with very different semantics.

JavaScript is primarily embedded in or included from HTML pages and used to interact with the Document Object Model (DOM) of the page. Since JavaScript code can run locally in a user's browser, the browser can respond to user actions quickly, making applications more responsive.

Since JavaScript is the only language that the most popular browsers share support for, it has become a target language for frameworks in other languages. The increasing speed of JavaScript engines has made the otherwise more performance limited language popular when bringing dynamic content to web pages. Many web applications today build their user-interface logic in JavaScript, with JavaScript dispatching requests of information to the server.

7.5.2 JavaScript frameworks/libraries

JavaScript libraries and frameworks are created to make common tasks in JavaScript trivial.

JavaScript libraries put much of their focus on that the library should help developers to separate behavior from page structure. This movement is known as *Unobtrusive JavaScript*. Unobtrusive JavaScript considers any JavaScript expressions or statements embedded in the `<body>` tag of HTML pages, either as attributes of HTML elements or in script blocks, to be incorrect. Rather than embedding behavior in the markup, it's moved outside the document body to the `<head>` section of the page.³

Anything that helps us make a clear separation between logic and markup should be a good thing from an MVC point-of-view. Unobtrusive JavaScript forces developers to apply good coding patterns but might at the same time increase code bulk as it will take more lines of code to accomplish even simpler

³Bear Bibeault and Yehuda Katz. *jQuery in Action*. Manning Publications Co. 2008. pp 2-5

tasks. Here is where the importance of JavaScript libraries and frameworks comes in to place.

8 Erlang as a Server-side Language

Klarna AB is a financial institution dealing primarily with high volumes (several millions per year) of small (1.000 SEK) transactions. An extensive Graphical User Interface is used today by Klarnas customer care as well by Klarnas customers (10.000 E-commerce sites). All of Klarnas code base (100.000 lines of code), a mix of logic and view, is in Erlang. Therefore, it is needed to describe why Klarna are so keen on using Erlang as a server-side language.

8.1 History

To understand why Erlang is a good language for a server we should talk about the history. Erlang was researched and developed by Ericsson's Computer Science Laboratory in the middle of the 1980:s. At Ericsson they tried to create prototype Telecom applications with the different languages that was available. These languages had all small pieces that was good for what they wanted to achieve, but none of them had all features that they needed. They decided to invent Erlang from the conclusions that have come from two years of research.

The first product launched with Erlang was in 1994 and gave good feedback to the team that integrated the new features into the 1995 Erlang release. After this project the language was ready to be used in major projects that included Ericsson's broadband, GPRS and ATM switching solutions. With this the OTP framework was developed and released 1996. Erlang was made to be in telephone switches in systems ⁴

8.2 Telecommunication and Web applications

In telecommunication they talk about the five nines, that means that a system must be up for 99.999% of the time. Downtime is 5.26 minutes on a year. Erlang is built with features that make it possible.

Erlang is easy to read and easy to understand. It is a functional programming language with high abstraction level and pattern matching.

Concurrency is a part of Erlang either as transparent or explicit. the processes is light-weight and highly scalable. Each process works in its own memory, which is good because processes will not disturb each other.

One of the parts in *Soft real-time* properties is garbage collection that is handled in milliseconds in the process.

Robustness comes with Erlang's simple and consistent error recovery and supervision hierarchies. With heart of Erlang a node or process that goes down can be started again.

⁴Francesco Cesarini and Simon Thompson. *Erlang Programming*. O'Reilly 2007. pp 3 - 4

Distribution between nodes is done based on TCP/IP and helps out to create a connected network.

Hot code loading enables a feature that makes a system not have to shut down to be updated. Telephone switches need to be up almost all the time.

Erlang runs on any UNIX, Windows or Vx Works, with that it has a good portability.

With all these features in a language it is very suited to be a server, it is created to have low downtime. It is easy to distribute it over many computers. It handles error recovery and users can change code on a running system. What a server does is to handle a lot of messages that comes in and then send messages back out. Erlang message passing and creation of processors for sending messages is in milliseconds.⁵

For a financial company like Klarna, where every second of uptime is valuable for all parties, there is no wonder Erlang is a given choice.

9 Graphical User Interface Guidelines

One of the questions we needed to address was how to define and incorporate a good User Interface. In *Appendix Part I: Klarna GUI Guidelines* we have created our Guidelines for Klarna that describes how they can build a consistent behavioral experience across Klarna's web interfaces.

⁵Francesco Cesarini and Simon Thompson. *Erlang Programming*. O'Reilly 2007. pp 4 - 10

Part III

Implementation

10 Preparation

With the knowledge gathered in the theory chapter in mind, we did our preparation by evaluating seven Erlang web frameworks. As some of these frameworks did not meet our basic requirements, e.g. missing documentation, not being continuously developed, being restrictive about storage options etc, these fell out of our attention quickly. Due to time restrictions, we choose only one of those that were left (Erlang Web, Nitrogen, Webmachine) to build our prototype with.

As we choose our framework, we got familiar with the native Erlang database Mnesia, and the JavaScript framework jQuery, both of which were necessary for the construction of our prototype.

11 Evaluation of Erlang Web Frameworks

When this evaluation was done, in the spring of 2010, the Erlang web frameworks below were the only ones known to us, after sweeping through the Erlang community.

11.1 Erlang Web

The Erlang Web is an MVC framework built on OTP principles and uses many standard Erlang practices. The framework has a form of templating approach to generate dynamic pages. It allows merging XHTML code with a dedicated, built-in XML tag called **wpart**. This concept makes it possible to develop pieces of functionality that can be reused in different pages, so-called template inheritance. A set of frequently used patterns has been implemented as a common library of wparts. Among them are patterns for building forms, retrieving data, manipulating and iterating over a list of Erlang terms, and branching constructions. Having wparts defined as XML elements allows the Erlang Web to validate the page with an XML parser (the Erlang `xmerl` library) ensuring the page is free of XHTML errors. A parsed file is converted to Erlang binary and cached on disk or kept in memory.

The Erlang Web also comes with a mechanism called **wtype** that is responsible for validating and formatting the data of some type. It allows formatting of date, time and numbers; and validation of data that comes with GET/POST requests. Formatting and validation can also be done for user defined complex types. Such complex types can be defined using Erlang record syntax.

An application in Erlang Web is constructed using controllers and templates. The applications data flow is defined in a controller as an Erlang module with functions. Each call to the controller function can be preceded with and followed

by one or more calls to the data flow functions. Such preprocessing might include, for example, checking for authentication or validating incoming request data. Moreover, Erlang Web provides a dispatcher engine based on regular expressions in a configuration file which maps URLs to controller calls.

For persistent storage, Erlang Web has a database management system layer supporting the Mnesia database as well as the document-oriented database CouchDB.⁶

Since Erlang Web is built to make individual parts of the application responsible for specific areas of functionality, e.i. after the MVC pattern, it has the necessary properties for separating system logic and representation. That together with the above mentioned features, (e.g. the powerful template language, i18n features etc.) it makes Erlang Web to an complete tool for building complex web applications. On the backside it has a long learning curve and currently only supports the native Erlang Inets Web server and Yaws.⁷

11.2 ErlyWeb

ErlyWeb is an open source framework built around the model view controller pattern. As many other MVC-based Web frameworks, the application flow logic is implemented in a controller module and feed the controller output to ErlyWeb's template language (ErITL) to create a separation between HTML and system logic. ErITL is a rather simple templating language that creates Erlang modules which generates iolists (nested lists of strings and/or binaries). It comes with a set of predefined tags used for the composing of the templates. The ErITL compiler transforms template files into Erlang modules whose functions are exported so they can be used in other modules. Since templates are compiled to BEAM, ErITL doesn't add overhead when writing a template's logic in Erlang.⁸

Using the path elements in the URL, ErlyWeb will automatically route a request to the application. A request to `http://example.com/foo/bar/1/2/3` would make a call to the `foo` module with the function `bar` and pass the argument list `["1", "2", "3"]` to it. This will come without configuration. The framework also contains hooks implemented in the controller for altering request and response as needed. Consequentially preprocessing filters can be created.

ErlyWeb comes with an database abstractions layer (ErlyDB) that provides access to the database via a standard API with automatically generated functions. The API uses the metadata from the database along with the code to automatically map models to the underlying database table. By declaring the relationships between models in the code it can provide one-to-many and many-to-many relationships. It currently supports the MySQL and Postgres relational databases as well as Erlang's Mnesia database.

At the current time of writing, ErlyWeb has not been in development since October 2008. The framework contains many of the necessary parts for creating

⁶*Erlang Web wiki*. Viewed in 2010-06-02. <http://wiki.erlang-web.org/>

⁷*Erlang Web About*. Viewed in 2010-06-02. <http://www.erlang-web.org/about.html>

⁸Yariv Sadan. *ErlyWeb documentation*. Viewed in 2010-06-07. <http://erlyweb.org/doc/>

a full-blown Web application but lacks any additional features that often is appreciated, like for example, i18n features or JavaScript generation. In addition, ErlyWeb only supports the Yaws web server at the moment.

11.3 BeepBeep

BeepBeep is a small Web framework for Erlang built around the MochiWeb Web server and the templating engine ErlyDTL. It's early in its development stage and currently doesn't support any advanced features. ErlyDTL provides dynamic pages using the Django template language. Unfortunately, a lot of features supported by the language has not yet been implemented.

BeepBeep will automatically map URI requests to modules and templates if the code structure and a few rules are followed. You could say it follows a "convention-before-configuration" approach.^{9 10}

11.4 Chicago Boss

Chicago Boss is an Erlang MVC framework still under heavy development. It uses the ErlyDTL templating engine that compiles templates down to Erlang BEAM code. Currently it supports the most common features of the Django Template language. Chicago Boss associates each URL with a function of a controller. The URL `/foo/bar` will call the function `foo_controller:bar`. Template files access those variables passed from the controller.

Chicago Boss comes with an database abstraction layer (BossDB) that provides access to a database via a standard API. Currently, it is only supporting the Tokyo Tyrant database, but other driver will be added in the future. The abstraction layer uses something called BossRecords which is specially compiled parameterized modules that follow the active record pattern. BossRecords will have functions generated for saving them into the database and for accessing related BossRecords.

Since Chicago Boss is newly released and still in need of much development, it's API are likely to change in the future. What Chicago Boss gives you is clean pattern-matching controllers, automatic code reloading, and automatic-generated documentation. The main work of Chicago Boss is on the backside. Accept from providing basic i18n facilities, it gives you no help with front end features like for example JavaScript generation. It has support for the MochiWeb and Misultin web servers.¹¹

⁹Dave Bryson and Steve Vinoski. *Build Your Next Web Application with Erlang*. IEEE Computer Society, July/August 2009. Available from http://weblog.miceda.org/wp-content/uploads/2009/07/build_web_app_erlang.pdf

¹⁰Dave Bryson. *BeepBeep a simple web application for Erlang*. 2010-04-16. Viewed in 2010-06-02. <http://github.com/davebryson/beepbeep/>

¹¹*Chicago Boss*. Viewed in 2010-06-14. <http://www.chicagoboss.org/>

11.5 Zotonic

Zotonic is an Erlang content management system that also claims to be a framework, easy to extend and adapt for specific use. It is built around Webmachine (see the section about Webmachine) and the MochiWeb server. As most other Erlang Web frameworks it has a complete separation between model, view and controller. Since Zotonic is built very modular, it becomes easy to extend and change the existing functionality.

In terms of features, Zotonic has made it the furthest. The framework provides plenty of features in terms of help of developing usable interfaces. Aside from being built on jQuery and the CSS framework Atatomic, it also has built-in comet support. As Chicago Boss and BeepBeep it uses a modified version of ErlyDTL for the view. In this modified version, most Django features is implemented, though sometimes with differences.

Unfortunately, Zotonic ties you to the PostgreSQL database. This might be fine when using Zotonic as an CMS but might hinder you when using Zotonic as a framework to build other types of applications that would require you to handle more load.

Zotonic is one of the most mature frameworks for building pages with web publishing in mind. It provides a solid framework built with proven technologies. Without digging thoroughly into the code, it is hard to determine if Zotonic is suitable for building any kind of Web application. For the moment, Zotonic might better be viewed as a publishing platform than a software development framework for the Web.

11.6 Nitrogen

Nitrogen is an Web framework using an event-driven approach built on top of Erlang pattern matching. With the help of the jQuery library, Nitrogen has support for Asynchronous JavaScript and XML (Ajax) and long-polling HTTP requests (Comet). Adding Ajax effects can be done in just one line of code with the built-in tags. Nitrogen allows you to tag elements with any Erlang term, and then act on the tag in server-side code when the user interacts with the element. Catching the events is then done by an Erlang functions.

A Nitrogen application is built with elements and actions. An element is an Erlang record used to build HTML tags like links, tables, lists and so forth. Nitrogen transforms these records to HTML. The framework comes prepackaged with a rich set of elements that cover most common needs. Custom elements can be defined by creating a new record and a module for that tag.

Actions can be used to capture interactions from the user (click, mouseover and so on) and build dynamic web pages. Nitrogen wraps the JavaScript behind an action using the jQuery library and provides many of its element effects, like fade, animate, and more. Actions such as sending information to the server can be done by binding elements to an Erlang function. By using pattern matching the request will be mapped to the function.

Each page in a Nitrogen application corresponds to an Erlang module using a

simple naming convention. For example, a request to `/example/foo/bar` would map to a module named `example_foo_bar`. Within the module you define a `main()` function as the entry point for the request along with different event functions for action processing.

Nitrogen includes a simple template system. Using a template element (`#template{}`), and a special placeholder syntax (`[[[Module:Function(Args)]]]`) to call back to your page or any Erlang module, a consistent style for the application can be defined. The function called must return Nitrogen elements, an Erlang string, or an Erlang binary. The template element caches the parsed template, eliminating unnecessary disk hits.

Among those Web frameworks using Erlang, Nitrogen is probably the framework that could have the largest success, as it gives you very little heavy lifting as a developer. Nitrogen creates a ready-to-use package for developing interactive Web applications in Erlang. It's work though is mainly in the front end and not in the back. Database decision is all up to the developer. The Nitrogen JavaScript generation let you create advanced user interfaces without much knowledge about JavaScript. The template system is still rather simple and could be a problem when building a non-trivial system requiring user (not developer) control over page markup. In April 2010, Nitrogen 2.0 was released, bringing many improvements over the previous version, and fixing many irritating issues bugging developers. Support exists for all the three major Erlang servers, Yaws, Inets, and MochiWeb.

11.7 Webmachine

Webmachine takes a very different approach in comparison to the other frameworks already processed in this report. Webmachine is rather to be considered as a REST toolkit. Instead of dictating the shape of the rest of a developers application, it provides a framework with conventions that directly map HTTP and REST. It is an application layer that adds HTTP semantic awareness on top of the bit-pushing and HTTP syntax-management provided by MochiWeb, and provides a simple way to connect that to the application behavior.¹²

Applications in Webmachine is built around a set of resources and a set of predefined functions over the state of the resource. A request flows automatically through a built-in decision path that can examine the request to determine the next processing step. At each step in the flow, a resource function can override to implement the application's logic. Each of the predefined resource functions, or *hooks*, has a sensible default return value that results in an appropriate HTTP status code, so that you only override the functions that your application need.¹³ In most Webmachine applications, these functions are small and isolated, which gives an easy understanding of the Web behavior and the relationship between them. Since these functions referentially transparent, tests can easily be written for each component in terms of input and output.

¹² *Welcome to Webmachine!*. Viewed in 2010-05-09. <http://bitbucket.org/justin/webmachine/wiki/Home>

¹³ *Build Your Next Web Application with Erlang*

Another feature of Webmachine is an built-in trace facility. A request can be followed via a dynamically generated view access through the browser. The trace provides a decision graph showing the request path via the application. Further details on the specific functions called, and the decision made at that point can be viewed.

When building a Web application where you want full control over HTTP flow and the components included, Webmachine is to consider. What Webmachine will not do, is provide you with a complete framework, for example including JavaScript generation, database layers, templating tools or coding patterns. That is up to the developer to decide.

11.8 Comparison

The table shows a simple comparison between the main features of the frameworks processed in this report. Webmachine has not been included as is would be unfair to compare it to the others as it takes quite a different approach.

	Erlang Web	ErlyWeb	Nitrogen	Zotonic	Chicago Boss	BeepBeep
Architecture						
Event-driven			X	X		
Data Model	X	X		X	X	
View/Controller	X	X	X	X	X	X
Development						
Admin interface				X	X	
Automatic recompile		X			X	
Error logging	X	X	X	X	X	
Data Model						
Code generation		X			X	
Data validation	X	X		X	X	
EDoc generation					X	
Databases						
Mnesia	X	X				
CouchDB	X					
MySQL		X				
PostgreSQL		X		X		
Tokyo Tyrant					X	
Templates						
ErlyDTL				X	X	X
ErTL		X				
WPart	X					
Nitrogen Records			X			
Container						
EWGI		X				
SimpleBridge			X		X	
Webmachine				X		
Servers						
Inets	X		X			
MochiWeb			X	X	X	X
Yaws	X	X	X			
JavaScript						
Built-in Ajax			X	X		
Built-in Comet			X	X		
JS form validation			X	X		
Other features						
i18n facilities	X			X	X	
API documentation	X	X	X	X	X	X

11.9 Erlang Web Servers

This section provides you with a brief overlook over the main Web servers used in Erlang frameworks.

MochiWeb Mochiweb is an Erlang library for building Lightweight HTTP servers. It provides a small API that gives complete control over how you handle HTTP request and response. MochiWeb is built around the OTP framework principles, and is a fast, production-ready HTTP server.

Yaws Yaws or *Yet Another Web Server* is a general purpose Erlang Web server. Yaws can either run in standalone mode as a regular web server daemon

but also in an embedded mode where it runs in another Erlang application. Yaws is multi-threaded where one Erlang process is used to handle each client.

Inets Erlang is shipped with a built-in HTTP server called Inets that includes several pluggable modules that can be used to extend the server.

12 Choice of Framework

Due to time constraints we only had time creating a prototype using one of these frameworks. Our prototype were developed using Webmachine, PURE, and jQuery. To help us on our way we used a jQuery Dashboard plugin which we extended for our purposes.

12.1 Webmachine

Since Webmachine is very different in comparison to the other frameworks, as well as directly map to the REST idea, we felt this would be the most interesting approach to take. By giving us an underlying toolkit instead of an pre-prepared framework we would get more freedom to build our application, while helping us to be MVC compliant. One of the greatest advantages would be that it would not bind us to any JavaScript framework, giving us the opportunity and flexibility to replace it if desired.

12.2 PURE

Pure Unobtrusive Rendering Engine (PURE) is a fast templating tool to generate HTML from JSON data. All templates are built with HTML, CSS, and JavaScript, nothing else. As a result there is no templating language or real templates anymore. Any HTML page can be used as a starting point and be transformed or made dynamic with some JavaScript directives. Any HTML portion of the modified page can be reused as a source for a new rendering. The representation (HTML), and JavaScript logic remains totally separated. This makes it truly unobtrusive. PURE runs on the browser side but could be made to run on the server side using a JavaScript server like Jaxer. Either Pure can become an extension of JavaScript framework/libraries like jQuery, DomAssistant, Prototype, and Mootools, or use the selector engine of Sizzle, Sly, or Dojo. For our prototype we used Pure as an extension to jQuery, an increasingly popular library, which we already felt comfortable using. The tool is Open Source and released under the MIT license.¹⁴

¹⁴*What is PURE & Why?*. Viewed in 2010-04-07. <http://beebole.com/pure/documentation/what-is-pure-and-why/>

13 Developing the Prototype

This section will give an overview of what idea the in what direction we wanted to make our It will also give a brief technical description on how we developed our prototype.

13.1 Overview

Banks in Sweden usually have static and bloated pages for their customer web services. Our idea was to create a page where the customers have a web application in which they themselves choose the services that are important to them. The application itself consists of a dashboard like interface (also called *portals*), where users can extend their page different kinds of widgets. Widgets is elements, such as a window or a text box, that displays information changeable by the user. The widget is a visual building block, combined in a application, that provides a single interaction point, with all available interactions, for a certain kind of data. At any time, new widgets can be inserted into, or removed from, the dashboard. All according to the users own interest in information. The user can have a palette of widgets to choose from, everything related to the core products of the company, in Klarnas case, invoices and accounts, to products related to Klarnas partners. These widgets can be chosen in a settings toolbar, easily accessible from the dashboard interface. The current widget settings and dashboard environment are saved so that the user will have the same environment as when they left it.

The dashboard would have a basic desktop like interface. Widgets are draggable between columns and are different in size. Each widgets have some basic controls, that could be found in any desktop interface, e.g. settings mode, fullscreen mode, minimize and maximize functionality. Since most users are used to desktop interfaces, the learning curve to use the service is at a minimum. As default the widgets are shown as an overview, and could be maximized to fullscreen mode, where more controls and views are shown related to that widget.

The prototype comes with three mock-up widgets. To get a feeling of how a customer interface for monitoring ones invoices (something that isn't possible today) could look like there is an invoice widget. Also, there is a stock-exchange widget, to illustrate some of the dashboard functionality, like dynamic updates and changeable settings. Finally there is a mock-up of how an partner to Klarna could create their own widget that could be integrated into the dashboard. The dashboard could of course be extended with any kind of services, like a Customer-2-Customer transactions widget, company specific payment widgets, e-wallet widgets, and so forth.

This prototype also has a simple log in page, with an registration form mock-up.

13.2 Technical

The first step we took when building our prototype was to set up Webmachine and a simple database backend. When creating an Webmachine application you get an ready OTP application structure, with basic start/stop functionality, supervisor and dependency controller. As Webmachine is built on top of Mochiweb, we didn't need to set up any web server. The whole application is started using an included shell script. The database backend used Mnesia, an Erlang native database included in Erlang OTP, with basic read, write, update and delete functionality. To control our resources we built an resource handler to handle the dynamic data, and another one to handle the static content, like images, CSS and HTML files.

For the frontend we set up jQuery, PURE and a jQuery dashboard plugin that we to some part extended for our purposes. Two HTML templates (login and index page) were created for the purpose of containing the basic DOM structure and link the CSS and JavaScript libraries.

To pass data between the client and server side, we used the JSON format. Data passed from the client side was converted to Erlang terms using an implementation of RFC 4627 (JSON) for Erlang. The data resource handler could then take appropriate action on how to handle the data and what to pass back as answer.

To handle the dashboard and the associated events connected to it we created an Erlang handler called from the data resource handler. This handler in turn controlled which specific widget handlers associated with each user. Each widget handler specified what dashboard functionality used, which JavaScript and basic HTML structure that was associated with it. Widget specific actions was also specified in the widget handler.

Due to time limitations, some shortcuts were taken, and an completely separated model-view was not implemented. For example, some basic DOM structure is specified in the widget handlers which written in Erlang.

In *Appendix Part II: Instructions* instructions can be found on how to start the application.

Part IV

Conclusion

14 Results

What results from this thesis is the Klarna GUI guidelines, evaluation of Erlang frameworks, and theoretical work about how to separate logic and view. In turn, the prototype is a result from that work.

On the question on how a good user interface is defined and incorporated, The Klarna GUI guidelines can be viewed as the result. The guidelines describes design principles, interaction guidelines and instructions on how to build a pattern library for Klarna. Together with the prototype it answers the question on what basic functionality the GUI provides to its customers.

The theoretical work done in this report, and the prototype as an extension of that shows how to separate logic and view in a good manner. The evaluation of the frameworks and the prototype would show how to do this in a Erlang web service context.

Building such a modularized GUI system shows how the GUI could provide and easily be extendible with new functionality. In the context of a banking service this would be a new way of thinking and building in accordance to structure of the social web that exists today.

15 Discussion

With todays community based world, it becomes ever so important to build products that are flexible and extendible to the customers own preferences. Companies like Facebook, Apple and Google all have in common that they depend on the community around them to build the applications that make their products so great.

As we take a look into the modern day banking services that exists on the web, you could without doubt say that there is much room for new angles, both to improve the usability, and offer a wide range of other services. We can see a increase in the use of mobile applications to offer traditional banking services. What would be interesting is if a financial institute would offer their services more like a community site where sharing and trading of information (and money) would be the core essence. And what would be even better is when the community itself build the services for you. But how would such an system be structured? An system like that would put tough demands on clear API structure, secure authentication methods, and an complete disconnection between client side presentation and server side logic. It would also put demands on clear ways of structuring the applications made for it, both in terms of code and presentation methods. In such an environment, where development is spread out to people with no or little connection to each other, good graphical

interface guidelines are needed to make the system as a whole consistent and usable. In addition, to decrease the friction of using such a service, you would need to simplify the identification methods, while still having security worth mentioning.

The advantages of looking at a banking service in this community based way could be many. As already been done today, social medias can connect to each other. Information sharing becomes easier and the borders on what system you are currently using disappears. With common authentication methods you can travel across different web services without the need to log in more than once. You are always connected.

The aspect of bringing more types of services into the banking system could increase the information gathering. If there would for example exist e-commerce application within the banking system, information about purchases could be used to offer them other related goods or services.

With the modern day customers influence and way of looking at the services they use, and how they want to use them, the innovation in the banking field may take the next step. What you need to provide is the tool to let them develop it. The bank doesn't have to be the service, but instead the gatherer of services.

16 Future Work

Disability issues It is easy to make a pretty web page with nifty features, but to make it usable for people with disabilities is a challenge. Building web services for people with disabilities would require further investigations so that proper guidelines can be formed.

i18n For a company like Klarna that operates in several countries or any major multi-language site for that matter, internationalization is an issue. Due to time concerns, this is something we ignored to investigate with our prototype implementation. A possible solution to this problem would be to create a hook in Webmachine, where data and JavaScript templates sent to the client, are parsed for language tags and replaced with appropriate strings. This is yet to be investigated.

Development API But how about the social spectra of the web? When having todays community- based web, it would be a shame not utilizing this to our advantage. A prominent feature of whatever larger system is the ability to extend it with custom based applications. Partners of the company could for example have the ability to create their own applications using a development API. These in turn could be offered to the end customer for use in the larger system. This could have the advantage of offering more services to the end customer without having to do the work for it. How such a system could work would need further work.

17 Problems

Despite the decent documentation (in comparison to the other frameworks), the learning curve in using Webmachine was high. This lead, from start, to a lot of bad design decisions in the prototype. Another reason for this was that web-machine is (in writing 2010-06-01) a somewhat unused software commercially. Therefore example code and common questions in the Erlang community was almost non-existing.

A lot of time, when writing the prototype was spent on trying to debug errors in the JavaScript code. Of course the largest reason for this was our own poor skills in the JavaScript language. We got some help from the web browser plug-in Firebug though.

Due to the lack of time, we didn't evaluate any JavaScript MVC frameworks for the client side. Even though this thesis was meant to look into Erlang web frameworks, we choose a path (with Webmachine) that would have to require building a client side with some sort of non-Erlang implementation. Therefore looking into a framework for client side development had been reasonable.

We did not do any testing of our prototype. Since this was a prototype, testing might have been redundant, but also might have helped us with debugging. E-unit tests or suite tests with the common test framework could have helped us. Web frontend tests with tools like Selenium would probably have taken to much time and are therefore considered as redundant for this prototype.

Part V

Glossary

Design Patterns A problem which occurs over and over again in our environment, and then describes the core of the solution to that problem.

Anti-Pattern Inappropriate solutions that appear to be good solutions to a given problem.

Rich Internet Application Web applications that have most of the characteristics of desktop applications.

Ajax Ajax (shorthand for asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client side to create interactive web applications. Ajax uses a method for exchanging data asynchronously between browser and server. This is done in the background without interfering with the display and behavior of the existing page, thereby avoiding page reloads.

Flash A free of charge multimedia platform, and popular method for adding animation and interactivity to web pages. Flash is commonly used to create animations, advertisements, video integration, rich Internet applications, and various web page components. Flash can manipulate vector and raster graphics, and supports bidirectional streaming of audio and video.

Heuristics Heuristics are *rules of thumb*, educated guesses, intuitive judgments or simply common sense. In human-computer interaction, heuristic evaluation is when the user interface is reviewed by experts and its compliance to usability heuristics (characteristics of a good user interface) is assessed, and violation aspects are recorded.

Encapsulation *The process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.*¹⁵

JSON JSON, or JavaScript Object Notation, is a lightweight data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays. JSON is often used for serialization and transmitting structured data. It's main application is in Ajax web application programming.

¹⁵Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2007. p 51-52.

LAMP LAMP stands for Linux, Apache HTTP Server, MySQL and Perl/PHP/Python, that is components for building application servers.

OTP The Open Telecom Platform (OTP) is an open source distribution of Erlang and a large collection of libraries containing tools like a compiler, an interpreter and a web-server.

Database "A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.". Taken from Elmasri and Navathe Fundamentals of Database Systems.¹⁶

SQL SQL stands for Structured Query Language and is a programming language that is used for relational databases. Some of the largest relational databases on the market is Oracle, MySQL, DB2 and Microsoft SQL server.

¹⁶Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Pearson International Edition. 2007. p 4

Part VI

Bibliography

1. Bill Scott, Theresa Neil. *Designing Web Interfaces*. O'Reilly, 2009.
2. Pawan Vora. *Web Application Design Patterns*. Morgan Kaufmann Publishers, 2009.
3. Jeff Johnson. *GUI Bloopers 2.0: Common User Interface Design Don'ts and Dos*. Morgan Kaufmann Publishers, 2008.
4. *Apple Human Interface Guidelines*. 2009. Viewed in 2009-02-20. Available from <http://developer.apple.com/Mac/library/documentation/UserExperience/Conceptual/AppleHIGuidelines/>
5. Jakob Nielsen. *Ten Usability Heuristics*. 2005. Viewed in 2009-02-20. Available from http://www.useit.com/papers/heuristic/heuristic_list.html

Appendix Part I

Klarna GUI Guidelines

These guidelines are designed to assist in developing web interfaces for Klarna products. Rather than helping in the visual design, what you will find here will help in creating a consistent behavioural (and in some aspects, visual) experience across Klarna's web interfaces. Some of the advantages are:

- Users will learn the application faster as the interface behaves like applications they already are familiar with.
- Users accomplish their tasks faster.
- Users with disabilities will find your product more accessible.
- The product will be easier documented. An intuitive interface doesn't require as much documentation.
- Customer support will be reduced which of course can be a cost motivation.

User interface design guidelines are developed to address those design challenges and usability problems that are met when managing a web application's "look and feel". Unfortunately design guidelines have limited efficiency, by being either too high-level (as design principles) or too specific in its guidance. Also, many corporations use guidelines to focus on branding and visual design aspects which does not really help us in creating useful interactions.

In order to provide something more practical, while still embodying high-level thinking, design patterns will be presented alongside the guidelines. Design patterns are explained as *"A problem which occurs over and over again in our environment, and then describes the core of the solution to that problem."*¹⁷ Patterns focus on the problem within the context of use. They guide interface designers on when, how, and why the solution can be applied. Recently, patterns have become popular for their benefits:

- Proven design solutions and guidance for their use.
- Improved design process.
- Reusability and consistent interfaces.
- A common, shared language.
- Efficient teaching aid and reference tool.
- Usable web applications.

¹⁷Christopher Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press. 1977.

Further this paper contains design principles (the basis of all guidelines), guidelines with complementing patterns, and suggestions on how to document further problems that may arise in future designs.

A Design Principles

Principles are useful for web designers when defining a philosophy when designing rich interactions. They describe the practice of good interaction and visual design and are thought of as the basis of all design strategies. You could say that the principles form the broader structural aspects of the design.

A.1 Design principles

Make it direct In the book *About Face 3: The Essentials of Interaction Design*¹⁸, Alan Cooper summarizes the simple rule: *Allow input wherever you have output*. We should make the interface respond directly to the user's interaction and not treat output and input as separate processes.

Allow users to directly edit content in context (In-Page Editing), control the interface with Drag and Drop, and directly manipulate objects (Direct Selection). All this work toward creating an intelligent interface according to the WYSIWYG¹⁹ principle.

Stay on the page According to Mihaly Csikszentmihalyi, in the book, *Flow: The Psychology Of Optimal Experience*²⁰, the state of optimal experience can be described as "...the times when people enter a state of concentration so focused it creates a a state of effortless control". People can enter into a flow and feel unselfconsciousness and rise to the peak of their ability. This flow can however be broken by a sudden awareness of the surroundings or by some other interruption. Web interaction is seldom this good. Each time an action is chosen, web experience is punctuated by a page refresh. This is an artefact of technology underlying the Web. With the rise of Ajax, Flash and other technologies, this is no longer the case. It is now possible to perform actions and bring back results for these, while remaining on the same page, and leaving the surrounding context untouched.

Instead of breaking the user's normal workflow with repeated page refreshes, we could now create a continuous visual perception that more closely matches the user's flow. Experience can now be created in context, within the current page.

Keep it lightweight The key to producing an effortless interface is respecting the user's level of effort. You need to understand the user's intent and provide

¹⁸Alan Cooper. *About Face 3: The Essentials of Interaction Design*. Wiley. 2003.

¹⁹What You See Is What You Get

²⁰Mihaly Csikszentmihalyi. *Flow: The Psychology Of Optimal Experience*. Harper Collins. 1991.

just the right amount of interface (Contextual Tools) within the current context. By making instructions, objects, actions, and options visible, the user's memory load can be minimized. The users should not have to remember information from one part of the dialogue to another. This is critical to provide a lightweight experience.

Instead of separating areas of functionality from data, actions can be brought closer to the objects being interacted with. The content is the interface. Fitts's law²¹ states that a time required to rapidly move to a target area is a function of the distance to and the size of the target. The law ties the contextual proximity to ease of use. This means that if a tool is close at hand and large enough, user interaction can be improved. Putting action tools in the context makes for a lightweight interaction.

Functionality can easily be isolated from its context, but this would work against Fitts's law by requiring more effort from the user. Instead of working with functionality separately, the functionality could be brought into the content. Tools become a clear call to action as it gets closer to the point of focus and shortens the path the user takes to complete a task. They should be easy to understand, easy to target, and quick to execute. By doing this, the user feels that the interaction is lightweight.

React immediately When a user starts an unknown process, they hope they will be guided step by step with clear, timely, and contextual feedback. Users want the application to react to their every action. Jacob Nielsen described that *"...the system should always keep users informed about what is going on, through appropriate feedback within reasonable time."*²² Newton's Third Law of Motion states this even simpler: "For every action, there is an equal and opposite reaction."

Speak the user's language Rather than using system-oriented terms (no codes), the system should speak the user's language. Use plain language with words, phrases, formatting, units of measure and concepts familiar to the user²². This can also include culturally relevant metaphors to interpret information. By following real world conventions, things appear in a more natural and logical order. This helps the users understand, recognize and diagnose information.

Grab attention A common problem with rich interaction features is their lack of discoverability. We can deal with this problem with the help of transitions and invitations²³.

Transitions are described as special effects that occur over a specified period of time. They provide the grease that smooths out what happens in the interface.

²¹Paul Fitts. *The information capacity of the human motor system in controlling the amplitude of movement*. Journal of Experimental Psychology, volume 47, number 6. June 1954. pp. 381-391.

²²Jakob Nielsen. *Ten Usability Heuristics*. 2005. Viewed in 2009-02-20. Available from http://www.useit.com/papers/heuristic/heuristic_list.html

²³Bill Scott, Theresa Neil. *Designing Web Interfaces*. O'Reilly, 2009. pp. xiv.

Invitations are the prompts and cues that lead users through an interaction. They can be the just-in-time tips or the visual hints on what happens in the next step of the interface.

Besides the normal visual-processing region of the brain (occipital lobe), there is a second region dealing with attention capture. Thanks to this region, sudden movement or light can grab your attention. This makes animation a powerful tool for rich web interfaces and are together with invitations keys to successful interactive interfaces.

Handle errors The best way of handling errors is making a careful design which prevents a problem from occurring in the first place²². Any documentation helping the user in the system should be focused on the user's task, be concrete, and stay slim. Despite elimination of error-prone conditions, appropriate instructions and labels, errors are inevitable. If an error occurs, an error message should be shown, clearly indicating the problem, and with appropriate instructions for solving it.

Give control to the user Create a system that can handle incorrect decisions made by the user²². If the user chooses a system function by mistake or regrets an action made, the user should be able to leave its unwanted state or undo the action. Moreover it means that the ways of navigation should be clear.

B Interaction Guidelines

The following guidelines are presented as a general set of patterns and techniques which will apply to those interface problems we will face when developing applications for the web. The patterns will state when they are to be used and how they are best practised. Common mistakes related to the patterns are also stated, so called anti-patterns.

B.1 Forms

Registration and order pages does not seldom require you to input information. These are common forms. Users have to enter information to the web application which can for instance be personal information for creating accounts.

Forms should be made as short as possible. Each form element should be analyzed for its importance and the downside for not including them. This way we keep redundant information to a minimum. In addition, consider how easy it is for users to provide that information. When having long and tedious forms, users might rush through its input process, or are prone to leave at an early stage.

Group and split information in forms according to their relevance and logical relations. This will make them more manageable. Required information are to be presented first and optional information later. If forms are split, they will be perceived as shorter in compared to a huge form presented on a single page.

Label Alignment Labels and their corresponding form elements needs to be clearly associated to make the form easier to fill out and to minimize input errors. There are three acceptable placements for labels in relation to their form elements:

- Above the element, with their left edge aligned with the element.
- To the left, with their left edge aligned to other labels.
- To the left, with their right edge aligned to other labels.

Applications that use internationalization should have labels above their input field to accommodate text swell. It then becomes important to include enough visual separation between the label and the previous form element. In an eye-tracking study performed by Penzo in 2006²⁴, this style had the shortest form-filling times. The downside with this layout is the extra vertical space required, since the elements becomes stacked on top of each other.

When placing labels left of their form element, right align them so they are closer to each other. The same study by Penzo showed that the number of fixations was reduced by nearly half with right-aligned labels compared to left-aligned ones. This reduced the cognitive load required by users to complete the task. Another recommendation in the study was to use plain-text labels over bold ones since it is easier to read.

Labels can also be embedded into the form elements. This is commonly done with search boxes. Since they are embedded into the field, they need to be removed when the user focus on them. From an accessibility point of view, this might be risky if you want to accommodate screen readers.

Input Hints/Prompts Sometimes web applications are made so that users must enter information in a specific syntax or formatting requirements. To avoid errors it is good to give hints or prompts on how to fill in the fields. Instructions should be hinted by providing examples, showing accepted formats and showing constraints. Keep these prompts/hints short so that the user reads and doesn't ignore them. Another way to give subtle hints is to keep the entry fields at the same length of what kind of data that should be entered.

Required Field Indicators Since users need to provide certain information to complete a task, required information in forms should be clearly indicated. This will help reduce the chance of a user getting missing information messages. Typically this is done by showing asterisks (often with the color red) in close association with input fields or labels. Try to keep the indicators consistently on all forms and avoid indicating optional fields as it may be confusing for the user.

For required input information that may be sensitive to the user, keep the user informed on why it is required.

²⁴Matteo Penzo. *Label Placement in Forms*. July 12, 2006. Viewed in 2010-03-04. <http://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php>

Smart Input Information can be presented in a variety of formats and syntaxes (e.g., dates, phone numbers etc). As long as it can be parsed correctly, allow user to enter information in their desired way, without showing an error message. Also, consider using alternative ways of filling in information. If filling in dates, a calendar pop-up can for example be shown, in which the user can choose a desired date.

Keyboard Navigation Forcing the user to use both keyboard and mouse when filling out a form can be frustrating but may also make the form inaccessible. Therefore, user should be allowed to use the Tab key when moving between form elements, the Enter key when submitting, and keyboard shortcuts if it creates efficiency. This can make the form-filling process a lot faster with HTML. If needed the default tab sequence can be overridden with the `tabindex` attribute. Below is an example:

```
<input type="text" name="fieldname" id="fieldname" tabindex="10" />
```

Action Buttons Use Action Buttons for submitting form data. It should be ensured that these have a meaningful and consistent labels, and a higher visual saliency than secondary actions. Visual prominence of secondary actions can be reduced by showing them as links.

Action Buttons are to be aligned with form input elements as it leads to faster form completion times. Enable the primary action when a user presses the Enter or Return key. This is especially important in forms with only one input field. After the user have clicked the action button it should be disabled to avoid that an action is performed multiple times. This can have many undesirable consequences, for instance that a query to the database are posted several times.

Error Messages Errors are inevitable even with the most careful design. Missing information, formatting errors and invalid information are particularly common when submitting forms. The solution to this problem is showing an error message that clearly indicates the reason for the error(s). Users should be able to refer to the error message while correcting the errors. In addition, they do not want to re-enter correct data. This is solved by showing them the error message on the same page as the form, along with appropriate instructions on how to fix them.

B.2 Application Main Page

An important design decision is what page users should view when they enter an application. Application main pages are often personalized based on user profiles, and presents the most relevant content for the application.

Control Panels When entering an application, users wants to access a variety of application functions. Control Panels give a good scope on what important

functions to access in the application and makes it quick to access them. When the users feel lost or disoriented, it also becomes a natural place to return to.

Control Panels establish an overall information design approach for pages within the application, in terms of layout, placement of navigation, and so forth. They also serve as a place to inform users with new features, notifications, announcements, functions, and items that need attention. Items that need attention are best highlighted since the user might not expect them.

Dashboard **Dashboards** are a single-page for tracking information and monitoring data. Instead of users having to navigate through several pages for determining items' status, they are presented with a complete overview. That way they have less of a chance to overlook important information. **Dashboards** usually serve users with functions to monitor and track statistics, analysis to determine trends and conditions, information reports to facilitate diagnosis and determine necessary actions.

Since **Dashboards** show monitored data, it is often important to show the information in its historical context. With the help of appropriate charting methods, exception indicators, trend icons, and appropriate colours, users can know the nature of the contextual information. A user expects to get the information fast, therefore it is important to keep statistics clean without jitter. Since the whole purpose of **Dashboards** is to present an overview, make sure the users can avoid scrolling on the page. This is especially important for **Dashboards** that present real-time data.

It is often hard to meet the need of every specific user role. Therefore it is important to allow users to customize their **Dashboards** to meet their own demands. Users might also need to access more detail information for better understanding of the summary view. Provide a way to let users *drill* down to more specific information. Analytical information often needs to be further analysed or shared among others. This can easily be accomplished by enabling download functions for such data.

Portal **Portals** are central applications that aggregate content and functionality from several different sources, and presents access points for those applications, with a common look and feel. In addition, it enables users to customize the content and presentation. Instead of functions that have to be discovered independently, it accesses all relevant content with a unified appearance. **Portals** have their own window like areas (referred to as *portlets*) within a web page, and often supports functionality like minimize, maximize and close.

B.3 Lists

Lists are common in web applications for showing a collection of items.

Simple List When representing items with one or more attributes, often associated with actions, it is common to use lists. **Simple Lists** are shown either

as numbered or non-numbered. Numbered lists are often used when representing items' ranking. Non-numbered list may use icons or symbols to improve the visual impact or indicate the type of the item. Consider showing secondary actions and information to reduce visual clutter. List may also include separators or different background color on alternate rows to improve readability.

Tabular List **Tabular Lists** are useful for users who wants to view items with several attributes. They can easily associate each item (listed in rows) with it's attributes (listed in columns). Allow users to sort items on one or more attributes to compare them to each other. With larger lists, it can be useful to have filter functions so the user can narrow down on fewer items.

When showing items with multi-attribute information it's best to use tables that will give a clear structure. Either add a separator line for each row or use row striping²⁵ when representing data. These approaches help create visual grouping of items in tables with many columns. Row striping leads to better task performance in tracing data²⁶.

To make data more readable, attribute values needs to be aligned according to the cell contents. Use the following rules for alignment:

- Use right alignment for numbers.
- Help the user see negative numbers and deviate data by representing them with the color red.
- Use left alignment for columns that contain text (including date and time).
- Use center alignment for columns that contain short words or status information.

Hierarchical List **Hierarchical Lists** are when items in a list relate to each other (e.g., forums or category-subcategory). When having a parent-child relationship, representing them in a tree structure is often beneficial. Keep it simple so the user can see what child connects to what parent. Another way to easy indicate a parent-child relationship is with indentation.

Sometimes users may want to sort attributes to make sure they can go from a hierarchical list to a non-hierarchical list. Using hierarchical lists can also cause some problems, one of them are deleting. When allowing users to delete parents, keep in mind and show the user, that it will also delete all the children.

Timelines **Timelines** are good to represent trends or changes over time. If the data is compressed over time it allows the users to access more details when hovering over the timeline.

²⁵Colours for alternative rows. Also referred to as *zebra striping*.

²⁶Jessica Enders. *Zebra Striping: More Data for the Case*. September 9, 2008. Viewed in 2010-03-02. Available from <http://www.alistapart.com/articles/zebrastripingmoredataforthecase>.

List Actions There are different actions that users may want to use for items in lists (e.g., edit, delete or compare). There are two different categories of actions: dedicated actions and shared actions.

Dedicated actions are used for single items and shared actions are used when the user can mark several items and choose an action for them. Shared actions should be placed either at the top or bottom of the list so the user understands that they apply for more than one item. Dedicated actions are to be placed on the same row as the item so it stays in close proximity. Use icons to emphasize that they are actions and not data.

Items selected should have their corresponding row highlighted since checkboxes seldom give enough visual indication. Create functions so the user can select and deselect all items at once.

List Utility Functions Some actions applies to the whole list (e.g., printing, data export). Because utility functions apply to the whole list and not individual items, it is important they be shown outside the list and separate from shared list actions.

B.4 Navigation & Tools

Web applications are often hierarchical and give users access to content and functionality using levels of navigation. Navigation can be divided into two main parts, primary and secondary navigation.

Primary Navigation Navigation is crucial for web applications, the user can perceive the page as non-serious or to complex, if not done right. Menus should always be kept clear and understandable. Commonly, navigation bars is placed horizontally when having a limited number of navigation alternatives, and vertically otherwise. An effect of placing the navigation vertically is that it gives less space horizontally for other content.

Consistency is always important. If users know where menus are and how they look they will find them easier to use. Try to keep them at the same position and in a common style. Use highlights to indicate where a user is on a page.

Secondary Navigation After a user have selected a **Primary Navigation** you can have **Secondary Navigation** nestling out from the first one. This will help the user to easier navigate around the web application since it gives categorization and a way to facilitate more menu options in a simple way. Have both the **Primary Navigation** and the **Secondary Navigation** highlighted so the user now were he/she is. If placed horizontally it should be just below the **Primary Navigation** so the user can see that they belong together. Have the menu vertical if the total number of menu options cannot be accommodated horizontally.

With new web frameworks, menus can be styled so they more resemble desktop application, with different kinds of animations like, fly-out menus, pop-up menus and drop-down menus.

Utility Navigation **Utility Navigation** are key application functions like log in, log off, preferences or similar functions that the user need to access wherever they are. Often it is functions that affect many pages on a site and would be confusing for users to have on separate pages. Try to keep these functions in the header (or other clearly visible areas) for accessibility and consistency purposes.

Faceted Navigation **Faceted Navigation** is when the ability to navigate is based on attributes. It might be an good idea to let objects reside with many attributes to help users find what they are looking for. Also, keep user informed about their selections.

Supplementary Navigation **Supplementary Navigation** are navigation options that can be offered to users like alphabetical- or alphanumerical indexes, recommendations or related items. This helps the user to always get new information and also get access to it.

Depending on what situation users are in, different **Supplementary Navigation** is useful. If users know what they are looking for, it can be good to have an index. In e-commerce applications, you want to offer related items, to promote similar products and accessories the customer might want to buy.

Breadcrumbs **Breadcrumbs** help the user navigate within applications that are several levels deep. If the user have teleported to a page using search or in some way lost track in an application with deep linking, they don't want to revisit the start page to find a previous page in the hierarchy. Therefore it's good to provide some help called **Breadcrumbs**, which is a trail of links from the main page leading up to the current one. In a simple way it indicate the users location within the application (e.g., **Main Page > Main-Category > Sub-Category > Page**). Place the **Breadcrumb** trail below the header so that the user can easy see it.

Wizards To help users filling out forms, it might sometimes be better to use a **Wizard**. **Wizards** are forms that the user fills in different steps to complete a task. Remember to keep it short, too many steps and large forms will make the user prone to not continue. Wizards are often seen in checkout processes on e-commerce sites, registration processes or configurations.

During a **Wizard** it can be important to give feedback to the user about where they are in the process. Keep a status bar, or a reminder, on what step they are currently on, and how much is left.

See to it that when creating a wizard structure, that information and input forms comes in a correct order. Have the relevant information grouped in a logical way.

Always-Visible Tools Contextual tools that always is visible. It provides a clear call to action and shows the importance of the function. If a function is critical, make sure it is always shown. The primary reason for using these tools is discoverability. Remember to keep visual clutter and visual items to a minimum.

Hover-Reveal Tools Contextual tools can now be shown on demand. One way to do this is by revealing tools when hovering over an object. This type of contextual tools can be used to reduce visual clutter when hiding non-primary actions. An important thing to think about when designing is how discoverable the additional functionality will be. Users need to be helped finding functionality with familiar idioms such as hyperlinks for actions or arrows to expose menus. Make sure these icons and text labels are clear and understandable. It is often best to avoid using overlays to reveal additional tools. You might end up in covering important information from the user. If using tool overlays, make sure you activate them instantly, as tools should be ready for immediate interaction.

Toggle-Reveal Tools A mode when actions are not part of the main flow, but instead are provided when the need arises. Tools are revealed in it's context whenever the mode is entered. When exited the tools are hidden. Activation and deactivation is symmetrical and the user should not be trapped in the mode.

Multi-Level Tools Contextual tools can be revealed progressively if certain tools wan't to be kept hidden on a mouse hover. This makes categorization of tools easier and makes activation of them explicit. Actions should be as close to the activation point as possible, especially the more common ones. **Muttons** (buttons with a menu) can be used when there is a default action that the user usually takes but alternative actions is still frequently used.

Secondary Menus **Secondary Menus**, or right-click menus, are not seen often in web applications but are possible to make. They can often conflict with browser right-click menus, both as visually confusing and also removing the browser menu when needed. Therefore secondary menus needs to be styled differently and be put in places not similar to traditional web interfaces. They should only contain redundant, or shortcut commands.

B.5 In-page Editing

In-Page Editing means that we don't leave the page for editing, it is done directly in the page within the context of the page. This is often necessary when the user needs to see the rest of the information while editing.

Single-Field Inline Edit Whenever having a single field on a page that needs editing, and the readability of this field is more important than the ability to edit, keep the edit function hidden until the user interacts with it. To improve accessibility, an alternative to the inline edit could be provided on a separate page.

Multi-Field Inline Edit Used whenever there are multiple field or more complex editing. This type of editing has a potential to be more disruptive since the form for editing takes up larger space than just displaying the information. Consider using a **Overlay Edit** or a separate page if the form grows to large.

Overlay Edit If the user's full attention is demanded (the editing is important in it's own right) or you don't need the inline context for editing, consider editing in an **Overlay Edit**. This brings the editing form into a layer above the page with a lightweight pop-up instead of editing directly in the flow of the page. **Overlay Edit** should also be used when the form is large and needs a clear editing area (often when the content is not frequently edited).

Table Edit Used for grid editing. Here the display of data is more important then the ability to edit. Avoid hover for activating edit mode and instead use single click. Use normal conventions of cell navigation from common spreadsheet applications.

Group Edit Provides a way to balance between visual noise and discoverability, when dealing with multiple items on a page. The mode implies that you enter edit mode for a group of elements (e.g. rearranging icons). When done editing you exit the mode in the same way as you entered it. The advantage of this is that you keep the display uncluttered, but you might have a problem with discoverability at the same time.

Module Configuration Provides the ability to control the amount and type of content displayed in a module of a page. This is done directly in that module and not on a separate page.

B.6 Drag and Drop

Drag and Drop is moving objects around directly with the mouse. Basically grab an object and drop it somewhere. There are at least 15 events available for the user during an drag and drop interaction and a number of actors you could visually manipulate. To capture and plan these complex interactions you could use a moments grid. It serves as a checklist when making sure there is no flaws in the interaction. Keep the user informed throughout the whole process.

Since drag and drop can be hard to discover, the ability to drag is good to indicate. This can be done for example with cursor changes or revealing of drag affordance on mouse hover. Availability can also be hinted when using

alternative functionality. Drag and Drop should only be initiated when an object has been dragged a certain distance (usually three pixels) or the mouse button has been held down a certain time.

Drag and Drop should never be forced but instead should it be used as a shortcut mechanism for performing direct actions in the interface. Alternative ways of doing the task should be provided. Drag and drop should follow the natural representation of the objects in the interface.

Drag and Drop Module **Drag and Drop Module** is used when rearranging modules on a page. It lets the users decide where they want the content. When doing this feature you should make sure that you use a placeholder (to avoid jitter an insertion bar could be used) to show a clear preview during drag. To determine drag position, use the midpoint of the dragged object.

Drag and Drop List Like **Drag and Drop Module** but instead you do it in one dimension. Same considerations as the latter but instead you use the mouse position for drop target positioning. Alternative ways to rearrange the list should be provided since drag and drop might not be easy to discover.

Drag and Drop Object When expressing relationships between objects that are visually represented, drag and drop can be used to make changes. If these relationships are complex, insertion targets for drop locations can be good to display. For parent/child relationships, the parent can be highlighted as well to indicate drop location.

Drag and Drop Action **Drag and Drop** could be used for invoking actions on a dropped object. Throwing files in the trash to delete or uploading files from desktop to browser with a drop target are examples of this. This action is not commonly used in web interfaces as they are not discoverable. It could be used when an action have to be repeated often, for example when uploading several files.

Drag and Drop Collection **Drag and Drop** could also be used when collection objects in a temporary area (e.g. a shopping cart). This could be used as an alternative way to collect items.

B.7 Overlays

Overlays are lightweight pop-ups that are displayed within the browser page as a layer over the page. These demand less system resources than normal browser pop-ups and doesn't display browser interface controls. Except that they are inexpensive to create, they are also controlled by the web application and not the browser, which gives total control over the visual style of the overlay and can be more integrated in the application design.

Overlays should be used when interrupting a process or when there is a multi-step process. Moreover overlays should be used when there are more than one place a dialog could be activated from.

Dialog Overlays **Dialog Overlays** replaces the old style browser pop-ups. These overlays can be modal or non-modal. A modal overlay requires the user to interact with the overlay before she can return to the application. A non-modal overlay allows you to dismiss the dialog and interact with the main page.

Display Overlays should always be used instead of browser pop-ups. Remember not to overuse them (**Idiot Boxes**), as they can interrupt the user's flow. Often can simpler, in-page interaction suffice. If an overlay contains important information and should not be ignored, the overlay should be modal and emphasized with the **Lightbox Effect**.

Detail Overlay **Detail Overlays** allows an overlay to present additional information when the users click or hover over a link or section of content. These can request additional content from the server without refreshing the page, and can be created across different browsers.

Use **Detail Overlays** to show additional information to avoid unnecessary page transitions. Activation of these should be simple with a click (when getting details should be explicit) or a mouse hover (when it's not obvious how to get more information). Activation and deactivation should be symmetrical to give the same amount of effort for dismissing an overlay as opening it.

Input Overlays **Input Overlays** is a lightweight overlay that brings additional information for each input field tabbed into. These are used when you want to simplify the visual style of a form (for example; displaying additional help only when necessary). This technique tends to give clear focus from field to field.

B.8 Direct Selection

Direct Selection is when applying actions to directly selected objects. An example of this is when initiating a selection when clicking on an icon.

Toggle Selection **Toggle Selection** is the most common and easiest form of discontinuous selection on the web. This is basically checkboxes and toggle buttons. Once items have been selected, actions can be performed on them.

Toggle Selection are used when selecting elements in a row. In addition to the checkbox-select, the row should be highlighted to make the selection explicit. When moving from page to page, actions should only work with items selected on that page. Unavailable actions are to be disabled. Moreover, clear feedback of number of selected items should be provided. You might also consider offering a 'select all' option for selecting over all pages.

Collected Selection Like **Toggle Selection** but selection is made possible across several pages. If this is allowed, selected items should be accumulated into a separate area. This makes the selection explicit even across several pages. Avoid ambiguity between items that can be selected with **Collected Selection** and items that can only be selected in a page.

Object Selection **Object Selection** is when objects are selected directly within the interface. This is most commonly used in desktop applications and is often extended by using the Shift or Control key. Object selection are used when simulating desktop style environments and when selectable elements can be dragged. When browser capabilities are limited, this kind of selection should be degraded to **Toggle Selection**.

Hybrid Selection An combination of **Toggle Selection** and **Object Selection**. It is a nice way to bridge a common web idiom with a common desktop idiom but can if not done right be somewhat confusing. To avoid this; use checkbox selection to select an object without opening it, but object selection when selecting and open an object.

B.9 Inlays

Information and actions can be inlaid directly within the page itself. Instead of navigating to a new page or use an overlay, inlays insert themselves directly into the context of the page. Doing so, inlays get around the problem of hiding important information that overlays sometimes do.

Use inlays when to avoid covering information on the page needed in the dialog. They are also used when showing contextual information or details about one of many items (as in a list).

Dialog Inlay **Dialog Inlays** are used when extending a part of a page. These should be used for secondary tools that aren't primary to the main page flow, for example page customization. The introduction of a page inlay should be done with an quick slide animation and connect the dialog with the element it slides out from.

List Inlay Instead of navigating to a new page for an item's detail or popping up the information in an overlay, information can be shown in it's context in the list. Details are shown when needed and preserved space on the page for high-level overview information. It makes the user understand information when focus is on the whole context. Multiple items can be allowed to be visible for parallel content.

Detail Inlay A common way to provide additional information is using a **Detail Inlay**. When hovering over an item, additional information is shown

inlay (similar to detail overlay). It has the advantage of not covering information when hovering over an item, which avoids the anti-pattern **Hover and Cover**.

Tabs **Tabs** can also be used to bring in content within a page, relieving the user of site navigation. Traditional tabs can be used to navigate through sections, content tabs to switch content in a module, and personal assistance tabs to make interaction feel deeper. Personal assistance tabs are activated by hover to reveal content.

Tabs are activated with mouse click and should not be used multiple times on a page. However, if this is done, creating a visual contrast can distinguish the tabs. The most important information are to be put in the first tab, since users may not navigate to the other tabs.

B.10 Feedback Patterns

Feedback patterns is help to keep the user informed about what is happening in the application and provide interactive feedback accordingly.

Live Preview **Live Preview** lets the user view a glimpse beforehand of how the application will interpret their input once submitted. Determining of password strength during account sign-up, username availability check, or showcasing of products, are all examples of this. It is also used to engage users and to avoid page transitions. Feedback is given immediate and the preview is in context of the action. If performance permits, real-time previews during character input can be provided. Alternatively, trigger on leaving the field or provide a clear call-to-action button to initiate the preview.

Progressive Disclosure **Progressive Disclosure** is used when users are faced with a series of steps, and hints is provided when needed along the way. The interface becomes less cluttered this way, instead of having all hints at once. The pattern is related to **Live Preview**. **Progressive Disclosure** has the advantage of making a heavy process feel more lightweight. Information is revealed just as it is needed and removed when the user leaves the input field. It might be appropriate to hint at what is about to be revealed.

Progress Indicator When the application is busy with a lengthy operation, it creates the opportunity for a reactive interface with **Progress Indicators**. It helps keep communication going with the user when the rest of the interface is currently unavailable. This is common on search-based application and heavily used in desktop applications. **Progress Indicators** are helpful for showing upload status, and to provide real-time feedback (e.g. characters left to write).

Progress Indicators is nice when you want to improve perceived performance. Indicators should be kept simple and in close proximity to where the action is. When the focus is on the user input, indicators should be put next to the input field, and when focus is on the result, progress indicators should be

put over the results area. If possible the indicators should show actual progress, alternatively something cyclical.

Periodic Refresh **Periodic Refreshes** brings fresh content on a periodic basis without direct user interaction. It is done when the interface is reacting to the larger community as a whole instead of the user's input. The content is kept fresh and it creates a sense of relevancy. Updates must be balanced between readability and relevance. It can often be best to allow the user to pause the automatic refreshes if the user feels the refreshes is done to fast.

B.11 Lookup Patterns

Since a large aspect of application interfaces is looking up information, lookup patterns are of great importance. Any real-time feedback will be valued as users work to complete their tasks.

Auto Complete **Auto Complete** is a lookup pattern that benefits from a reactive interface. When a user types into a field, a drop-down menu of matching values are displayed. **Auto Complete** is used for input assistance. It should match on multiple fields and show results when the users pauses typing. On Tab key, the matched value should be selected. The user should never have to scroll through the drop-down to choose the already selected item.

Live Suggest **Live Suggest** provides real-time search term suggestions for creating a search. It aims to narrow toward the user's goal and avoid distracting her with needless information, not unlike **Auto Complete**. **Live Suggest** is used for suggestion assistance, not for direct search results. Considerations for **Auto Complete** also goes for this pattern.

Live Search Much like **Live Suggest** and **Auto Complete**, but instead the search results are shown in real time. The pattern creates a tight feedback loop directly with the search results themselves, and becomes a powerful way for users to find what they are looking for. **Live Search** should be used for free-form searches, and provide generous context for each result returned. Results need to be returned quickly, but can be a difficult task, since it can be a expensive operation. Enough results should be shown to narrow in on a solution, without being distracting. If possible a combination of **Live Suggest** and **Live Search** can be good to use.

Refining Search A variation of **Live Search** which provides sets of live filters that allow the search results to be refined in real time. It is often used for product searches when there are multiple facets to filter. Avoid page refreshes since it can cause the user to loose context. Unnecessary refinements (when the user is still tweaking) can be avoided using proper timing delays or event-based execution.

B.12 Invitations

Static invitations provide cues directly on the page using visual techniques to invite interaction. By doing this we can statically indicate to the user the expected interface behavior. Invitations can also be used to get the users attention at the moment when they need it. Dynamic invitations is used at the point of interaction or as a guide through the next step of interaction.

Call to Action Invitation An **Call to Action Invitation** are generally static instructions on the page for primary actions or to call out 1-2-3 steps. These invitations should be made visually stimulating to get the user's attention. Competing visual clutter are best to avoid. Empty areas can be seen as opportunities to call the user to action. Unfinished areas can be left as an invitation for the user to complete the task.

Tour Invitation Closely related to **Call to Action Invitations** are the **Tour Invitations** that often is used as a way to introduce new features on a site. **Tour Invitations** could be used to guide users through a series of features, and is best integrated in the live site as much as possible. Tours is made short, simple, easy to exit and clear to restart. Just don't think that tours make a difficult site easy to use.

Hover Invitation A **Hover Invitation** is when hovering over an item to show an invitation to further action. They are used when actions are secondary to content and the visual style is kept without jitter. Keep it clear what the invitation is referring to by placing the invitation in close proximity. Cursor change, background change, and tool tips can be used to clearly indicate the invited action. If possible, a preview can be showed of what happens if the user executes the action. It can also be good to bridge the new invitations with old familiar idioms to make the user understand more quickly.

Affordance Invitation New unfamiliar interactions are best to bridge with classic unfamiliar idioms. These invitations should be placed in context, within proximity to the interaction. Perceived affordance can be used to cue an invitation.

Drag and Drop Invitation This is simply invitations to indicate **Drag and Drop**. There are at least 15 different events throughout **Drag and Drop**. As many moments as possible should, through subtle invitations, be used to keep the user engaged. Draggable areas are always to be provided with a cursor change. Also provide an unequivocal space to grab items for dragging.

Interface Invitation During interaction, visual interferences can be used to cue users as to what the user has inferred about their intent. This is often used

in sketch applications. With an intelligent engine these cues can make complex interface combinations clear.

More Content Invitation When appropriate, animation can be used to indicate that there exists more content. If there is more content, hover can be used to reveal its existence or letting some of it be slightly revealed as a sneak-peak.

B.13 Transitions

Transitions make changes on the web appear more natural. They create a richer and more compelling experience and the interface seems more alive and reactive. However, going overboard with these techniques can distract from what is being communicated.

Transitions makes a story more compelling, fills in the hard jumps, and make an action more concrete and believable. They give us a way to maintain context while changing views, explain what just happened, show relationships between objects, focus attention, improve perceived performance, and create the illusion of virtual space.

The more rapid a change is, the more important an event seems. Rapid movement is often seen more important than a color change. Movement toward the user is seen as more important than movement away from the user. Slow change can be processed without disrupting the user's attention. By seeing an object move from one place to another, movement can communicate an object's new place. Symmetry of interaction is important using transitions. Reverse transitions should be activated from the same spot as the previous.

Transitions should be near the user's area of focus, as it will make the transitions more discoverable and make them feel less like advertising. Overusing transitions with gimmicky effects can distract more than communicate, as with the overuse of ads. Communicating changes in the interface is rarely done good when relying solely on transitions.

Brighten and Dim **Brighten** and **Dim** can be used to control the user's point of reference. Brightening an area of the screen focuses attention there, while dim on an area causes the elements to be treated as secondary or not in use. These effects communicate importance, whether objects are in use, and if the interface is ready for interaction. A common technique to use **Brighten/Dim** is the **Lightbox Effect**. An overlay is showed in a normal non-dimmed state. This brightens an area and dims the rest.

Brighten and **Dim** can also be used to indicate whether or not an interface is active. Visual noise can be decreased when elements that are secondary or not in use are dimmed. It can also indicate that an element is not ready for use, perhaps when an application is being loaded.

Expand/Collapse **Expand and Collapse** is used to control a panel's visibility in the flow of the page. It is a typical way to bring inlays into a page. An animated transition is often provided to connect the control that activates the **Expand/Collapse** with the panel. It is more eye-catching than **Brighten/Dim** since the movement is often more dramatic.

The effect can be used to manage lots of content or modules, provide details about an item in a list, or make content available for edit.

Self-Healing Fade When deleting or moving items, it often exposes a hole where the object being removed once lived. This effect is used to animate the closing of that hole. It is used when you remove an object from a list or grid and convey that the removal happened and where the object was removed from. The effect can also indicate the completion of a drop operation.

Animation **Animation** is a great technique used to build association between elements on a page. It can be used to smooth out transitions or show how an object has changed places or containment on a page.

Spotlight **Spotlights** are useful to temporarily call attention to a change in the interface, or to different parts in the interface that might normally not be noticed. The **Spotlight** is often accomplished by first highlighting the background of an object, then fading out the highlight.

B.14 Internationalization

Internationalization (I18N) is the process of designing an application so it can be adapted to various languages without needing changes in the code. Related to internationalization is localization (L10N) which can be described as the process of adapting an application for a specific region by adding locale-specific components and translating text. The process of internationalization helps incorporate necessary flexibility and adaptability in web applications to meet a global audience.

Extensible Design If meeting a global audience, web applications should be designed in such a way that, a complete overhaul of interface and content, never becomes required. This means, country- and language-specific elements need to be identified and replaced with localized ones.

Since different languages have different ways of emphasize text, this can make localization difficult. Therefore it is important, to avoid using presentational tags like `` and `<i>` and instead using their semantically equivalent tags such as `` and ``.

Especially when designing pages with form elements and tabular data, you have to take text swell into consideration. Often words in other languages take up more room (e.g., non-English languages take up to more than 30-40% more

space on average than English) which need to be considered. Labels are to be placed above form elements and fixed width should not be forced.

Never make pictures country- or language-specific. If you have text embedded in pictures you need to create pictures for every translation. This is probably something you want to avoid. Try keeping images cultural-neutral, since images and signs might be interpreted differently by different cultures.

Text constructed using both static and variable text fragments (referred to as variable text or concatenated strings) should accommodate ways of reordering the structure of them. This is important to make localization easier since different countries use different ways of presenting information.

To allow the maximum coverage in terms of language support, use the UTF-8²⁷ encoding format. Below is an example how to do it in XHTML²⁸.

```
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
```

Date Format When displaying date formats, you have two options to choose from.

1. Use a locale-neutral ISO 8601 recommended format. This format of `yyyy-mm-dd` (year-month-day) is the solution most commonly adopted by web applications with a diverse user base.
2. An approach to make the date more natural for users, is to use a format that make the month and year more obvious. This approach uses a name for the month and four digits for the year.

Time Format If possible, use local time conventions and show times in users' local time zones. If not possible, use ISO 8601 recommendations (24-hour system with `hhmmss` or `hh:mm:ss` format). Let users specify their time zone.

Number Format Follow the conventions, formats, and regulations of the country from which users are accessing the application.

Currency Format In applications where users might encounter prices (e.g., e-commerce sites) in a currency other than the one they wish to transact, it is a good idea to allow them to change the derived currency. These prices should be shown in their native format and the exchange rate along with the currency conversion.

Language Selector Provide users with available language options, and the ability to set the language on the page content. To make it easy for users to determine if their preferred language is available, use the native words for a languages. Remember the user's language choice.

²⁷Unicode Transformation Format-8-bit version

²⁸Extensible Hypertext Markup Language

Geolocation Geolocation refers to identifying the geographic location of a computer on the Internet. Knowing the location of the user, web applications can direct users to an region specific version of the application. It can also make a reasonable guess about users' language, currency, time and so forth.

Alternatively, a global gateway page can be shown, where the users specify their region. This will redirect them to a local version of the application. Remember to save this information, but also give them the opportunity to change it.

B.15 Accessibility

Accessibility in the context of web design, refers to designing web interfaces such that they are available to all users, including those with visual, physical, auditory, cognitive or other disabilities.

Progressive Enhancement Today users have a variety of browsers to choose from and they are used with different versions. It is therefore important to support user who have older browsers or assistive technologies to access content and functionality on web applications. This way the widest possible reach in terms of platform use can be met.

The first thing to do is use web standards to mark up and structure page content without using client-side scripting technologies. The next step is to improve the visual representation using style sheets without compromising the web page usability and accessibility. The final step is to use client-side scripting technologies to make the application more responsive and efficient to use. Also make sure that Ajax features work when JavaScript becomes unavailable.

Semantic Markup It is important to have a consistent presentation of the web application. Therefore it is good to use **Semantic Markup** to obtain this. What this means is selecting and using HTML²⁹ tags such that they match the desired structural intent of page elements and do not rely on the presentational effect as interpreted by browsers. For example, headings, paragraphs and lists should be marked with available HTML markup tags such as `<h1>`, `<h2>`, `<p>`, ``, `` and not tags such as `<div>` and `` with any inherent meaning.

To control presentation and layout of the page we use CSS³⁰. All style declarations are put in separate files to enable easy updates. Style sheets are not to be used to simulate structural markup.

Unobtrusive Style Sheets With so many different browsers, it is important that we don't totally rely on style sheets. Style sheets need to be added to support different usage and browser context. The page should handle if the CSS is disabled, unavailable or badly supported, so it can satisfy the widest possible

²⁹Hypertext Markup Language

³⁰Cascading Style Sheets

audience. The style sheets should also be developed so that they support that users have the flexibility to adjust the presentation for their own readability.

Unobtrusive JavaScript JavaScript is used to make advanced functionality on web pages. The problem with this is that not all browsers can take part of web pages that have JavaScript. Therefore JavaScript should be incorporated in such a way that it doesn't affect users' ability to use the application when not having it available. To do this we need to keep markup and JavaScript separate. Have JavaScript in external files rather than embedded in the page. The DOM³¹ scripting approach can be used to attach functions to page events. In addition, never use dropdown lists to initiate navigation or form submissions.

Accessible Forms Forms need to be designed in regard to the use of keyboard or assistive technologies. Therefore forms should be laid out with appropriate accessibility tags. At minimum, form elements should be associated with `label` tags and group related forms with `fieldset` tags. In addition, support tabbing through elements using the `tabindex` attribute.

Accessible Images Provide text alternatives for images to help people with visual impairments. This will help screen reader users to understand the purpose and functions of images. This can be achieved simply by using the `alt` and `longdesc` attribute for image tags.

Accessible Navigation There should be a way for users to navigate directly to the main content. It is easy for us who can view pages, to navigate on them, but if you use a keyboard or have visual impairments it can be a lot harder. Add *Skip navigation* or *Skip to main content* links to make it easier for screen readers. Also, use headings in the markup to identify page structure.

Accessible Alternative When using technologies like RIAs³² it becomes hard to make them accessible. To handle this problem, we need to create alternative web applications that offers the same content and functionality, but using simpler technologies. Develop alternative versions of these applications.

B.16 Visual Design

Visual design not only plays the role of reflecting how usable an application is perceived, it also affects how credible it is considered by users.

³¹Document Object Model

³²Rich Internet Application - Web applications that have most of the characteristics of desktop applications.

Liquid-Width Layout Since users today have different resolutions or sizes of their screen, web application should not be designed for one resolution. Try to use **Liquid-Width Layout** so that the presentation of the page will match users' browser in width. This will minimize the horizontal scrolling for users. Elements like sidebars and navigation are best kept in fixed width while the main content is kept flexible. It is also important that the relative positioning of page element maintains their alignment at different window widths. This is the layout preferred by most users³³.

Fixed-Width Layout Using **Liquid-Width Layout** can sometimes create empty spaces between elements if the user has large resolution. This can make the content feel disconnected and visually unappealing. The solution is to create a design with fixed width to ensure that page components remain together. The web page content is set to a certain pixel unconcerned of browser size. The designer has complete control over the placement of page elements which allow them to ensure identical layout between browsers. This design is best suitable for applications that do not demand excessive horizontal space.

Optimal width for the largest number of users is an design for 800 X 600 (width is set between 750 to 760 pixels) resolutions, but it is also common with designs for 1024 X 768 (width is set to between 960 to 980 pixels) resolutions.

Fixed-Width Layouts should be centered on the page to reduce the perception of empty space for larger screens. Appropriate backgrounds with color or images should fill out empty space to make it more visually appealing. Don't forget to make the pages printer friendly.

Progressive Layout Some users with a large screen resolution may find pages with a liquid layout difficult to read and prefers **Fixed-Width Layouts**. Users with small screens may on the other hand find layouts with fixed width difficult because it forces them to scroll horizontally. To avoid this problem, use **Progressive Layout** to maintain a good readability for the user. **Progressive Layout** works so that there are two minimum and maximum values for width. If a user has a browser that is inside this scope then it will use **Liquid-Width Layout** to view the pages. But if it is below or above the values it will use the **Fixed-Width Layout**.

Grid Structure To help users understand relationships between page elements, you could use a grid-based structure for placing and aligning web elements. It creates appropriate visual hierarchies for understanding of content organization. When using a grid, pages are divided into rows and columns. Often the grid is established with the rule of thirds, or the golden ratio³⁴. Page

³³Michael Bernard and Laurie Larsen. *What is the Best Layout for Multiple-Column Web Pages?*. Usability News. July 2001, Vol. 3 Issue 2. Viewed in 2010-02-04. <http://surl.org/usabilitynews/32/layout.asp>

³⁴The ratio between two segments such that the smaller (*bc*) segment is to the larger segment (*ab*) as the larger segment is to the sum of the two segments (*ac*), or $bc/ab = ab/bc = 0.618$.

elements are placed along grid lines and with each other either vertically or horizontally.

Visual Hierarchy To let web pages make sense to users, it becomes important to establish visual hierarchies. This way users know what information is important and what is of secondary use. In addition, it centers the users attention, and creates a sense of order and balance. To do this we use visual cues, to clearly indicate grouping and the order of elements. Visual components used to establish a hierarchy is color, size, alignment, contrast, font sizes, shapes and so forth. They will create a contrast to the desired element and draw the users attention.

Highlight Highlight is used to direct focus on an element or changes. It also provides visual feedback about selected elements. Highlighting can be done by changing the background color, text color, border size, use animation, or make a text bolder. Highlighting must be kept consistent for same kinds of actions. When there are something that requires user response, the **Lightbox Effect** can be used to emphasize highlight changes.

Icons Icons are used to represent commonly used objects and actions. They also make you recognize actions faster. Most important is to use icons familiar to the user. See to it that they clearly suggest the object or action they represent. Supplement icons with labels or tool tips to make actions even easier to identify. Avoid text in them since it will be hard or impossible to translate them to another language. Try to have consistency in the icons visual style but remember to make them distinguishable from each other.

C Building a Pattern Library for Klarna

Design patterns offer benefits as proven design solutions, guidance for their use, consistency, reusability and so forth. To harness these abilities, it is important that these patterns are documented and made available in a format which promotes reuse. In the absence of a formal process for documenting and reusing patterns, the design process becomes quite inefficient. It only seems strange and a waste of time that a designer tries to solve a problem when an already successful implementation exists.

The solution to this problem is building a repository (i.e., a pattern library), where the knowledge of design problems and solutions are documented. This library should be shared among developers as their collective knowledge base. A repository like this creates many advantages. In a corporation you would want to strive towards having consistent interfaces. It help strengthening usability and the corporate brand. Not having a consistent design may very well do the opposite. For developing teams, pattern libraries can increase efficiency and minimize duplicate work.

C.1 Documenting Patterns

Interestingly enough, there are no patterns for documenting patterns. Pattern authors all have their own approach of notation. We have adopted Pawan Voras approach which is rather minimalist, but has quite a structured presentation. Each pattern should contain the following sections:

Pattern name The name of the solution that clearly communicates what the pattern stands for.

Problem A brief description of the design problem(s) together with design challenges and possible trade-offs.

Solution The core solution to the problem, with examples.

Reason of the design solution (why) Explanation of the solutions efficiency, based on empirical research, design principles, or heuristics.

How to apply the design solution (how) Best practices for solving a problem and the possible variations, together with illustrative examples.

Related design patterns One pattern is seldom used alone to solve a design problem. Because it is often the case that several patterns are used together, this section identifies related patterns to consider when solving a problem.

Research Evidence for the patterns validity and strength.

C.2 Best Practices for a Library

- Include reusable design components, that can be applied immediately (or possible with certain tweaks). Apart from pluggable solutions, code snippets, visual components and wire-frames can be included.
- Include lots and lots of examples. Patterns are problems that occurs over and over again, where a solution for it have been found. Examples of already successful implementations proves the patterns strength and gives the designer concrete ways of solving a problem.
- Include anti-patterns. Anti-patterns are design practices that should be avoided. They should be referenced whenever a designer might employ its practice.
- Encourage participation by enabling ratings and comments about patterns. This can help improve patterns.

- Create a formal process for managing pattern libraries. To ensure the library stays relevant it is important to monitor and update it, just like any other software application.
- Finding patterns must be an easy task. Consider including related patterns, searching, filtering, and tag clouds.
- Use a wiki for development of the pattern library. A wiki contains all the tools necessary for managing a good library and in all likelihood increases participation.

C.3 Using Patterns

Simply, patterns are abstract blueprints, and are not in any way precise design solutions. Apply them as they fit your purposes and specific problems. They are starting points, or strategies, that designers can adapt and refine to develop creative solutions.

Appendix Part II

Instructions

D Running the Proptype

The prototype is delivered as a tarball. In an Linux shell, run:

```
$ tar -xvzf prototype.tar.gz prototype #Untar
$ cd prototype #Application directory
$ make #Compile
$ ./start.sh #Start the application
```

When the Erlang shell is started, run:

```
1> kB_db:reset(). %Reset mnesia
2> dashboard:init_user(). %Initialize users
```

The prototype can then be viewed at <http://localhost:8000>

Log in with user: *user1* and password: *user1*

D.1 Common Problems

Login fails.

Make sure you have initialized the users with `dashboard:init_user()`. Sometimes log in fails almost randomly. Then keep trying logging in or change browser.