TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

IDK40LT

Vjatšeslav Sobolev 112661IAPB

LIFE AND PENSION INSURANCE SELF-SERVICE DEVELOPMENT FOR LIFERAY PLATFORM

Bachelor's thesis

Supervisor:	Jekaterina Tšukrejeva
	Master degree
	Lecturer assistant

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

IDK40LT

Vjatšeslav Sobolev 112661IAPB

ELU- JA PENSIONIKINDLUSTUSE ISETEENINDUSE ARENDAMINE LIFERAY PLATVORMIL

Bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
Magistrikraad
Õppejõu assistent

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Vjatšeslav Sobolev

23.05.2016

Abstract

The main target of this thesis is to implement simple and easy-to-use self-service demo prototype, which run as separate portlet on Liferay portal. Portlet provides opportunity to make pension insurance online, without insurance company office visit. Pension insurance self-service gives information to final user about future pay-outs, payments and other policy parameters.

Thesis describes requirements to self-service and portlet technology. Analysis of possible portlet development technologies was done is order to choose optimal technology solution according to requirements.

During thesis work, best technology solution for self-service portlet development were chosen and demo self-service portlet was implemented.

This thesis is written in English and is 37 pages long, including 4 chapters, 5 figures and 2 tables.

Annotatsioon

Elu- ja pensionikindlustuse iseteeninduse arendamine Liferay platvormil

Peamine lõputöö eesmärgiks on rakendada lihtne ja mugav iseteeninduse portlet, millest saab osa Liferay portaalist. Iseteenindus portlet annab võimalust teha pensionikindlustuse läbi veebi, pensionikindlustuse kontori külastamiseta. Pensionikindlustuse iseteenindus annab võimalust lõppkasutajatele vaadata ja muuta sissemakse, väljamakse ja muud pensionikindlustuse taotluse parameetreid.

Bakalaureusetöös on kirjeldatud nõudeid iseteeninduse rakendusele ja portlet tehnoloogiale. Analüüsi käigus oli tehnoloogia valitud.

Lõputöö kaigus oli püstitatud probleem lahendatud ja demo prototüüp oli implementeeritud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 37 leheküljel, 4 peatükki, 5 joonist, 2 tabelit.

List of abbreviations and terms

REST	Representational state transfer
UI	User interface
IE	Internet Explorer
JSF	JavaServer Faces
IDE	Integrated development environment
HTML	Hyper Text Markup Language
API	Application Programming Interface
SPA	Single Page Application
CSS	Cascading Style Sheets
IBAN	International Bank Account Number

Table of contents

List of figures

List of tables

1 Introduction

The aim of this thesis is to implement self-service portlet to Liferay [1] portal.

Liferay portal popularity is growing, and usually is used for business solutions and big companies.

Self-service will provide opportunity to make pension investment policy by using portlet in website. Self-service will use main pension investment application with REST [2] to get, validate and commit data. In nowadays providing self-services to final user is very important part which gives opportunity to get a new customer.

Technology possibility to develop Liferay portlet will be analyzed and chosen for self-service in analytics part of this thesis.

1.1 Problem

This thesis describes real demo solution of self-service for a software developing company producing pension/group pension investment product. In this case final user will invest money in his own pension. Big part of people is investing to their own pension, but for that they must visit invest company office to complete a contract, to make that procedure shorter was decided to make self-service for that, where final user can make investment contract by using web application in his browser. In that case this application must be very stable, secure and simple to improve.

In final case of self-service, final user can watch status and manage investment amount, investment/payment periods, check bills and other opportunities which is available in real office, but without visiting investment company office, that will save big amount of time as for final user such as for investment company.

1.2 Thesis aims

Main goal of this thesis is to create a self-service portlet application with easy availability to continue development.

Sub goals:

1. Analyze requirements
2. Analyze Liferay portlet development technologies for solving that problem.
3. Implement application.

1.3 Thesis workflow

Thesis have three main parts.

Chapter 2 describes functional and non-functional requirements to self-service and to technology what will be used to make it.

Chapter 3 describes the technology choice, enumerated and analyzed most trend technologies to make Liferay portlet, and reason of technology choice.

Chapter 4 describes self-service system and implementation details of components.

2 Requirements

Main goal of the application is to supply to final customer possibility of making policy without visiting investment company office.

In thesis Author will describe part of whole portlet. Described part is based on portlet core functionality development and on details frame parts and components development for details frame.

This application will be developed to show demo process of pension insurance self-service possibilities to customer.

Core functionality what will be described in this thesis is parameters handling process.

Details frame [3] is frame where final user will specify additional policy information, in this thesis frame means one page of portlet UI [4], which final user can see at the moment. Frame will contain two Option Groups and two event Text fields as main components and secondary components such as labels and numbering. Details frame initialize all components (other beneficiary filed and IBAN field will be hidden from UI) and set default values to option group fields (Beneficiary in case of death and Recurrent payment method), in case of other beneficiary parameter is selected then additional parameter text field will be shown.

2.1 Requirements to application

2.1.1 Functional requirements

Functional requirements to self-service portlet:

- Possibility to change policy parameters (beneficiary in case of death, recursively payment method).
- Possibility to get overview of all user parameters which final user is specified during policy creation process.
- Each time when parameter change event is fired, then handling process pass changed parameter to main controller.

2.1.2 Non-functional requirements

Liferay portlet development technology:

- must be chosen to have good availability in future improvement.
- have a simple way to create UI component for final user.
- have a simple way to manage UI components, their content, composition and style.

Self-service:

- Working same in any browser [5] (IE11 [6], Microsoft Edge [7], Chrome [8], Firefox [9], Safari [10], Mobile Safari, Mobile chrome, Mobile Windows Phone)
- Have good looking user interface.

3 Choosing Liferay portlet development technology

For technology analysis where chosen couple of technology which Liferay portal support and they are in trend in nowadays. In that work Author compare different technologies to verify most effective one, to solve upcoming tasks.

Liferay portlet technology is one most important part of all application, it will tell what language will be used to make portlet to live, and how much time will be spent to implement upcoming tasks. In other kind of view, technology must be easy to use, and make complicated UI elements with couple line of code. Technology must be chosen or be already known for developers or be very familiar to already known one.

In this analysis Author will compare next technologies:

- PrimeFaces [11]
- AngularJS [12] + Spring [13]
- Vaadin [14]

Author will compare those technologies by next criteria:

- Development availability
- Code simplicity
- Availability for modify
- Browsers support
- Extendability
- Build time and Deployment complexity
- Testing and debugging possibility

3.1 PrimeFaces

PrimeFaces technology usage will be described on JFS 2 [15] and PrimeFaces 5 framework. PrimeFaces is built for dynamic web pages with server driven architecture, with it is not possible to create single page applications, but provides very flexible UI development.

PrimeFaces development is supported by plugins for NetBeans integrated development environments no plugins for Intellij IDEA or Eclipse. Huge availability to create different templated projects from maven archetype.

Code is very similar to HTML, but contains PrimeFaces framework specific tags to control dynamic components. By default, not contains any styling frameworks for style components and it must be specified separately for each element and later describes styles for them, style injection is available by inserting specific tags inside code, is possibility to inject styles directly to HTML element.

```
<center></center>
```

This tag will tell that all code inside those tags will be centered relatively to parent object their parent element.

To show static data is possibility to use basic HTML elements, to show dynamic data need to be used faces tags.

```
<p:inputText />
```

This code will show simple input text field.

Modification of existing component can be complicated, because is needed to change couple of files at once.

PrimeFaces support next browsers: IE9+, Chrome, Firefox, Safari, does not support Microsoft Edge and IE6, mobile browsers are not fully supported.

Does not encourage extending existing components with java and JavaScript [16].

Build dime directly depends from amount of code and can take seconds to minutes, without using advanced hot deploy possibilities or hot deploy plugins. Build final result is war file what is deployable to server with couple of seconds.

For testing it is possible to use automated acceptance testing tool, load testing with standardized tools like Jmeter [17] or Gatling [18], UI elements is possible to test with Junit [19] tests.

3.2 AngularJS + Spring

In this part Author would like to describe Spring framework as REST-full API as backend [20] and AngularJS framework as frontend [20] SPA [21] framework. This software architecture gives a lot of advantages in fast development, scalability, availability to simply modify or extend application.

First of all, Spring and AngularJS frameworks are stable and actively developed by the communities. They have plenty of ready to use build-in functionality, modules and classes. Spring provides quite simple and understandable syntax for mapping database entities objects and handling REQUST/RESPOSNE objects in backend REST controllers. Using AngularJS framework can easy consume REST API and create dynamic web application. AngularJS template syntax for HTML files is very useful and helps to write a lot of functionality with less code. AngularJS has own $scope [22] object logic for each part of application, that isolates/divides parts of application and keeps them independent.

AngularJS is standardized for all modern browsers and mobile platforms, it gives opportunity to write in same way for all web browser. Developers can easy include and configure existing AngularJS modules with understandable and intuitive JavaScript code.

This technology stack makes backend and frontend independent from each other, giving developers possibility develop them separately in different teams, decreasing development time with more human resources.

AngularJS provides instant build and deployment time, that also increase development time. Bug finding and fixing can be complicated due JavaScript hard debug ability.

3.3 Vaadin

Vaadin is a java web framework, java code will be translated to HTML and JavaScript via compiler, Vaadin has server driven architecture. Vaadin development is simplified with Vaadin plugins witch many integrated development environment provides such

Eclipse or Intellij. Eclipse provides also Visual Designer plugin for quick UI development. You can create a new Vaadin project with pair of clicks or use Apache Maven [23] archetype for build from archetype.

Code is very simple and very easy to understand, has entry point class as main class for application start. By default, Vaadin uses Twitter Bootstrap [24] framework for styling, if you want change that you can specify another default styling framework, or totally remove it. Vaadin provides easy color theme change opportunity, with what you can change whole site color with 10 line of code.

```
Label label = new Label("MY Label");
label.setStyleName("my_label_css");
```

This code will create div HTML element with “My Label” inside it, and add to this div CSS [25] class name “my_label_css” after that you can manage style with usual CSS in attached css file. Label caption can be modified dynamically what is some action completed, like button clicked or other event handled.

```
.my_label_css{
  font-weight: bold;
}
```

This code will add font-weight to label element what is declared before.

Vaadin have big amount of widgets and default UI elements what can be used for build more complex element compositions without creating a new Vaadin Widgets. Huge extendability with java as GWT [26] client and server side components and possibility to extend with JavaScript components for client components.

Modification of already made component composition is quiet simple, but can be complicated when you start changing handling events on different component. Changing style will be very easy, because for that you need only change CSS style for that element or add style class to element and describe it on css file.

As all java applications Vaadin will compile to war file for server deployment. Compilation time will be directly depending from amount of code, and can take seconds to minutes on compilation.

3.4 Technology choice summary

Analysis of technologies in this chapter provided good overview of possibilities PrimeFaces, AngularJS and Vaadin frameworks. Every framework has plusses and minuses. All those frameworks are currently supported and their development is going quiet fast, last official release, of any framework, was about half year ago. All frameworks support fully internalization and localization, they all is possible to test with automated acceptance testing tool and load testing with load testing standards.

Another aspect what will depend on that choice is that main application is using java as backend and Struts 2 for frontend java web framework. That means java must be used as backend language for self-service.

PrimeFaces big plus is that you can make your own very complicated component with all your requested attributes, directly inject java code into HTML with all possibilities java possibilities, minus of PrimeFaces is that code complexity is quiet difficult so development time and support time can take more time that usual can be provided to that issue solving.

The most important advantage of using AngularJS with Spring framework is that server side and client side are totally independent from each other, project has instant build and deploy client side, exists big community for helping developers to solve their problems, minuses can be hard bug fining opportunity, client side code is written on another language what can add some complicated aspect and code is available for final users.

Vaadin big plus is that you will not need to change a code whitening language to write client side part, all code as backend and frontend can be made with java, so it is easy to modify code, GWT technology knowledge will be big benefit if you will start developing with Vaadin, minuses will be that Vaadin deployment is not instant and build take also time, from couple of seconds to pair of minutes, depends of amount of code to compile, some component not working exactly same for all browsers.

3.5 Technology choice for self-service portlet.

For self-service development author will choose Vaadin framework. To that choice affected mostly code simplicity and development lightweight, Available difficultly big amount of prepared UI elements what can be composed to more complicated components. Another important part is that Vaadin is very secure, because all code is handled on server and it is not allowing to somehow affect code process. Another most important part is that styling availability from java code is on top level, that means possibility to add or modify styles dynamically from java code.

4 Self-service development

Liferay portlet development as all web application development is divided to two main parts Backend, Frontend and optional styling part, for UI elements look customization.

Backend in Life and pension insurance self-service will deal with frame composition, data handling and user parameters handling.

Liferay portal provide authorization possibility. This allows to manage authorized and non-authorized user UI frames customization according to authorization status without separate authorization module implementation. Other part will be handled from portlet itself. For navigation between frames was used modified Navigator Vaadin component. Whole portlet is divided to frames, each frame contains specific data to show to final user.

In thesis self-service development will be described on Details frame example. Details frame contains additional fields to complete pension policy. Those fields are:

- Beneficiary in case of death
- Other beneficiary
- Recursive payment method
- IBAN number

4.1 Portlet process flow

Self-service process flow describes final user actions and reaction to those actions, each action will be handled in backend and reaction to that action usually will be shown in UI frame. Reaction to user action can be one from following list:

- Changed frame to another
- Changed UI parameters component value
- Changed visibility of UI component
- Change visualization of UI component

Each frame specialized to special data show to user, in self-service are next frames:

- Welcome frame
- Age clarification frame
- Payout and payment data frame
- Beneficiary frame
- Confirmation frame
- First payment frame
- Contact frame

All navigation between frames is controlled by Navigator [27], which decide with DecisionMaker what frame to show next and is next frame showing allowed in this context and with user parameters.

Main frame flow is divided to two parts, this division depends on authorization status, then if final user is authorized to portal then all up listed frames will be shown, except age clarification age during age is known from final user registration data, if user is not authorized then will be shown next frames: welcome frame, age clarification frame, payout and payment data frame and contact frame. Between frames navigation is made with next and back button, those navigation buttons are available in all frames, except back button in first frame and next button in last frame.

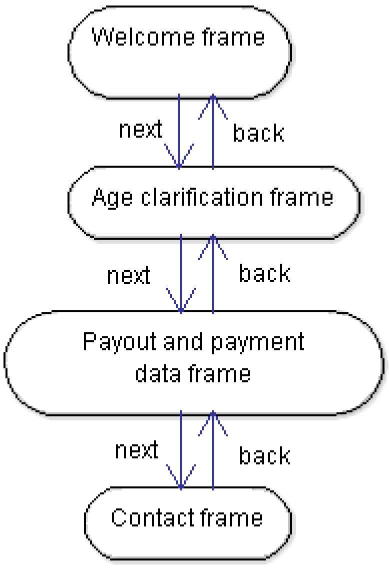


Image 1 Non authorized user frame flow diagram

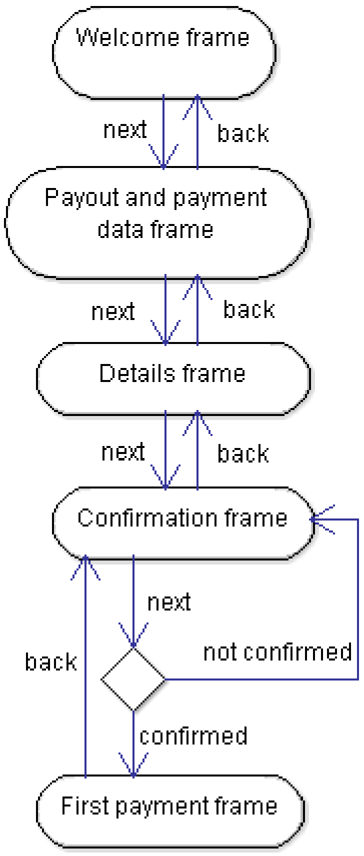


Image 2 Authorized user frame flow diagram

Author provide diagrams, to show visual process flow between frames for authorized user(Image 1) and for non authorized user(Image 2), main different in flow is, that non authorized user cannot make final confirmed policy, not authorized user can only see available payment plans and pay out plan and have possibility to contact with administrator for next steps.

4.2 Backend development

One of the most important in Backend part is user parameters handler. User parameters is needed for store parameters and provide access to parameters from each frame. User parameters is dynamically created for each user and pre specified with default parameters if default parameters are defined. Parameter handlers are wired to specific events, due final user affection to those events parameters can be changed. Parameters store different types of data for example: Date, Locale, Payment amount, Payment frequency and others. All parameters are classified to object types like String, Boolean, Enumeration and others.

Second Backend part is navigation processing and conditions for navigation between frames. In Vaadin Navigation part is handled with Navigator component, to that component Author specify enumeration of frames, and navigator will navigate to next of previous frame according to enumeration.

Third Backend part is data processing part, deals with get or send data from main life pension insurance application. Data processing is totally independent part and not affect to final look and process of details frame, due that case data processing part will not described in this thesis.

4.2.1 Details frame development

Filed for details frame are based on user parameters. In those parameters are specified available values for those fields and default values. For details frame are specified three parameters:

- Enumeration parameters
    - Beneficiary in case of death
    - Recursive payment method

- String parameter
    - Other beneficiary
    - IBAN number

For enumeration parameters is specified value list with all possible values, all possible values for details frame enumeration parameters is shown in Table 1. Default vales for enumeration parameters are shown in Table 2.

Beneficiary in case of death	Spouse
	Children
	Spouse and children
	Other beneficiary
Recursive payment method	e-Invoice
	Invoice list

Table 1 Enumeration parameters value list.

Beneficiary in case of death	Spouse
Recursive payment method	e-Invoice

Table 2 Enumeration parameters default values.

In case of String parameters, no default parameter is specified, due string parameter are specialized to store string type data.

In frame initialization process all default parameter values will be set to according UI components, and will be shown to final user as selected or entered.

```
public enum BeneficiaryType {
  SPOUSE("Spouse"),
  CHILDREN("Children"),
  SPOUSE_AND_CHILDREN("Spouse and children"),
  OTHER_BENEFICIARY("Other beneficiary");

  private String value;

  BeneficiaryType(String value) {
    this.value = value;
  }
  @Override
  public String toString() {
    return this.value;
  }
}

public class BeneficiaryParameter extends
AbstractEnumUserParameter<BeneficiaryType> {

  public BeneficiaryParameter() {
    super(UserParameterType.BENEFICIARY);
  }

  @Override
  public BeneficiaryType getDefaultValue() {
    return BeneficiaryType.SPOUSE;
  }
}
```
This code will show enumeration of all available Beneficiaries and default parameter setup. Similar code parts will be for recursive payment method parameter.

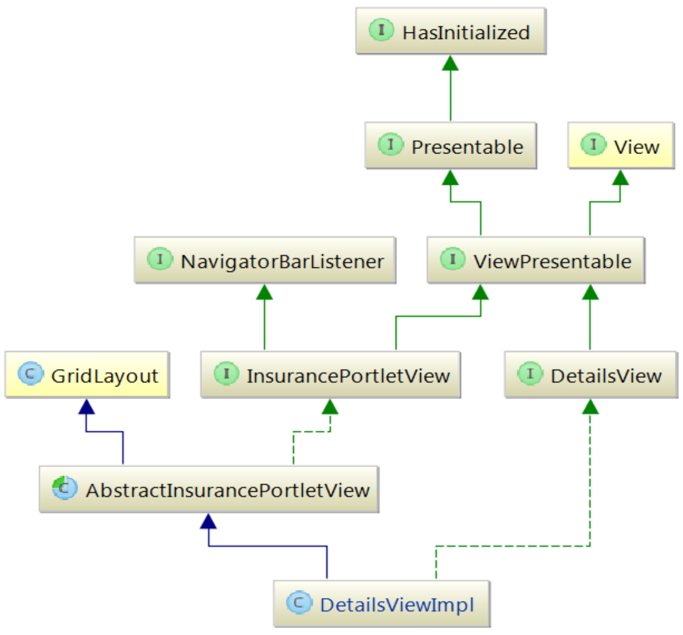


Image 3 Details frame class diagram

Author shows in Image 3 class diagram for Details frame, it has multiple implement interfaces for passing parameters to main controller, and to set default user parameters automatically to fields in frame initialization process.

4.3 Frontend development

Frontend development is second big part of full application, and it is same important as backend. Frontend provide user interfaces to final user, and with them final user can use application, that means frontend must be intuitively easy to understand, have a good look and contains all necessary data.

In self-service Frontend parts are separated to frames and each frame contains some specific data.

4.3.1 Details frame development

Details frame fields are connected to parameters. Enumeration parameters can be introduced as array or arrayList, this possibility provides short way to create Option group.

```
OptionGroup beneficiaryOption = new OptionGroup();
beneficiaryOption.addItems(BeneficiaryType.values());
beneficiaryOption.setNullSelectionAllowed(false);
```

In this code is shown Option Group creation with Vaadin framework, and initialization values from beneficiary type enumeration, and set no availability to unselect value. Similar code is for recurrent payment method Option group. By default, no value in option group is selected, but in frame initialization process default value will be set.

Details frame contains text field for initialize user parameters.

```
TextField ibanNumberTextField = new TextField("IBAN number");
```

This code will create text filed with “IBAN number” label for that field, and blank value.

Details frame contains not only parameter specific UI elements, but also navigation element bar and common frame UI elements, like Frame labels and labels for options group and other UI components, what makes self-service more intuitive for use. Details frame contains next static UI labels:

- Main frame label
- Beneficiary options group label
- Recursive payment options group label

Some UI components is created for multiple usage in many frames, those components can be created in separate package and later used/initialized in needed frame. Shared UI elements, which will be used in details frame are: navigator bar and user parameters summary component.

Navigator bar UI component will exist in every frame and will handle all navigation in self-service. Navigator bar contain back and next button.

User parameters summary UI component will be shown in details frame. Parameters summary component contain Label of this component and list with user parameters key value pairs. Only active parameters are shown at the moment.

4.4 Styling and UI component composition

Frame styling is divided to two parts: usual styling part with using direct css and styling using Vaadin layout components and Vaadin component styling. In both parts styling goes thru CSS, but when Vaadin components are used, then styling will be managed from java code. Vaadin provides all available layout types: vertical layout, horizontal layout, grid layout, form layout, absolute layout, CSS layout and custom layout.

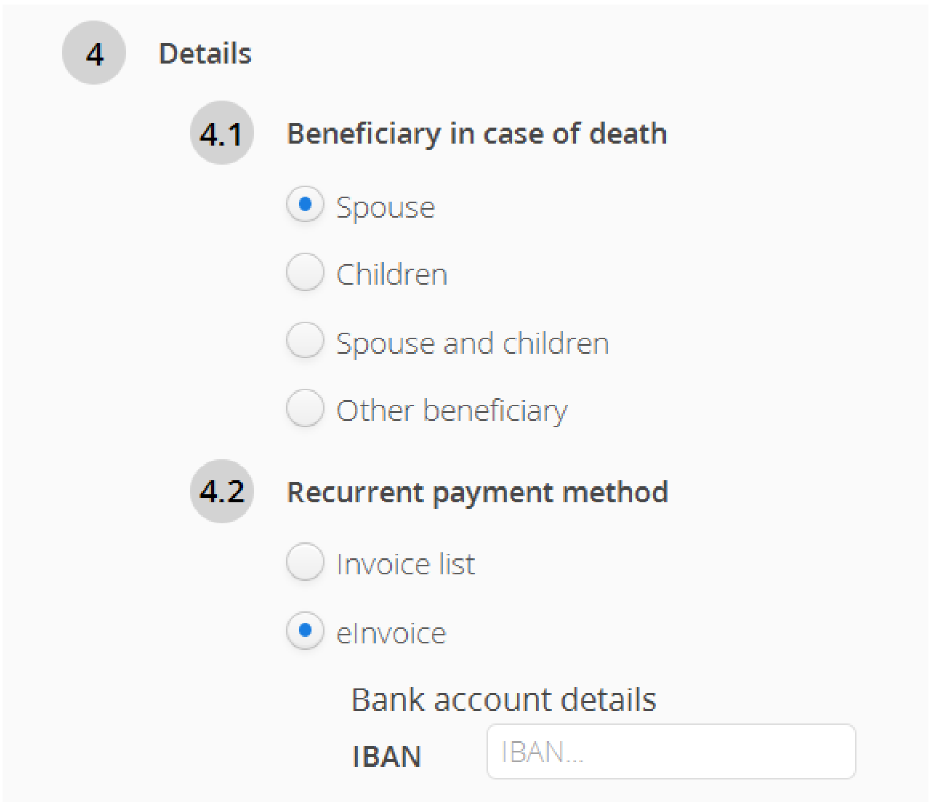
4.4.1 Details frame styling

Details frame extends default frame layout, which is made for setting navigator bar to the bottom of the frame, all other components must be set in details frame initialization.

For main layout Author has chosen CSS layout, this choice depends on mobile view for details frame, due mobile view user parameters summary and main content width is too high and cannot be placed in mobile screen same as in desktop screen. The position of main content will be changed for mobile devices with media queries in CSS. Css layout contains user parameters summary component and content layout. Content layout has vertical layout type and next UI component composition:

- Frame label
- Beneficiary label
- Beneficiary option group
- Other beneficiary text field
- Recurrent payment method label
- Recurrent payment method option group
- IBAN text field

IBAN text field and another beneficiary text field are only available if other beneficiary option was chosen and e-Invoice was chosen for Iban text field, if those options were not selected, then fields are hidden. In image 3 is shown visually this part, Image 4 shows user parameters component. Most of the components in content layout have margins and paddings in their style, this styling is made with CSS.


4 Details

4.1 Beneficiary in case of death

[✓ Spouse]

Children

Spouse and children

Other beneficiary

4.2 Recurrent payment method

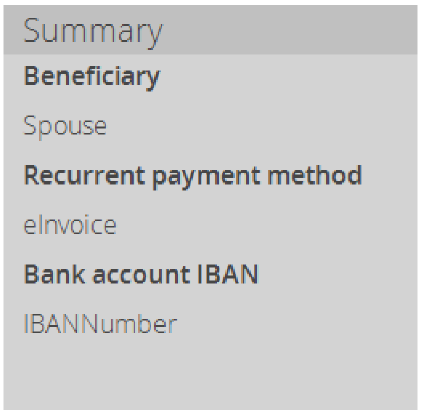
Invoice list

[✓ eInvoice]

Bank account details

IBAN: IBAN...

Image 4 Details frame UI component content layout visual composition.


Summary

Beneficiary

Spouse

Recurrent payment method

eInvoice

Bank account IBAN

IBANNumber

Image 5 User parameters summary UI component.

In Image 4 user parameters summary component shows only available parameters to user and frame if the final user is navigated back to the previous frame then selected is next frame parameters will be hidden from view and not been processed in any processing part. Image 4 shows only example data for selected fields in details frame and not contains other parameters from other frames or components.

Step markers, component before labels, shown in Image 3 is made for better orientation in navigation for the final user. Step maker is made with a simple label and CSS styling to it, numbers to it are set from code.

Mode detailed code is added in appendix part of thesis.

4.5 Development environment

Most of development code were written in Java, for that Author used Intellij as Java integrated development environment(IDE). Intellij Idea provides good opportunity to write code, corrects your typographical errors and provides shortcut keys to access needed part of code and to navigate in project structure.

For project build were used Apache Maven. Maven provides control to lifecycle process and quick deploy opportunity to server.

Git [28] was used for version control system. Git gives very flexible opportunity to make project develop work flow.

4.6 Details frame development summary

In previous parts were shown whole develop process to develop Details frame. All required UI components were set, and handled to store user parameters to backend to forward usage in policy creation.

Due good separation to parts details frame is easy to modify or improve. Each component deals with specific part of process.

Details frame functionality was fully realized, according to details frame requirements.

4.7 Self-service future development

Self-services are very perspective direction of product improvement, due that customers are interested to improve they.

Future improve possibility is very big, can be improved:

- Change pay out amount during pension period
- Change payment plan(payment amount and frequency of payments during year)
- Have possibility to add multiple policies to your account for payment.
- Add possibility to show accumulated income in current working policy.
- Add notification module, what will notify final user for upcoming pay outs or payments.

And many others.

Summary

In this thesis main target was to implement demo self-service portlet for Liferay portal. First main step was Liferay portlet technology choice, in this part analysed possible technologies and best technology, according to requirements was chosen – Vaadin framework. According to requirements demo prototype was implemented.

The result of this thesis is a working demo prototype of self-service portlet to Liferay portal. The working demo prototype can be improved or modified in future.

Also this thesis tested Author developing skills and knowledge, which is gained during studying at university. In addition, Author gained new knowledge during thesis process flow in Java development and in Vaadin framework.

Kokkuvõtte

Lõputöö eesmärgiks oli rakendada iseteeninduse Liferay platvormil. Esimene samm oli iseteenindus rakenduse tehnoloogia valik, selles osas autor tegi analüüsi võimalike tehnoloogiate hulgast ja oli valitud tehnoloogia, mis kõige rohkem vastab nõuetele – Vaadin raamistik. Lõputöö kaigus oli viidud ellu demo iseteeninduse prototüüp.

Lõputöö tulemusena on töötav iseteeninduse prototüüp Liferay portaalis. Tulevikus prototüüpi saab muuta ja modifitseerida edasiarendamiseks.

Lõputöö kaigus autor oli testitud oma nii praktilisi kui ka teoreetilisi teadmisi, mis oli omandatud ülikooli õppimise käigus. Samas autor oli suurenenud oma Java keele teadmisi ja Vaadini raamistiku teadmisi.

References

[1] [Online]. Available: https://en.wikipedia.org/wiki/Liferay. [Accessed 01 05 2016].

[2] [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 01 05 2016].

[3] [Online]. Available: https://en.wikipedia.org/wiki/Frame_%28GUI%29. [Accessed 04 05 2016].

[4] [Online]. Available: https://en.wikipedia.org/wiki/User_interface. [Accessed 02 05 2016].

[5] [Online]. Available: https://en.wikipedia.org/wiki/Web_browser. [Accessed 02 05 2016].

[6] [Online]. Available: https://en.wikipedia.org/wiki/Internet_Explorer_11. [Accessed 02 05 2016].

[7] [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Edge. [Accessed 02 05 2016].

[8] [Online]. Available: https://www.google.com/chrome. [Accessed 02 05 2016].

[9] [Online]. Available: https://www.mozilla.org/en-US/firefox. [Accessed 02 05 2016].

[10] [Online]. Available: www.apple.com/safari/. [Accessed 02 05 2016].

[11] [Online]. Available: primefaces.org/. [Accessed 03 05 2016].

[12] [Online]. Available: https://angularjs.org/. [Accessed 03 05 2016].

[13] [Online]. Available: https://spring.io/. [Accessed 03 05 2016].

[14] [Online]. Available: https://vaadin.com/. [Accessed 03 05 2016].

[15] [Online]. Available: https://en.wikipedia.org/wiki/JavaServer_Faces. [Accessed 03 05 2016].

[16] [Online]. Available: https://en.wikipedia.org/wiki/JavaScript. [Accessed 04 05 2016].

[17] [Online]. Available: jmeter.apache.org/. [Accessed 04 05 2015].

[18] [Online]. Available: gatling.io/. [Accessed 04 05 2016].

[19] [Online]. Available: junit.org/. [Accessed 05 05 2016].

[20] [Online]. Available: https://en.wikipedia.org/wiki/Front_and_back_ends. [Accessed 05 05 2016].

[21] [Online]. Available: https://en.wikipedia.org/wiki/Single-page_application. [Accessed 05 05 2016].

[22] [Online]. Available: https://docs.angularjs.org/guide/scope. [Accessed 05 05 2016].

[23] [Online]. Available: https://maven.apache.org/. [Accessed 06 05 2015].

[24] [Online]. Available: getbootstrap.com/. [Accessed 05 05 2016].

[25] [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets. [Accessed 06 05 2016].

[26] [Online]. Available: www.gwtproject.org/. [Accessed 06 05 2016].

[27] [Online]. Available: https://vaadin.com/api/com/vaadin/navigator/package-summary.html. [Accessed 07 05 2016].

[28] [Online]. Available: https://git-scm.com/. [Accessed 08 05 2016].

[29] [Online]. Available: https://netbeans.org. [Accessed 03 05 2016].

[30] [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 03 05 2016].

[31] [Online]. Available: https://eclipse.org/. [Accessed 03 05 2016].

Appendix 1 – Details frame frontend development initialization part

```
final OptionGroup beneficiaryOption = new OptionGroup();
final OptionGroup recurrentPaymentMethodOption = new OptionGroup();
final TextField otherBeneficiaryTextField = new TextField();
final TextField bankAccountIBAN = new TextField("IBAN");
CssLayout contentLayout = new CssLayout();
contentLayout.addStyleName("detailsView");

VerticalLayout detailsLayout = new VerticalLayout();
detailsLayout.addStyleName("optionsLayout");
detailsLayout.setSizeUndefined();

PhaseHeader detailsPhaseHeader = new PhaseHeader("4", "Details");

VerticalLayout subPhaseDeathCaseBeneficiaryLayout = new
VerticalLayout();
subPhaseDeathCaseBeneficiaryLayout.addStyleName("subPart");

PhaseHeader subPhaseDeathCaseHeader = new PhaseHeader("4.1",
"Beneficiary in case of death");
beneficiaryOption.addItems(BeneficiaryType.values());
beneficiaryOption.setNullSelectionAllowed(false);

otherBeneficiaryTextField.addStyleName("beneficiaryTextField");
otherBeneficiaryTextField.setVisible(false);

otherBeneficiaryTextField.addListener(new
FieldEvents.TextChangeListener() {
  @Override
  public void textChange(FieldEvents.TextChangeEvent event) {
    for (DetailsViewListener listener : listeners) {
      listener.otherBeneficiaryFieldChanged(event.getText());
    }
  }
});

subPhaseDeathCaseBeneficiaryLayout.addComponents(subPhaseDeathCaseHead
er, beneficiaryOption, otherBeneficiaryTextField);
detailsLayout.addComponents(detailsPhaseHeader,
subPhaseDeathCaseBeneficiaryLayout);

OptionGroup.ValueChangeListener
DeathBeneficiaryOptionValueChangeListener = new
OptionGroup.ValueChangeListener() {
  @Override
  public void valueChange(Property.ValueChangeEvent event) {
    for (DetailsViewListener listener : listeners) {
      listener.deathCaseBeneficiaryChanged((BeneficiaryType)
event.getProperty().getValue());}
    if ((event.getProperty().getValue()).equals(
BeneficiaryType.OTHER_BENEFICIARY)) {
      otherBeneficiaryTextField.setVisible(true);
    } else {
      otherBeneficiaryTextField.setVisible(false);
    }
  }
};
beneficiaryOption.addValueChangeListener(DeathBeneficiaryOptionValueCh
angeListener);

VerticalLayout subPhasePaymentMethodLayout = new VerticalLayout();

PhaseHeader subPhasePaymentMethodHeader = new PhaseHeader("4.2",
"Recurrent payment method");
recurrentPaymentMethodOption.addItems(RecurrentPaymentMethodType.value
s());
recurrentPaymentMethodOption.setNullSelectionAllowed(false);
recurrentPaymentMethodOption.addStyleName("subPart");

final VerticalLayout bankAccountDetailsLayout = new VerticalLayout();
bankAccountDetailsLayout.addStyleName("bankAccountDetails");
Label bankAccountDetailsLabel = new Label("Bank account details");
bankAccountDetailsLabel.addStyleName(ValoTheme.LABEL_BOLD);
bankAccountIBAN.addStyleName("bankAccount");
bankAccountIBAN.setInputPrompt("IBAN...");

bankAccountIBAN.addListener(new FieldEvents.TextChangeListener() {
  @Override
  public void textChange(FieldEvents.TextChangeEvent event) {
    for (DetailsViewListener listener : listeners) {
      listener.bankAccountIBANChanged(event.getText());
    }
  }
});
bankAccountDetailsLayout.addComponents(bankAccountDetailsLabel,
bankAccountIBAN);
OptionGroup.ValueChangeListener
recurrentPaymentOptionValueChangeListener = new
OptionGroup.ValueChangeListener() {
  @Override
  public void valueChange(Property.ValueChangeEvent event) {
    RecurrentPaymentMethodType newRecurrentPaymentOption =
(RecurrentPaymentMethodType) event.getProperty().getValue();
    for (DetailsViewListener listener : listeners) {
   listener.recurrentPaymentMethodChanged(newRecurrentPaymentOption);}
    if
(recurrentPaymentMethodOption.getValue().equals(RecurrentPaymentMethod
Type.EINVOICE)) {
      bankAccountDetailsLayout.setVisible(true);
    } else {
      bankAccountDetailsLayout.setVisible(false);}
  }};
recurrentPaymentMethodOption.addValueChangeListener(recurrentPaymentOp
tionValueChangeListener);
subPhasePaymentMethodLayout.addComponents(subPhasePaymentMethodHeader,
recurrentPaymentMethodOption, bankAccountDetailsLayout);
detailsLayout.addComponents(subPhasePaymentMethodLayout);
```