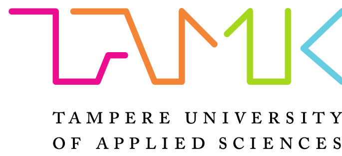


OPENEDGE ABL APPLICATION MODERNIZATION

Case: Technical Dashboard

Olli Havilehto

Bachelor's thesis
February 2017
Business Information Systems
Option of Software Development

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Tietojenkäsittely
Ohjelmistotuotanto

Havilehto, Olli
OpenEdge ABL Application Modernization
Case: Technical Dashboard

Opinnäytetyö 38 sivua, joista liitteitä 1 sivu
Helmikuu 2017

Opinnäytetyön toimeksiantaja on UPM Kymmene IT, tarkemmin IT Services for Raflatac –osasto.

Opinnäytetyön tarkoituksena oli uudistaa vanha Progress OpenEdge -pohjainen työpöytäsovellus palvelukeskeiseen arkkitehtuuriin ja toteuttaa web -käyttöliittymä käyttämään luotuja palveluita.

Tavoitteena oli mahdollistaa vanhan valvontasovelluksen informatiikan hyödyntäminen mistä tahansa pääteohjelmasta joka kykenee kommunikoimaan käyttäen HTTP -protokollaa.

Lopputuote muodostuu AngularJS web –sovelluksesta, OpenEdge ABL Web App REST –rajapinnasta, sekä Node.js –pohjaisesta käänteisestä välityspalvelimesta joka todentaa ja välittää kaiken OpenEdge –rajapintaan kulkevan liikenteen. Vanhan ohjelmiston uudistaminen palvelukeskeiseen arkkitehtuuriin edellytti runsasta perehtymistä ohjelmistojen uudistamisen teoriaan sekä OpenEdge –alustan erityispiirteisiin.

Projekti voi myös toimia astinkivenä Raflatac IT:n sovelluskehityksen päivittämiselle, sekä suurempien ja kriittisempien järjestelmien uudistamiselle.

Asiasanat: Progress OpenEdge Rest SPA AngularJS Node.JS Testaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Software Development

Havilehto, Olli
OpenEdge ABL Application Modernization
Case: Technical Dashboard

Bachelor's thesis 38 pages, appendices 1 page
February 2017

Principal of this thesis is UPM Kymmene IT and precisely IT Services for Raflatac -department.

Purpose of this thesis was to renew old Progress OpenEdge –based desktop application into a service-oriented architecture and to implement a web –based user interface to interact with the new services.

Objective of this thesis was to enable access to old monitoring program data for any application which is able to communicate through HTTP –protocol.

End product consists of AngularJS web application, OpenEdge ABL Web App REST -interface and Node.js reverse proxy solution for authenticating and routing the traffic to the OpenEdge back-end. Renewing the old desktop into service-oriented architecture required a lot of research regarding best practices of software renewal and the traits of OpenEdge platform.

This project may also act as a stepping stone for further modernization of Raflatac IT development and renewal of its larger and more business-critical applications.

Key words: Progress OpenEdge Rest SPA AngularJS Node.JS Testing

CONTENTS

1 INTRODUCTION 6
2 TECHNICAL DASHBOARD AS A CONCEPT 7
2.1 Use cases, users and groups 8
2.2 Mobile Technical Dashboard pilot 9
3 NEW SINGLE PAGE WEB APPLICATION 10
3.1 AngularJS SPA 12
3.2 MVC pattern 14
3.3 Application structure 15
3.4 Unit testing 17
4 RENEWED SERVER ARCHITECTURE 19
4.1 Progress Application Server 20
4.1.1 PAS for development and production 20
4.1.2 PAS authentication 21
4.2 ABL Web Application 24
4.2.1 Application modernization approaches 25
4.2.2 Steps of application reverse- and re-engineering 27
4.2.3 Recognizing the reusable components 27
4.2.4 Component quality assurance 29
4.2.5 Building web services 30
4.3 API testing 32
DISCUSSION 34
REFERENCES 35
APPENDICES 38
Appendix 1. New application architecture 38

GLOSSARY

ABL	OpenEdge Advanced Business Language
ABL Web App	Advanced Business Language web application
AD	Active Directory
API	Application Programming Interface
COTS	Commercial off-the-shelf solution
IDE	Integrated Development Environment
JMX	Java Management Extensions
MTD	Mobile Technical Dashboard
MVC	Model-view-controller
OO	Object-Oriented programming
OpenEdge, OE	Development platform for business applications
OS	Operating System
PAS	Progress Application Server (old Pacific App Server)
PDSOE	Progress Developer Studio for OpenEdge
Progress 4GL	Previous name of OpenEdge ABL
SPA	Single Page Application, modern web application
Technical Dashboard, TD	Tool used to monitor different IT / business systems
Tomcat	Open-source Java Servlet Container
WAR	Web application archive
XHR	XMLHttpRequest

1 INTRODUCTION

The principal of this thesis is UPM Kymmene IT Services for Raflatac, later only Raflatac IT, which is a UPM Kymmene IT department specifically meant to build and run Raflatac Oy IT software and application architecture. Raflatac IT has built major part of its own IT systems for a long time. One of the in-house made applications is Technical Dashboard. The Technical Dashboard project was started in 2009 and the target of the project was to deliver a centralized tool which allows IT personnel to monitor Raflatac's critical global IT and business systems. Technical Dashboard has been built with Progress OpenEdge platform and it has been written with OpenEdge Advanced Business Language programming language.

Current challenges with the Progress OpenEdge based desktop application are but not limit to very slow start time since the application is served to the end user via Citrix session, regular Citrix connection issues, Progress OpenEdge based user interface development which does not support responsive design, monolithic application architecture where data access and business logic is written into the UI components of the application thus maintainability of the source code and further development is hard.

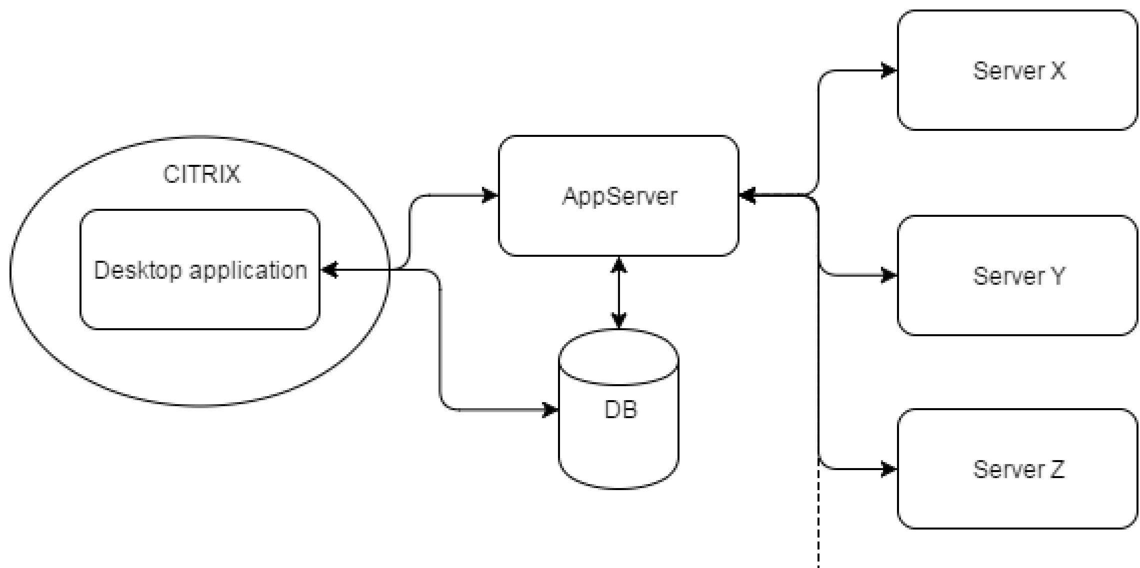
Steps towards RESTful application design and service-oriented architecture were taken during 2015 when RESTful capabilities of Progress OpenEdge platform were first evaluated and old application (Technical Dashboard) parts were served as RESTful services. Project also included implementation of a proof of concept website and a pilot mobile application which consumed these services. This thesis continues in the footsteps of previous project.

Purpose of this thesis was to renew the old Technical Dashboard desktop application into a service-oriented architecture and to implement a web –based user interface to interact with the new services.

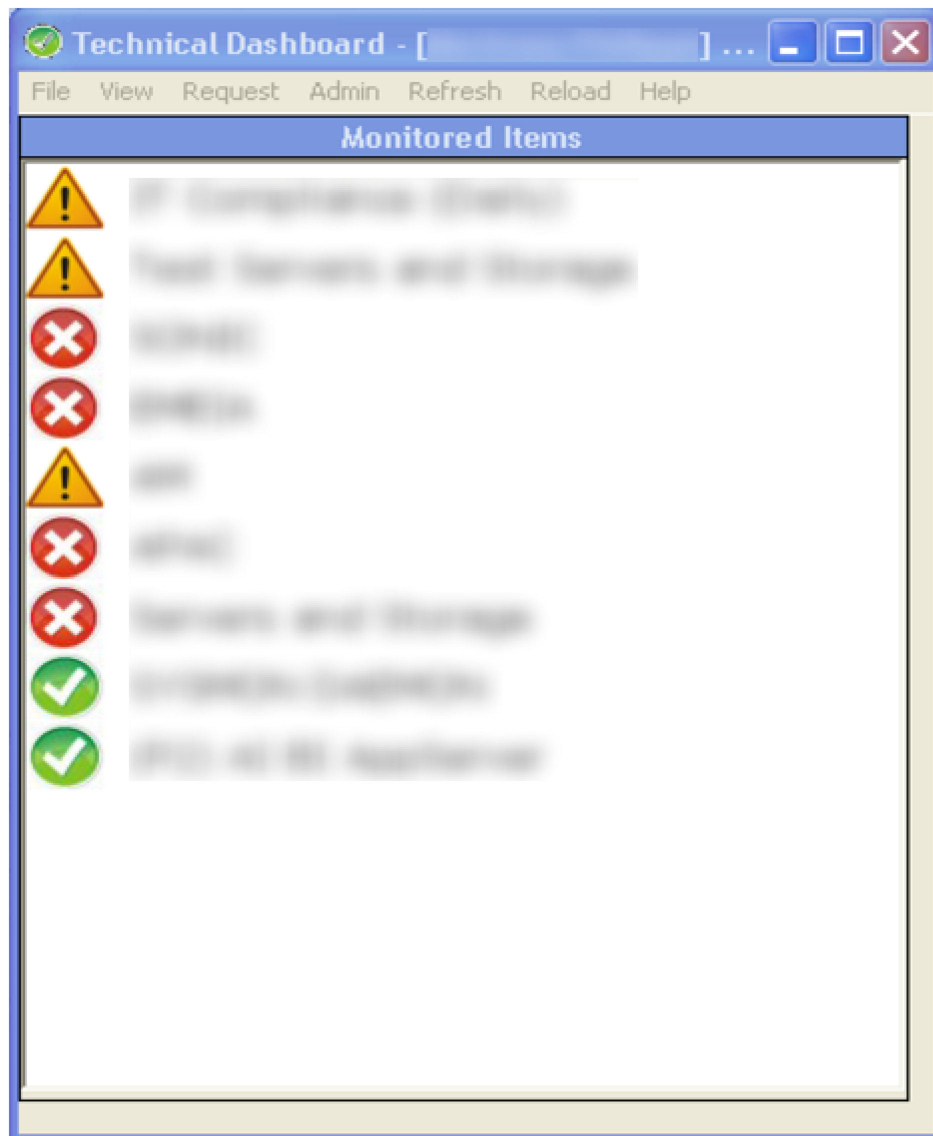
Objective of this thesis was to enable access to Technical Dashboard data for any application which can communicate through HTTP –protocol.

2 TECHNICAL DASHBOARD AS A CONCEPT

Technical Dashboard was created to provide a centralized monitoring tool for Raflatac applications and systems. Prior Technical Dashboard concept consists of a desktop application for clients, server application for invoking direct OS commands and requests to attached systems and to partially update and maintain database, small distributable server application to be deployed on observed servers and a database which holds Technical Dashboard related data. Communication between client and server application was implemented by Progress OpenEdge specific solutions.



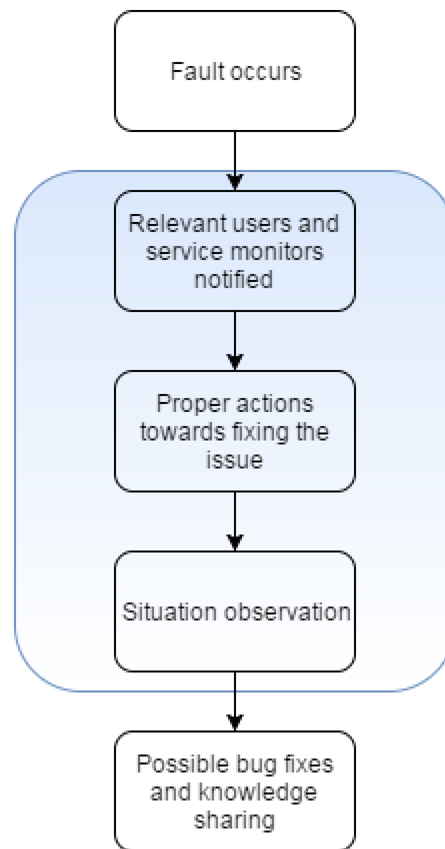
PICTURE 1. Partially monolithic Technical Dashboard architecture



PICTURE 2. Old Technical Dashboard user interface

2.1 Use cases, users and groups

In addition to supervising the different servers and processes Technical Dashboard is also used to configure these systems and related connections, maintain own user register and even to restart halted services. Technical Dashboard supports alerts via SMS, email and the most recent addition: push notifications via Telerik cloud. (mobile application pilot).



PICTURE 3. Example of a workflow in a fault situation. Technical Dashboard may be used to handle steps highlighted in blue

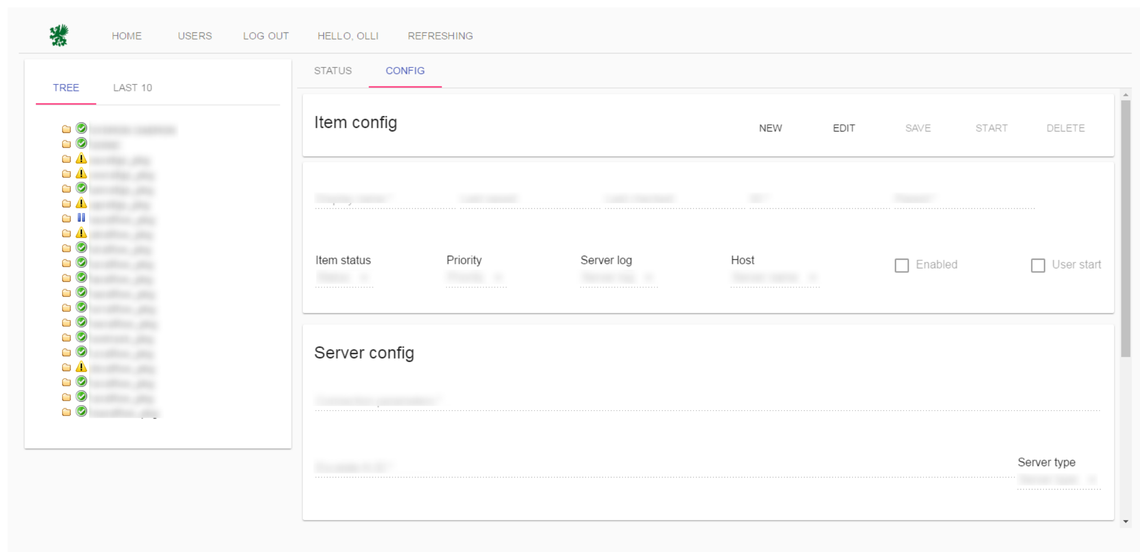
Different key users and service owners from business functions may also use the application to follow the state of crucial processes related to their own work.

2.2 Mobile Technical Dashboard pilot

Project Mobile Technical Dashboard, or "Mobile Dash" was established early 2015 to evaluate mobile client requirements and to pilot Progress Application Server and OpenEdge ABL Web Application capabilities. The project was split into three parts: mobile application for smartphones with a possibility to receive native push notifications of ongoing incidents, web application for observing the situation online with a web browser and backend services for providing the required data to the front-end apps and to handle push notifications in addition to SMS and emails. Web application was originally out of scope and was then implemented as a very brief proof of concept with vanilla JavaScript and jQuery. Authentication of these proof of concept backend services was handled with HTTP Basic authentication. With HTTP Basic authentication username and password are sent within every HTTP request (IETF 1999). Future solution provides more solid approach for validating user credentials and authorizing access to the system.

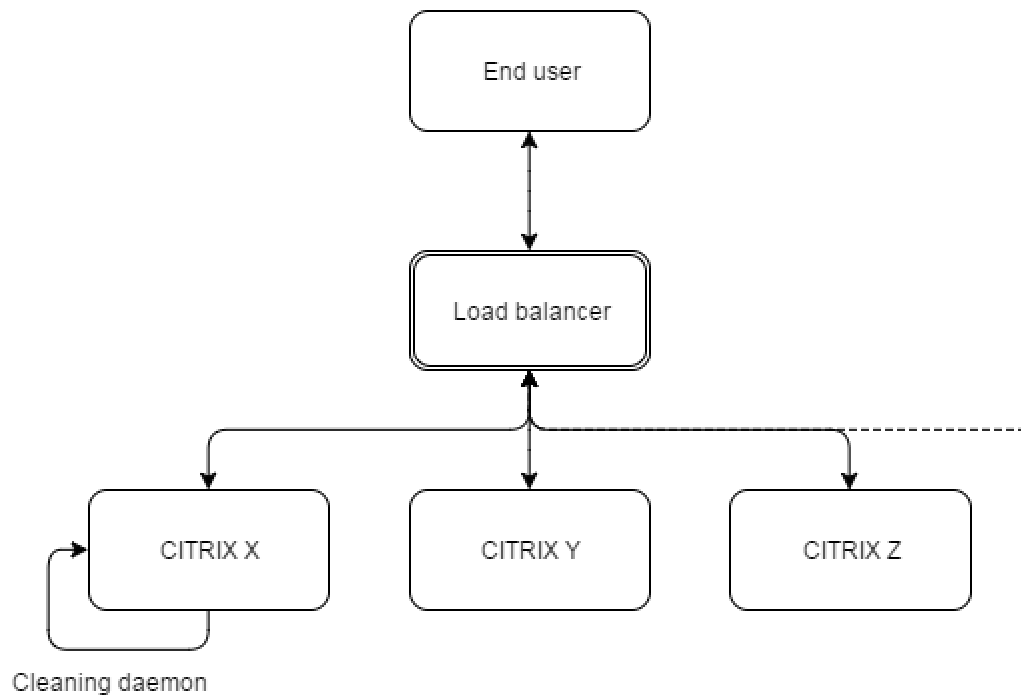
3 NEW SINGLE PAGE WEB APPLICATION

One purpose of the thesis was to create a web based user interface to consume newly built web services and to provide easier access to the application itself. New UI had to represent all good attributes that modern web application has to offer and it had to be built with a sustainable technology that could guarantee support for further future application development. It was also important to study and introduce key features of the selected user interface framework because internal development of web based user interfaces is a rather new concept in Raflatac IT.



PICTURE 4. Technical Dashboard SPA web application

The old desktop application is a Progress based tool and requires an active Progress software installation on the client machine. This requirement has been currently tackled by serving application to the end users with virtual Citrix desktop sessions. Citrix based distribution may cause extra latency, delay and license costs. New web based user interface eases the access to application by removing both requirements for Citrix based distribution and/or active Progress installation by allowing use of application with a regular web browser.

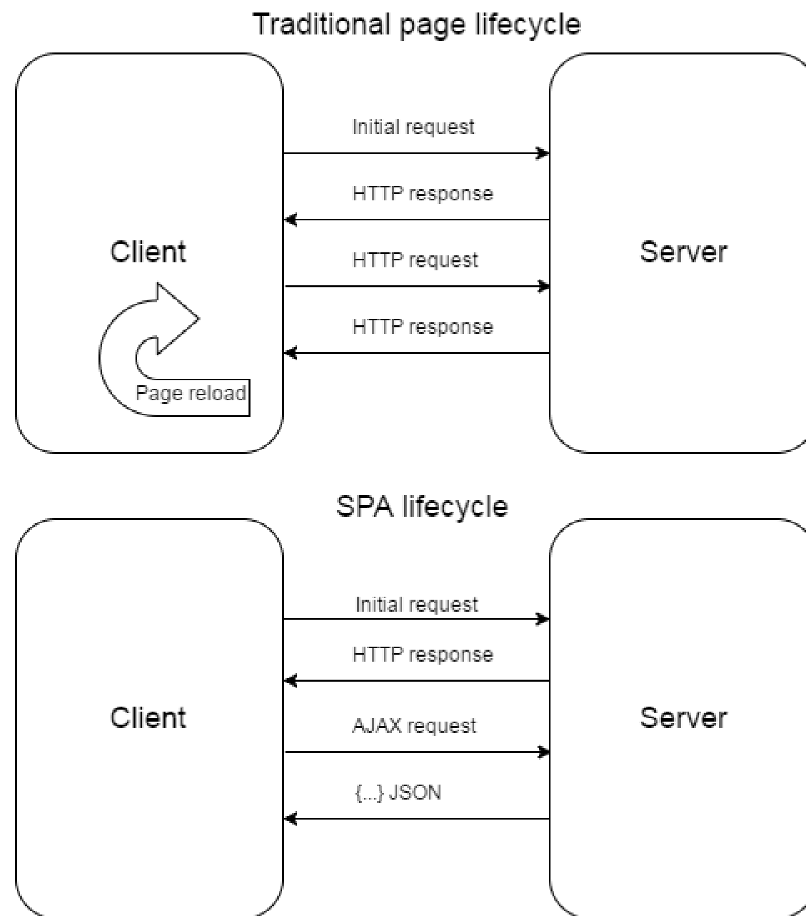


PICTURE 5. Citrix servers serving old desktop application

When making the decision about the future front-end technology stack it was important to evaluate many things: the probability of framework continuum, availability of corporate support if required, cost of future development and possible refactoring, framework complexity if trained for in-house developers et cetera. Though these requirements are not as stiff as with monolithic applications since the service-oriented architecture will always be there even if the user interface layer is renewed.

Variant of modern day web applications are "Single Page Applications" – SPA.

Wasson writes in his MSDN issue that Single-Page Applications (SPAs) are web apps that load a single HTML page and dynamically update that page as the user interacts with the app. SPAs use AJAX and HTML5 to create fluid and responsive web apps, without constant page reloads. However, this means much of the work happens on the client side, in JavaScript. Luckily, there are many open source JavaScript frameworks that make it easier to create SPAs (Wasson 2013).



PICTURE 6. Traditional Page Lifecycle versus the SPA Lifecycle (Wasson, 2013, modified)

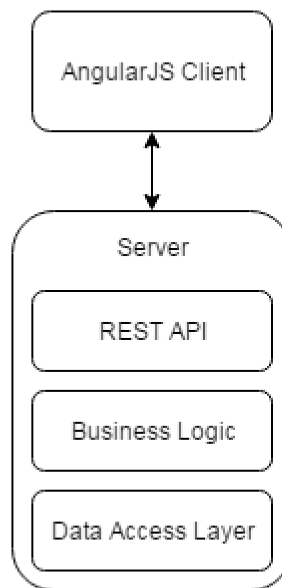
When creating a web application, a developer does not necessarily have to use a SPA framework to achieve the desired outcome, but full and ready framework really eases the application development by providing sophisticated tools from controlling the UI to working with web services and parsing the responses. React, a JavaScript framework for web applications is maintained by Facebook. Facebook has also created Native React framework out of the predecessor: tool for building mobile applications with technologies and the old syntax developers already know. AngularJS is a single page web application framework maintained by Google. AngularJS has also attracted open source mobile frameworks around itself, Ionic as a solid example.

3.1 AngularJS SPA

AngularJS is what HTML would have been, had it been designed for building web-apps (AngularJS.org 2016).

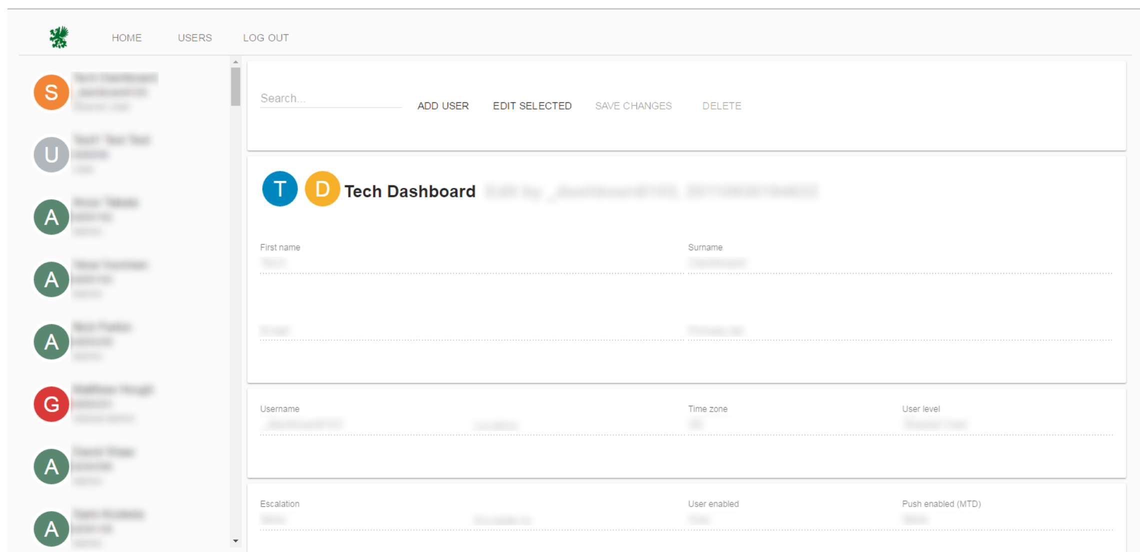
AngularJS is a JavaScript framework for building modern day web- and mobile applications. AngularJS was chosen due to zero license costs, widespread and active developer community, very detailed documentation, and good earlier experiences with the framework. AngularJS is completely open source and free to use. Example alternatives to AngularJS are Backbone.JS and Facebook's React.

By the time of writing Google has released AngularJS 2 to public beta but its predecessor was chosen due to more mature codebase and proven functionality.



PICTURE 7. Example of AngularJS web application as a part of SOA architecture (Shukla 2014)

SPA frameworks can be completely independent of user interface design considering styling and graphic libraries. Google has published Angular Material design CSS –library first introduced on Android devices which implements Google's Material design standard in web applications.

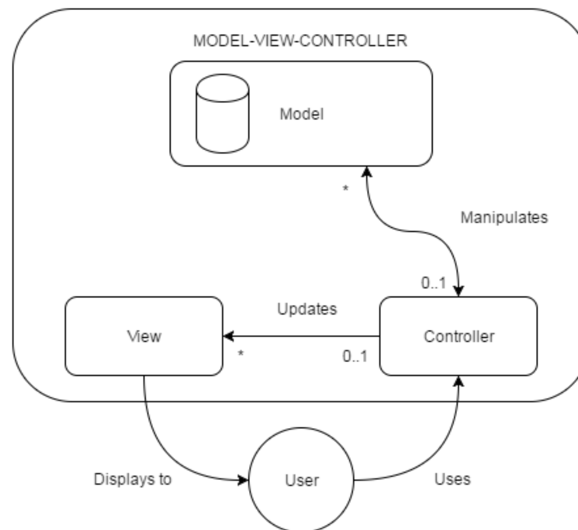


PICTURE 8. Technical Dashboard SPA web apps’ user register implemented with Material design

3.2 MVC pattern

Model-view-controller, MVC, is an application design pattern used to implement user interfaces. During renewal it was important to understand how we should divide Technical Dashboard Model (data and application state), View (web application user interface) and Controller (web application logic) so we can easily test and develop different subareas.

Basic idea of MVC design is that user interface can be easily changed and even transformed to a different graphic framework. User interface will also always display the realtime data in appropriate form (Koskimies, Mikkonen 2005, 142).



PICTURE 9. MVC pattern of AngularJS (AngularJS Tutorial, 2014)

MVC is popular because it isolates the application logic from the user interface layer and supports separation of concerns. The controller receives all requests for the application and then works with the model to prepare any data needed by the view. The view then uses the data prepared by the controller to generate a final presentable response. (Tutorialspoint, 2016.)

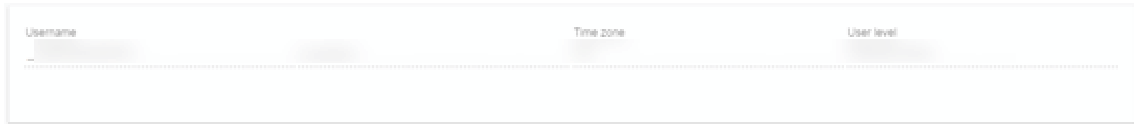
In AngularJS the MVC is implemented with HTML and JavaScript. HTML is used to display View to user whereas JavaScript is used as Controller to update the View and manage Model. Model is the actual application state and active data. In our particular case Model data is updated and fetched from REST interface to our Angular web application.

3.3 Application structure

AngularJS extends basic HTML mark-up with different ng-directives. To validate webpage against W3 standards developer may use data-ng prefix with Angular directives. According to AngularJS documentation following directives are used to assemble Angular functionality to HTML:

- **ng-app** states that AngularJS is running within this element.
- **ng-model** can be used to bind input element value to controller side variable.
- **ng-bind** binds inner HTML of an element to controller side variable.

We can also use ng-directives to control user interfaces. For example, **ng-disabled** can be used to toggle input element state by binding it to Boolean value on controller side.



PICTURE 10. Technical Dashboard web application UI element

```
<md-card>
    <md-card-content>

        <div layout-gt-xs="row">

            <md-input-container class="md-block" flex="50">
                <label>Username</label>
                <input ng-model="selectedUser.UserName" ng-disabled="!editUser">
            </md-input-container>

            <md-input-container class="md-block" flex="50">
                <label>Location</label>
                <input ng-model="selectedUser.LocationCode" ng-disabled="!editUser">
            </md-input-container>

            <md-input-container class="md-block" flex="50">
                <label>Time zone</label>
                <input ng-model="selectedUser.UserTimeZone" ng-disabled="!editUser">
            </md-input-container>

            <md-input-container class="md-block" flex="50">
                <label>User level</label>
                <input ng-model="selectedUser.UserLevel" ng-disabled="!editUser">
            </md-input-container>

        </div>

    </md-card-content>
</md-card>
```

PICTURE 11. Technical Dashboard web application UI element HTML mark-up

Controller side of the application is implemented with JavaScript. AngularJS is very versatile framework and allows easy injection of third party plugins and libraries to the application. With AngularJS we can make asynchronous requests to data sources and update user interface seamlessly. This is called two-way binding, or bidirectional binding. With it the user interface and application data model can be kept in synchronization always.

```
myApp.controller('UsersCtrl', ['$scope', '$http', function ($scope, $http, $state) {

    $scope.users = {};
    $scope.searchUser = "";
    $scope.selectedUser = {};
    $scope.editUser = false;

    $scope.selectUser = function(user) {
        $scope.editUser = false;
        $scope.selectedUser = user;
    };

    $scope.enableEdit = function() {
        $scope.editUser = !$scope.editUser;
    };

    $http({
        url: "http://localhost:8810/rest/TDOOService/users",
        method: "GET"
    }).then(function (response) {

        $scope.users = response.data.response.users.users;
        $scope.selectedUser = $scope.users[0];

    });

}]);
```

PICTURE 12. Technical Dashboard web application Controller code and data fetching and binding

3.4 Unit testing

As features and codebases grow, manual QA becomes more expensive, time consuming, and error prone (Morgan 2016).

Unit testing is an important part of AngularJS application development. It is a method to ensure that single unit of written program logic acts as intended. Unit testing itself is a very universal concept but we inspect it from the Angular point of view.

Unit testing, as the name implies, is about testing individual units of code. Unit tests try to answer questions such as "Did I think about the logic correctly?" or "Does the sort function order the list in the right order? … In order to answer such a question, it is very important that we can isolate the unit of code under test. That is because when we are testing the sort function we don't want to be forced into creating related pieces such as the DOM elements, or making any XHR calls to fetch the data to sort (AngularJS.org 2016).

Good toolset for AngularJS unit testing is Karma + Jasmine. Karma and Jasmine are Node.JS packages and can be installed with NPM package manager. Karma is a command line tool for running the tests whereas Jasmine is behaviour-driven development framework for testing JavaScript code. Both Karma and Jasmine are also standard tools recommend by AngularJS development team (AngularJS.org 2016).

```
describe('getProcesses()', function() {
  // A simple test to verify the method getProcesses exists
  it('should exist', function() {
    expect(getProcesses).toBeDefined();
  });

  // A test to verify that calling getProcesses() returns an array of objects
  it('should return an array of objects if it exists', function() {
    expect(getProcesses()).toEqual(arrayOfObjects);
  });
});
```

PICTURE 13. Technical Dashboard Karma and Jasmine testing action.

With this toolset we can quickly define tests for application functions and behaviour. It also forces us to develop good, testable code. Karma based tests can also be automatically executed with continuous integration tools such as Jenkins and Travis.

```
04 11 2016 10:11:44.951:INFO [watcher]: Changed file "C:/Users/k400872/WebstormProjects/TD2/app/scripts/spec.js".
Chrome 52.0.2743 (Windows 7 0.0.0): Executed 1 of 1 SUCCESS (0.018 secs / 0.001 secs)
```

PICTURE 14. Successfully executed unit test

4 RENEWED SERVER ARCHITECTURE

Back-end application platform is OpenEdge by Progress Software Company. OpenEdge has been *de facto* development platform in Raflatac IT for a long time and its programming language OpenEdge ABL the main programming language.

OpenEdge Advanced Business Language (formerly known as Progress 4GL) is a programming language introduced by Progress company in the early 1980's. OpenEdge ABL is a fourth generation programming language (hence the original name). Fourth generation programming languages were supposed to take application development to higher abstraction layer and use a more developer friendly syntax compared to third generation (C, C++, C#, Java...) languages (Wikipedia 2016).

PICTURE 15. Progress OpenEdge

As the predecessor application is a monolithic "fat-client" program major part of this renewal project was reforming old monolithic application business logic as service-oriented architecture and web services so the old logic is accessible by any application which supports HTTP protocol, in this case AngularJS SPA web application. Service requests are authorized per user level.

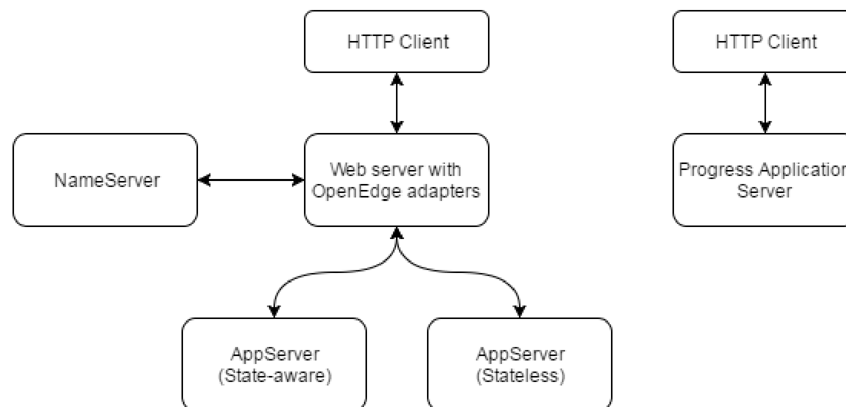
Refactoring was very teaching and provided plenty of learnings. Harsu writes in her book that such a project should be carried out solely without other distractions to avoid prolonged software projects where application is already outdated when it reaches the end users. Developers should also be careful when defining modernization project scope. Introducing too many new features while reshaping every old one can cause a great workload which may also lead to unexpected results in application behaviour (Harsu, 2003, 174).

4.1 Progress Application Server

Bridge the gap between yesterday's mission-critical applications and the innovative technology needs of today (Progress OpenEdge 2016).

Progress application server, prior known as Pacific Application Server, acts a very important role when we are considering application renewal to service-oriented architecture, since it introduces the out-of-the-box support for RESTful web services. It runs the same ABL source code than its predecessor, Classic Application Server, but removes the requirement of external web server and OpenEdge REST adapters.

Progress application server takes also care of handling traditional Progress specific things like NameServer.



PICTURE 16. Progress Application Server in comparison with old HTTP enabled architecture (Progress OpenEdge 2015)

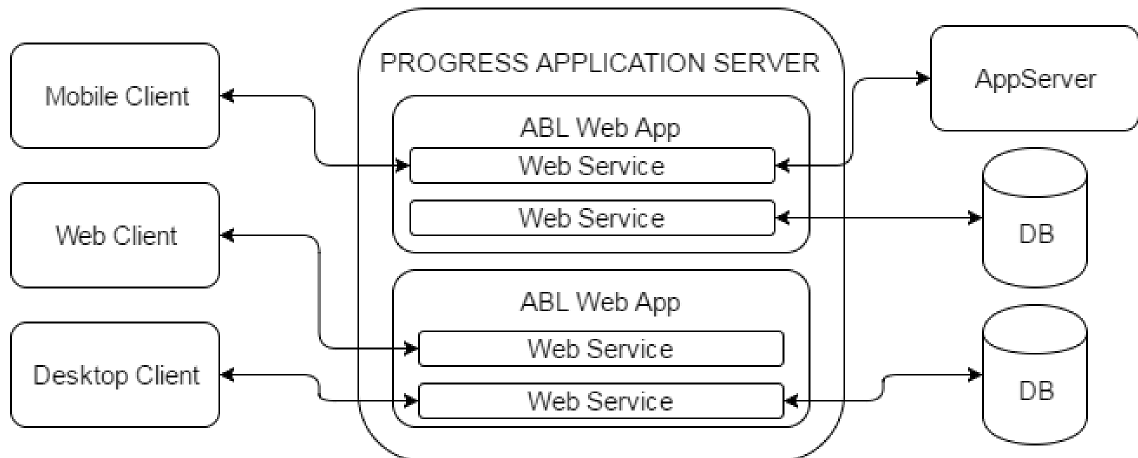
4.1.1 PAS for development and production

Progress Application Server comes in with two different versions. We can tell from the naming convention that development version should be used for developing applications and vice versa. Mostly the differences consider increased security through disabled features. Some key differences of production version compared to development are:

- Default Tomcat remote administration applications removed
- Replaced ROOT application that specifically supports Progress apps
- Automatic app deployment from development tools disabled

- Shutdown port disabled for UNIX
- JMX remote access disabled
- Web crawler filtering disabled

Development features of a Production Progress Application Server can be enabled if required (Progress 2015).

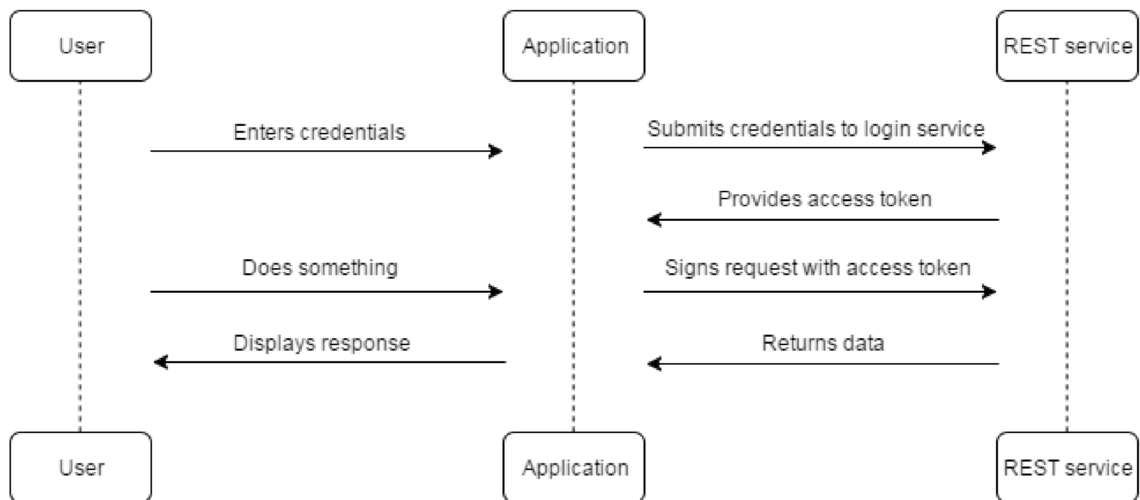


PICTURE 17. Progress Application Server serving ABL Web applications

4.1.2 PAS authentication

Authenticating RESTful web services and modern web applications differs from “traditional” web applications and session based solutions. Since RESTful web services are stateless and no session is stored on server side so should the authentication be.

Token based authentication is a common stateless solution. Variant of token based authentication is a JSON Web Token, JWT. JWT is an IETF industry standard RFC 7519. When compared to another stateless authentication method, HTTP BASIC authentication, JWT tokens do not include user’s password but can contain much more data.

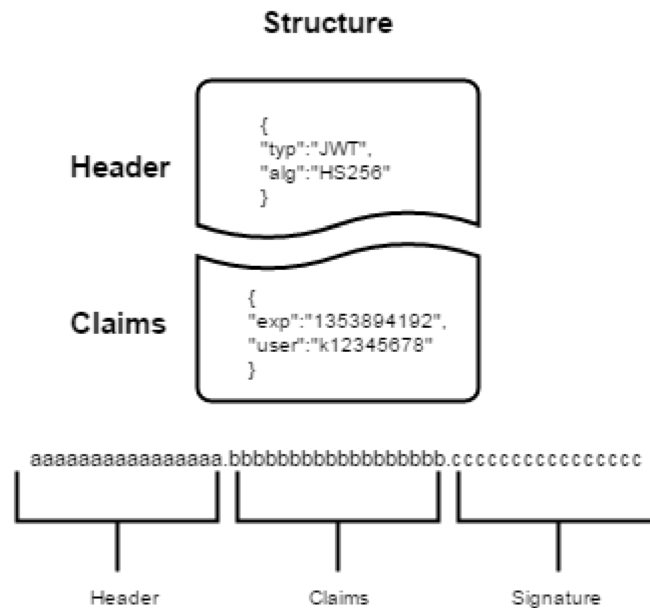


PICTURE 18. RESTful web application stateless authentication sequence

During generation JWT tokens are encoded with agreed algorithm (HS256, RS256...) and a salt ("passphrase") and are sent back to client. JWT token is stored on user's device and should be sent back to RESTful web service within every request where it is again decoded and validated. This way the RESTful web service can be sure that the user really is who he or she claims to be. JWT tokens can for example be placed in a HTTP header "x-access-token".

It should be noted that JWT tokens can be set to last forever or for a desired amount of time. With stored token user does not have to "log in" to acquire new token every time he or she needs to access RESTful web services with client application. Generated tokens can also be held in a database where they can be marked as invalidated if necessary. Such a case could be where an employee is leaving the company before his or her web tokens expire et cetera.

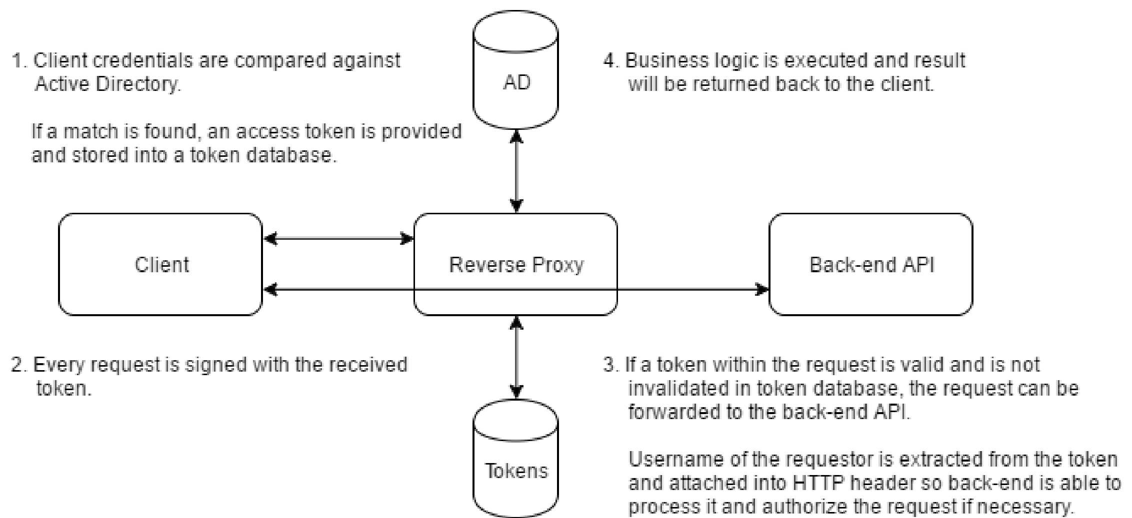
Secure HTTPS connections should always be used with JWT tokens to efficiently prevent man-in-the-middle and replay attacks.



PICTURE 19. Example of an encoded and decoded JSON Web Tokens (NoteJS 2014, modified)

Considering OpenEdge platform good news are that these great stateless authentication methods exist. Bad news is that they are not yet supported by Progress OpenEdge. Progress OpenEdge platform currently supports only HTTP BASIC authentication as a stateless authentication method (Progress Knowledge Base, 2016).

Since we wanted to avoid logging in on every application launch and we wanted to support token based authentication, a reverse proxy solution with Microsoft Active Directory integration and JWT token creation and validation was implemented. Reverse proxy only takes care of user authentication and routing of the requests to the actual REST interface. This is invisible to the actual end-user of the application. Username of the requestor is extracted from the token of a valid request and attached to the request headers. User action authorization is still left to Technical Dashboard API itself. Technical Dashboard API accepts requests only through this reverse proxy.

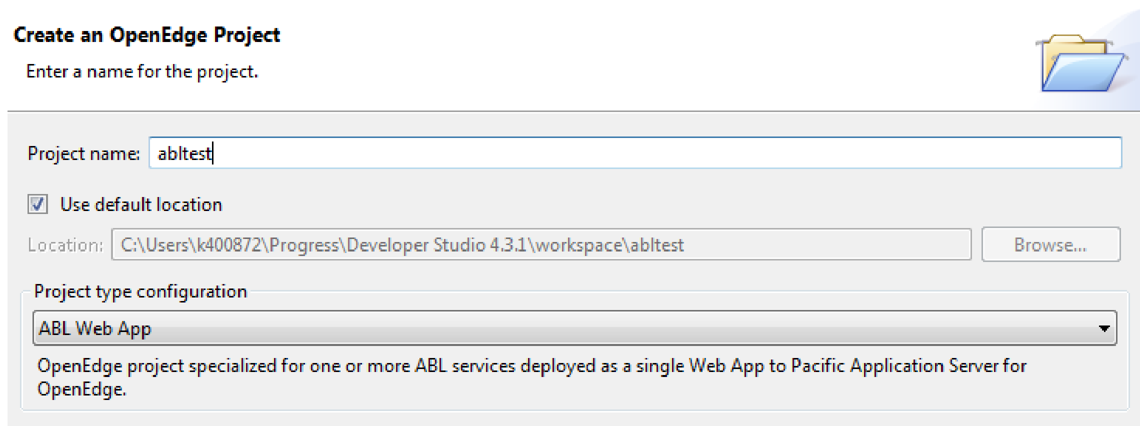


PICTURE 20. Reverse proxy protecting Technical Dashboard API

4.2 ABL Web Application

Closer inspection to old desktop application did show us that a great part of business logic was built into old user interface and existing documentation was close to none. Simple application wrapping with new data access layer was not possible. Application had to be reverse- and re-engineered into service-oriented architecture. Result and end product is an ABL Web Application which serves old and partly improved source code as a RESTful web services.

The ABL Web App project type is an OpenEdge project that lets you deploy one or more ABL services as a single web app to PAS for OpenEdge. It contains the ABL Web App, ABL AppServer, Progress Adapters, and OpenEdge facets. It contains ABL WebSpeed (for ABL Service of WebSpeed type), ABL REST (for ABL Service of REST type), or ABL Data Object (for ABL Service of Data Object type) facets depending on the ABL Service type that you created while creating the project (Progress 2016).



PICTURE 21. Creation of ABL Web Application project

4.2.1 Application modernization approaches

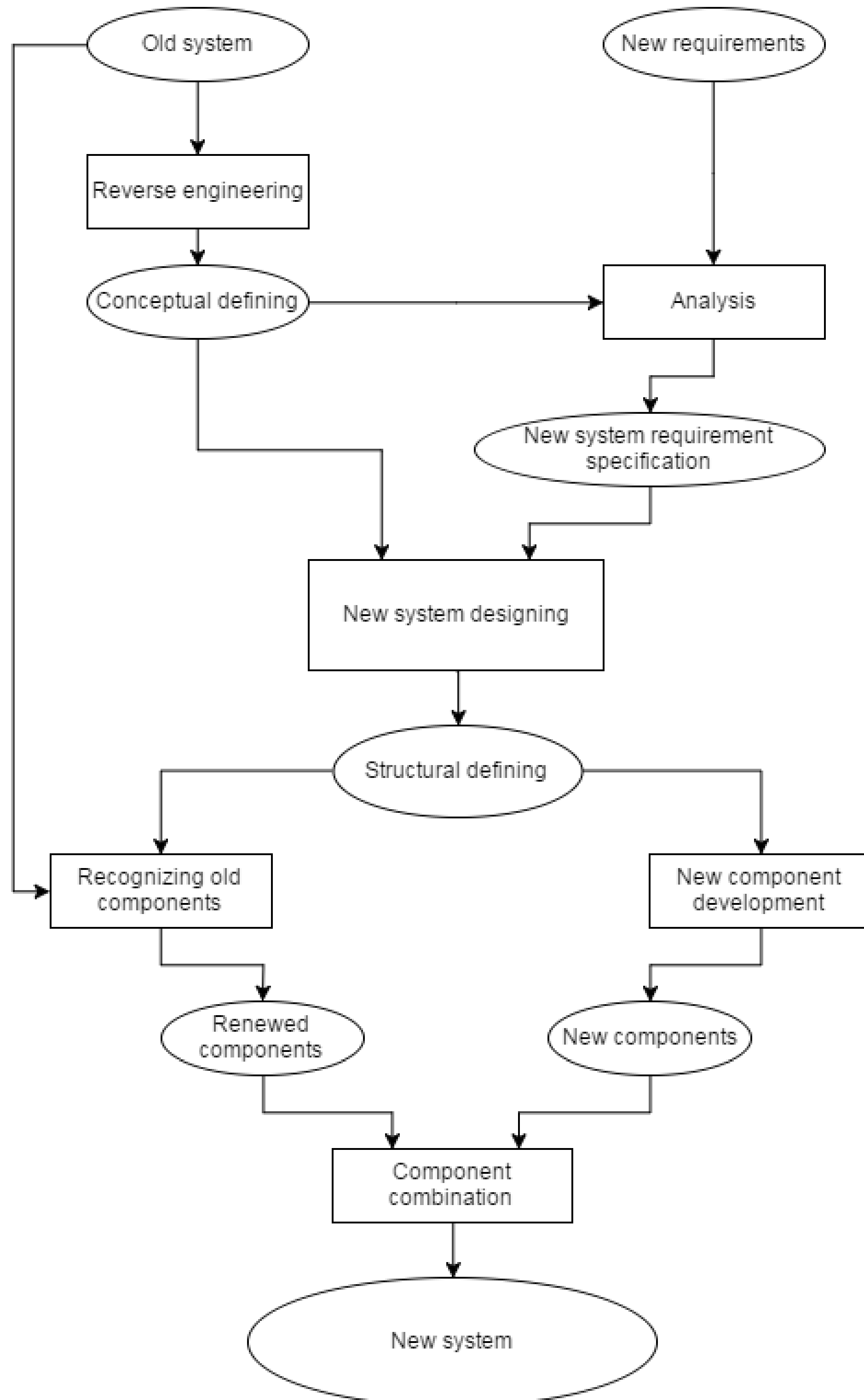
Need for application modernization can occur for many reasons: old systems start to become unreliable due to tremendous code base, monolithic pattern applications do not scale effectively enough, new need for supporting mobile clients has risen, old frameworks might not meet modern day requirements, support for old tools is deprecated, upkeep of old application becomes too expensive or consuming et cetera. In this particular case we had to renew old application in order to serve old functionality as a services to be able to separate the user interface from the application itself.

Williamson and Laszewski write in their book Oracle Modernization Solutions that there are five primary approaches to legacy modernization:

- Application re-architecting to a new environment
- SOA integration and enablement
- Re-platforming through re-hosting and automated migration
- Replacement with COTS solutions and data modernization
- Data modernization

These options may be implemented solely or multiple together. They are not mutually exclusive or necessarily have interdependence (Williamson & Laszewski 2008, 14).

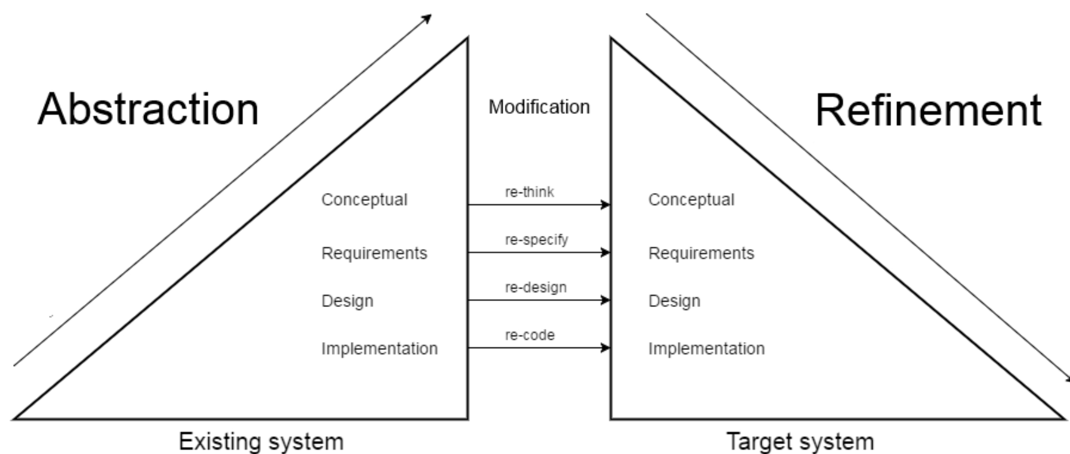
In this thesis we focused on SOA integration and enablement and Re-Architecting of Technical Dashboard. We also took a look at the best practices and available methodologies which help developers to understand application behaviour and to recognize abstract application functionality for reusable components.



PICTURE 22. System renewal steps (Harsu, 2003, 181)

4.2.2 Steps of application reverse- and re-engineering

Technical Dashboard required rather large scale overhauling while converting old functions to web services. In order to create required services we had first to understand application workflow, what does it do, why does it do it and how does it do it. By recognizing higher abstractions from the application workflow we were able to dig deep into source code itself.



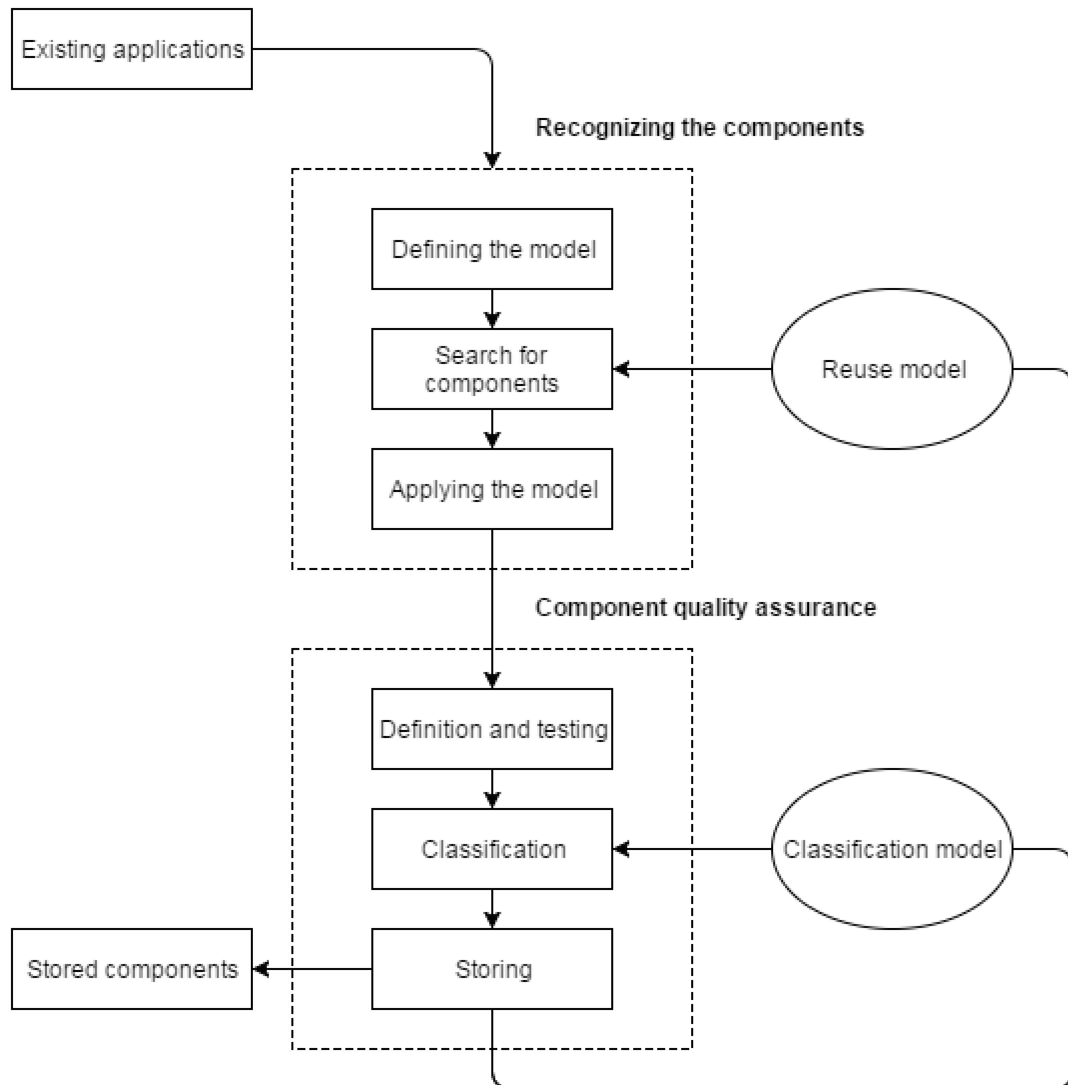
PICTURE 23. General software renewal process from reverse engineering to rebuilding (Harsu 2003, 26, modified)

4.2.3 Recognizing the reusable components

Part of application renewal and reverse engineering was recognizing reusable components from old software. For example, one good recycled component of Technical Dashboard was “CreateRequest” –procedure which is used to invoke different operating system commands on observed servers. Recognizing possible objects (OO-pattern) within procedural program is also a way to reuse old components (Harsu 2003, 255).

Harsu writes in her book that reusable components may or may not be dependent on programmatic scope or be application specific. Independent components can be examined straight from source code. Example of such a components are highly similar or identical code blocks which have been copied and pasted to different places. This kind of snippets or fragments should be reformatted into a subprogram (procedural paradigm) or possibly

into a class (object-oriented programming). Application specific solutions can be examined from documentation. These solutions may give a hint about application specific reusable components (Harsu 2003, 255-256).



PICTURE 24. Component recognition and quality assurance (Harsu 2003, 256)

Defining the model for reusable components consists of listing the different kind of properties and features that reusable components should have. These properties may be refined based on found components. Found components which fulfil the defined model are then accepted for the next step, component quality assurance (Harsu 2003, 256-257).

From Technical Dashboard we were mainly looking for complex functions that either have high cohesion with each other or to common libraries and would be hard and costly to re-write again.

4.2.4 Component quality assurance

Harsu also writes in her book that not every application component is reusable and developers can set requirements for recycled components. These requirements can be set for new, old and renewed components. Some of example requirements:

- Components must be independent of environment
    - Reused components must be able to run regardless of the environment. This means that component must work outside of the origin application.
- High cohesion
    - Components must implement one or multiple functions which have a high cohesion with each other. This will make describing the component functionality easier.
- Loose coupling
    - Components must have little to none connections to other components. High cohesion between components will decrease the reusability.
- Adaptability
    - Reused components must be adaptable in a way that they can be used in many similar situations.
- Understandability
    - Reused components must be easily understandable in a way that developers are able to quickly interpret the functionality.
- Reliability
    - Reused components must be reliable. Components must be able to function under unexpected circumstances.

(Harsu 2003, 257-258).

```
/* this CreateProcess function is a simplified version of the
   CreateProcess API definition.
   Parameters:
   1. Commandline, for example "notepad.exe c:\config.sys"
   2. CurrentDir,  is default directory for new process
   3. wShowWindow, 0=hidden, 1=normal, 2=minimized, 3=maximized
   r. return       if <>0 then handle of new process
                   if  =0 then failed, check GetLastError  */

FUNCTION CreateProcess RETURNS INTEGER
         (input CommandLine as CHAR,
          input CurrentDir  as CHAR,
          input wShowWindow as INTEGER) :

   def var lpStartupInfo as memptr.
   set-size(lpStartupInfo)     = 68.
   put-long(lpStartupInfo,1)   = 68.
   put-long (lpStartupInfo,45) = 1. /* = STARTF_USESHOWWINDOW */
   put-short(lpStartupInfo,49) = wShowWindow.

   def var lpProcessInformation as memptr.
   set-size(lpProcessInformation)   = 16.
```

PICTURE 25. Example of a complex component from Technical Dashboard hich might not fulfil the requirements

Requirements may not be limited to these. Other requirements could concern up-to-date and up-to-standard documentation. Reused component cannot be too complicated and user feedback has to be taken in consideration. Chosen components should be carefully inspected and tested. Component functionality will be defined based on the inspected source code and available documentation. Component functionality should be carefully tested against the definition. Component must be discarded if it is not possible to define the functionality or if the component functions in an unexpected way. (Harsu 2003, 258.)

During renewal it was possible to recycle numerous complex Technical Dashboard components and libraries. Suitable components were turned into ABL classes and old libraries were kept as found and were invoked with dynamic-function calls from within new classes.

4.2.5 Building web services

After reverse-engineering the application and therefore understanding its workflow and recognizing, testing and storing its important components we could use them to rebuild the services

Some of the extracted components required honing. For example, user authorization and advanced error handling had to be added to methods. Some features were so simple that it was easier just to implement new method to fetch and serve data from database than

to recycle old component. Opposite cases were where old functions had high cohesion to other functions and libraries so it was easier to extract and recycle common libraries in addition old function.

```
@openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
METHOD PUBLIC VOID getProcesses(OUTPUT TABLE processes):

    EMPTY TEMP-TABLE processes.

    FOR EACH CONFIG NO-LOCK:

        CREATE processes.
        BUFFER-COPY CONFIG TO processes.

    END.

END METHOD.
```

PICTURE 26, Simple RESTful web service written in ABL

API and access point design is a broad topic. Vinay Shani (2015.) has written very detailed article about it. The key principles of REST involve separating your API into logical resources. These resources are manipulated using HTTP requests where the method (GET, POST, PUT, PATCH, DELETE) has specific meaning.

But what can I make a resource? Well, these should be nouns (not verbs!) that make sense from the perspective of the API consumer. Although your internal models may map neatly to resources, it isn't necessarily a one-to-one mapping. The key here is to not leak irrelevant implementation details out to your API!
(Shani, 2015.)

Once you have your resources defined, you need to identify what actions apply to them and how those would map to your API. RESTful principles provide strategies to handle CRUD actions using HTTP methods mapped as follows:

- GET /tickets - Retrieves a list of tickets
- GET /tickets/12 - Retrieves a specific ticket
- POST /tickets - Creates a new ticket
- PUT /tickets/12 - Updates ticket #12
- PATCH /tickets/12 - Partially updates ticket #12
- DELETE /tickets/12 - Deletes ticket #12

(Shani 2015).

We used pragmatic approach while designing Technical Dashboard access points, nouns were preferred over verbs during naming, services were set to return data in universal JSON format and the API versioning is written straight into the resource URL.

Method	URL	Description
Get	/v1/processes	Get status of every process.
Get	/v1/processes/{processId}	Get status of a particular process.

PICTURE 27. Example of Technical Dashboard back-end resources

4.3 API testing

RESTful web services are great testing-wise. In this section we talk about “black-box” RESTful API testing. White-box and application unit testing is whole another topic. While testing Technical Dashboard API we used JavaScript frameworks called Chai and Mocha which both run on Node.JS. Both can be installed with NPM package manager.

Mocha has a support for simple asynchronous and promise –based testing and hooks that can be applied to code which will fire at a desired moment. With Chai we can utilize API tests that have been previously done by a developer with a REST client and automatically analyze the responses.

```
describe('/GET processes', () => {
    it('it should GET all the processes', (done) => {
        chai.request(server)
            .get('/processes/20')
            .set("Content-Type", "application/json")
            .end((err, res) => {

                expect(res.status).to.equal(200);
                expect(res.body.response.success).to.equal("true");
                res.body.response.singleProcess.should.not.have.length(0);
                done();

            });
    });
});
```

PICTURE 28. Technical Dashboard API testing

These tests can be combined, stored and executed every time something has changed either in API configuration or in database schema. This is called regression testing.

```
> mocha --timeout 10000



  /GET processes
    V it should GET all the processes (55ms)


  1 passing (69ms)
```

PICTURE 29. Successfully executed API test

This kind of automatic testing has not been possible in the past since communication between clients and servers has been mostly handled with Progress specific technologies or such a testing methodologies have not been studied at all.

```
/GET processes
  1) it should GET all the processes


0 passing (81ms)
1 failing

1) /GET processes it should GET all the processes:

    Uncaught AssertionError: expected 'true' to equal 'false'
    + expected - actual

    -true
    +false
```

PICTURE 30. API test which has caught an error

Automatic API testing can really lift off weight from the shoulders of the application testers.

DISCUSSION

Writing of this thesis has been extremely rewarding and teaching. Even though I had at least some earlier experiences from every technical aspect of the final end-product I can fair and square admit that past months have been very educational. The scope of the project, consisting of a full-stack solution and authentication design and implementation in addition to writing test cases with modern tools, was admittedly quite wide for a bachelor’s thesis and I might have not fully understood what I signed up for in the beginning. Though at the time of writing this I can be happy and pleased about the technical outcome of the project.

Project results do not only consist of concrete software and tools but also of a higher understanding of application renewal, different modernization approaches and their requirements and of a RESTful service development within our technology stack. Also for example finding out that Progress OpenEdge platform does not currently support any kind of stateless token based authentication was very good regarding future development and larger projects where schedules are more tight and development delay could be crucial.

Business and money-wise we can tell from this project that web based user interfaces could save a pretty penny from license costs. Removing the requirement of Citrix and/or Progress client side installations in addition to more flexible development environments and tools is a big benefit which should be calculated carefully.

Future steps for Technical Dashboard concept are a deployment to run environment and observation and learning possible fault situations and bottle necks before killing the old desktop application completely. Node.JS based authentication solution should also be investigated and possibly developed further since need for similar service is only growing within the company.

I’d like to thank UPM Kymmene IT for providing me this chance and my fellow colleagues and superiors for valuable support. Especially with our beloved Progress OpenEdge platform.

REFERENCES

AngularJS Tutorial 2014. AngularJS Basics Part 1: The Model-View-Controller approach. [online] [referred 7.11.2016] http://www.angularjstutorial.com/2014/02/12/angularjs-basics-part-1-the-model-view-controller-approach/

AngularJS Documentation 2016. Unit Testing [online] [referred 7.11.2016] https://docs.angularjs.org/guide/unit-testing

Donald Bren School of Information and Computer Sciences 2000. Representational State Transfer (REST). [online] [referred 7.9.2016] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_1_3

Harsu, M. 2003. Ohjelmien ylläpito ja uudistaminen. Jyväskylä: Talentum.

HowToDoInJava – IN-DEPTH CONCEPTS AND BEST PRACTICES 2016. Microservices – Definition, Principles and Benefits. [online] [referred 9.9.2016] www.howtodoinjava.com/design-patterns/microservices-definition-principles-benefits/

Javarevisited 2015. Difference between SOAP and RESTful Web Service in Java. [online] [referred 6.9.2016] www.javarevisited.blogspot.fi/2015/08/difference-between-soap-and-restfull-webservice-java.html

Kurtz, J., Wortman, B. 2014. ASP.NET Web API 2: Building a REST Service from Start to Finish. New York: Apress.

Microsoft 2016. Three-tier Application Model. [online] [referred 5.9.2016] www.msdn.microsoft.com/en-us/library/aa480455.aspx

Microsoft 2016. SOA in the Real World. [online] [referred 5.9.2016] https://msdn.microsoft.com/en-us/library/bb833022.aspx

Morgan, A. 2016 Testing AngularJS with Jasmine and Karma (Part 1) [online] [referred 20.10.2016] https://scotch.io/tutorials/testing-angularjs-with-jasmine-and-karma-part-1

Nginx 2016. Refactoring a Monolith to Microservices. [online] [referred 12.9.2016] https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/

NoteJS 2014. Working with JSON Web Tokens (JWT) with JavaScript only? [online] [referred 3.10.2016] http://notejs.com/?p=274

Progress 2015. Progress Application Server production server customizations [online] [referred 2.11.2016] http://documentation.progress.com/output/ua/PAS/index.html#page/pas/progress-application-server-production-server-cu.html

Progress Knowledge Base 2016. WHAT ARE THE STATELESS AUTHENTICATION MECHANISMS AVAILABLE WITH THE PACIFIC APPSERVER? [online] [referred 2.11.2016] http://knowledgebase.progress.com/articles/Article/What-are-the-stateless-authentication-mechanisms-available-with-the-Pacific-AppServer?q=token+authentication&l=en_US&fs=Search&pn=1

Progress OpenEdge 2015. Comparing the architecture of the OpenEdge AppServer and PAS for OpenEdge [online] [referred 4.10.2016] https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/pasoe-intro/comparing-the-architecture-of-the-openedge-appse.html

Progress Software Corporation 2015. REST Web service architecture in OpenEdge. [online] [referred 7.9.2016] www.documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dvwsv/rest-web-service-architecture-in-openedge.html
Progress Software Corporation 2016. OpenEdge platform. [online] [referred 5.9.2016] www.progress.com/openedge/platform

Rossberg, J., Ehn, J. & Olausson, M. 2014. Beginning Application Lifecycle Management. New York: Apress.

Sahni, V. 2015. Best Practices for Designing a Pragmatic RESTful API [online] [referred 20.10.2016] http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#restful

Shukla, A. 2014. A basic SPA application using AngularJS, Web API and Entity Framework [online] [referred 20.10.2016] http://www.codeproject.com/Articles/737030/A-basic-SPA-application-using-AngularJS-WebAPI-and

The Internet Engineering Task Force 2015. JSON Web Token (JWT). [online] [referred 9.9.2016] www.tools.ietf.org/html/rfc7519

The Internet Engineering Task Force 1999. HTTP Authentication: Basic and Digest Access Authentication. [online] [referred 20.9.2016] https://www.ietf.org/rfc/rfc2617.txt

Tutorialspoint 2016. AngularJS - MVC Architecture. [online] [referred 20.10.2016] https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

Wasson, M. 2016. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. [online] [referred 9.9.2016] https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

Wikipedia 2016. OpenEdge Advanced Business Language [online] [referred 20.9.2016] https://en.wikipedia.org/wiki/OpenEdge_Advanced_Business_Language

Williamson, J., Laszewski, T. 2008. Oracle Modernization Solutions. Birmingham: Packt Publishing.

APPENDICES

Appendix 1. New application architecture