|  |
| --- |
| CSI Alpha Las  &  Windesheim University of Applied Sciences |
| Technical Reference |
| Technical Reference for the Percurrentis Project |
|  |
| **Thomas Baart (s1054659) & Tim Lagerburg (s1045334)** |
| **February 2014** |

|  |  |  |  |
| --- | --- | --- | --- |
| Signatures | | | |
| **Roel van Rijswijck**  Work-Study Supervisor | **Marco Krop**  School Supervisor | **Thomas Baart**  Project member  S1054659 | **Tim Lagerburg**  Project Member  S1045334 |

|  |
| --- |
| During the Percurrentis Project, two students of the Windesheim University of Applied Sciences have worked on automating several tasks within CSi AlphaLas’ internal travel department. Many choices have been made during the project, which have affected its outcome. This report documents those choices and gives guidance for maintainers of the project. |

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Notes |
| 0.1 | 18/02/2014 | Thomas Baart | First draft of chapters “Technical Requirements”, “Development Platform” and “Using a JavaScript Framework” |
| 0.2 | 25/02/2014 | Thomas Baart | Added chapter “Using an HTML Template” |
| 0.3 | 12/06/2014 | Thomas Baart, Tim Lagerburg | Added various chapters |
| 1.0 | 13/06/2014 | Thomas Baart | Finalised first version |

# Document Accountability

This chapter shortly denotes which project member contributed to which chapter. All chapters that were authored by one of the project members were reviewed by the other.

The styling of the document was done by Thomas.

|  |  |
| --- | --- |
| Chapter | Author |
| Introduction | Thomas |
| Technical Requirements | Thomas |
| Development Platform | Thomas |
| Using a JavaScript Framework | Thomas |
| Using an HTML Template | Thomas |
| Package Diagram | Thomas |
| Form Wizard | Thomas |
| Travel Request Approval | Thomas |
| Data persistence | Tim Lagerburg |
| Database Modeling | Tim Lagerburg |
| Client-Server technology | Tim Lagerburg |

# Table of Contents

Document History 2

Document Accountability 3

Table of Contents 4

Introduction 5

Intended Audiences 5

Scope of the Document **Error! Bookmark not defined.**

How to Read This Document **Error! Bookmark not defined.**

Technical Requirements 6

Development Platform 7

Framework Comparison 7

Differences between SharePoint Editions 9

SharePoint Licensing Types 10

Conclusion 10

Using a JavaScript Framework 11

Choice Comparison 11

Conclusion 12

Using an HTML Template 13

Versions **Error! Bookmark not defined.**

Package Diagram 15

Form Wizard 16

Travel Request Approval 17

Server Technologies 19

Data persistence 19

Database Modeling 19

Client-Server technology 19

# Introduction

This report documents the technical decisions made during the project. It also documents its implementation to an extent.

## Intended Audiences

The intended audiences of this document are the **maintainers** and **developers** of the system that was developed during project Percurrentis.

# Technical Requirements

There were a number of technical requirements for the system that were taken into account during the implementation of the system. Those requirements are the following:

* **Authentication**. Microsoft Active Directory should be used as the authentication provider. Employees of CSi already log in to their systems using Active Directory, and implementing authentication using Active Directory within the system would mean that employees would be automatically logged into the system. Password handling, then, would also not be a concern to deal with within the system.
* **User Roles and Permissions**. There are various user roles within the system. These need to be provided using the functions of employees, which are available through Active Directory. Permissions for certain functionality within the system need to be provided based on the user role; not everyone can execute the same tasks.
* **Complex Entity Relationships**. It should be possible to have complex relationships between data model entities. Please refer to the Domain Model for more information.
* **Forms**. Much of the data needs to be entered in forms. Users of the system spend most of their time filling these forms. Because of this, forms should meet the following requirements:
  + **Keyboard Navigation**. Within most forms, it is possible to use the TAB key to move from one field to another. This should be possible.
  + **Suggestions / Auto Completion**. Many entities have relationships. They can be leveraged for form completion. Forms need to be able to display suggestions for certain form fields.
* **Document Generation**. It should be possible to generate documents based on data within the system.
* **Notifications**. The system should be able to send notifications, by mail for example, based on events within the system.
* **Web Deployment**. The system should be available to access from a web browser. The system will be built for modern web browsers, taking into account at least the latest versions of Google Chrome and Firefox. The system should also be fully compatible with at least Internet Explorer 9.0.
* **SharePoint Coupling**. CSi hosts a SharePoint environment. The application should be easily accessible from SharePoint, and it would be desirable for the system to reside within SharePoint’s environment. There is no relevant data for the system to interact with within SharePoint itself, though.
* **Windows Server**. The system should be able to be deployed to Windows Server 2008 R2.

There are no specific requirements pertaining to application performance. The system does need to be useable, even when there are numerous records within the database. The system should not experience unusual slowdowns even when there are numerous records.

A feature which would be nice to have is the functionality to show a calendar view in which time-based entities can be shown. This means that the framework should either have calendar functionality built-in, or provides the possibility of including custom code.

# http://webservices.thesba.com/wp-content/uploads/2013/05/sharepoint-20131.jpgDevelopment Platform

There are a couple of development platforms suitable for our system, given the requirements. Three options where considered, both because of platform restrictions (it should be able to run on Windows Server 2008 R2) as well as of knowledge restrictions (knowledge of C# is commonly shared between both the organization and the development team). The options that were considered are:

* SharePoint 2013;
* ASP.NET;
* ASP.NET in combination with Angular.js.

The question that is considered here is: *What development platform should we use, and what are the benefits over the alternatives?*

## Framework Comparison

In order to compare the platforms, we listed the requirements and other points of interest, and then compared each platform, trying to see which platform provided benefits over the others. Below you will find a rather large table comparing those options, along with their benefits and drawbacks.

| Requirement | SharePoint | ASP.NET | ASP.NET & Angular.js |
| --- | --- | --- | --- |
| Active Directory Authentication  SharePoint works natively with Active Directory. With ASP.NET as a backend, Active Directory integration is also possible. | + | + | + |
| User Roles and Permissions  User roles can be assigned to users within SharePoint. When building in ASP.NET, you can define permissions on a very granular level. This isn’t possible in SharePoint in a way that is as customizable as within ASP.NET. | ± | + | + |
| Complex Entity Relationships  It is possible to create complex relationships using the Entity Framework, which is available within ASP.NET.  When you want complex relationships within SharePoint, a REST API is needed with which SharePoint can communicate. Facilitating this would need a separate system.  When building the system with Angular, care needs to be taken with input validation and model consistency. There are libraries that make this process much easier, such as Breeze.js, with which you can export a data model from the Entity Framework, ready to use on the client side and complete with validation. | ± | + | ± |
| Forms  Forms can be created programmatically using all solutions. All data needs to be sent to REST endpoints, so within SharePoint a separate environment will be needed.  Suggestions and autocomplete are not directly available within SharePoint, but it should be possible to create this functionality.  When using Angular, forms can be posted using AJAX, so that page reloads aren’t necessary. This makes the system feel more responsive. | ± | + | + |
| Document Generation  Document generation is not possible within SharePoint, so an external service will need to be created. It is possible using ASP.NET, or client-side using Angular. | − | + | + |
| Notifications  Scheduled notifications within SharePoint are possible for certain views, but there is no easy integration with the data model.  A Windows Service, Windows Task or Agent is needed to perform scheduled notifications when using ASP.NET. | ± | + | + |
| Web Deployment  All solutions are accessible from a browser. Special care needs to be taken in the choice of JavaScript framework, though, in order to ensure browser compatibility. | + | + | ± |
| SharePoint Coupling  At the time of writing, there is no data coupling needed between the existing SharePoint environment and the new system. Coupling between the two systems for this reason is not essential.  When this requirement changes, it is good to know that ASP.NET can interface with SQL server, making data available from SharePoint. | + | + | + |
| Windows Server  All applications are able to be deployed to Windows Server 2008 R2. | + | + | + |
| Development Team Experience  The development team has experience with certain technologies. Developing with languages and frameworks that the development team already has experience with, should make development faster and should return a more stable system.  The development team has experience with ASP.NET and Angular, but not yet with SharePoint. | − | + | + |
| Business Development Experience  The people responsible for maintaining and further developing the project need to be able to do so. This means that they need a background in programming.  Experience in SharePoint is available. Experience with C# is also available, which is used in ASP.NET. JavaScript is not yet known within the organization, but should be relatively easy to pick up when there is a background in C#. | + | ± | ± |

## Differences between SharePoint Editions

CSi uses SharePoint Foundation at the moment, which is a free edition. There are two other self-hosted SharePoint environments available, which are SharePoint Server 2013 Standard CAL and SharePoint Server 2013 Enterprise CAL.

The latter two editions have a broader and more advanced feature set. Up until now, most of those features weren’t strictly necessary. When implementing an application within SharePoint, however, some of those paid features are necessary. For a full feature comparison, please refer to the page “SharePoint Online Service Description” (<http://technet.microsoft.com/en-us/library/jj819267.aspx#bkmk_FeaturesOnPremise>). Necessary paid features include:

* **Business Connectivity Services (BCS)**. The BCS component includes functionality that is necessary for the operation of the application, for example Business Data Web parts, which are used like standard SharePoint web parts but are based on external content types, or the OData connector with which you can connect to a RESTful OData endpoint. This is because the data will be hosted within an SQL environment; SharePoint itself does not have the capability to enable complex data models with many relationships.
* **Forms Based Applications**. A form view is basically a view that contains controls. A Forms Based Application lets the user create and use one or more forms within the application. These forms are necessary to provide a point where users can enter data.
* **InfoPath Forms Services**. Forms Services provides a Web browser form-filling experience in SharePoint, based on form templates that are designed in InfoPath. This is an addition or alternative to Forms Based Applications.
* **Workflow 2013**. SharePoint 2013 workflows are powered by Windows Workflow Foundation 4 (WF), which was significantly redesigned from earlier versions. Workflows enable custom entities, like the one in this project’s data model, to be processed in several steps. Workflows are able to communicate using the BCS OData connector.

## SharePoint Licensing Types

SharePoint is licensed with a Server/Client Access License model. SharePoint Server 2013 is required for each running instance of the software, and a CAL is required for each person or device accessing a SharePoint Server. For more information about this licensing model, please refer to the “SharePoint 2013 licensing” page (<http://office.microsoft.com/en-001/sharepoint/sharepoint-licensing-overview-collaboration-software-FX103789438.aspx>).

Please refer to the Microsoft License Advisor for a rough quote for licensing (<http://mla.microsoft.com/>). An example of licensing costs:

* **Country**: The Netherlands
* **Agreement term**: Two-Year
* **Payment schedule**: Pay Up-front
* **Software Assurance Coverage**: None
* **Organizational Type**: Corporate
* **SharePoint Server**: License Only
  + 1 instance running at a given time (no redundancy, normally you would want at least two instances running in order to account for failures)
  + 300 devices
  + 200 users

**Costs:**

SharePoint Server, single license: €10.699

Microsoft SharePoint Standard CAL, €171 per user, 200 licenses: €34.200

Total: €44.899

## Conclusion

While certain functionality is available within SharePoint, a backing service will be needed to provide data for workflows and to facilitate functionality not provided in SharePoint such as document generation. This means that, although an application could be built within SharePoint, not everything can be managed by the SharePoint environment. Building the application within SharePoint would not yield additional returns over the alternatives when it comes to the delivered functionality, but it would make the application easier to maintain.

SharePoint Server 2013 Standard CAL or higher is needed in order to provide functionality. CSi currently runs SharePoint Foundation, meaning that licensing would have to be bought. Even when the correct edition is used, there still remains the need for a backing service which implements, at the very least, the OData protocol, such as an ASP.NET RESTful service. For some functionality such as document generation, a custom endpoint is needed.

The choice of platform then depends on the following factors:

* **Licensing**. The licensing costs are quite steep. In order to start the development on time, CSi will need to know whether they want to invest in SharePoint in a relatively short amount of time. If they eventually decide that SharePoint 2013 might not be worth it, then our system would be unusable.
* **JavaScript**. The choice of platform, when CSi decides to implement the system within ASP.NET instead of in SharePoint, then is to decide whether or not the costs of using JavaScript outweigh the benefits. Choices for and against the use of JavaScript can be read in the next chapter.

# Using a JavaScript Framework

It is often useful to use a framework when developing a system. Frameworks provide functionality that is common to many systems, saving developers a lot of time that would otherwise be spent implementing that same functionality.

The choice of a framework is important because it often has a large impact on the way the system is built, tested, deployed and maintained. More importantly, not all frameworks support the technical requirements of the project.

The framework that was considered is Angular.js. This framework enjoys a large developer community and is backed by Google. Its versions are now stable and ready for production use; Google itself uses it in some of its public projects such as DoubleClick. The developer team already has some experience with Angular.js, and thinks it can yield major benefits related to functionality and user experience.

Next to the benefits, using Angular.js will also have a number of drawbacks. The question that is considered here is: *Why should we use Angular.js in combination with ASP.NET instead of just ASP.NET alone?*

## Choice Comparison

In order to provide a truthful comparison, a number of aspects were weighed. You will find those aspects below, along with their benefits or drawbacks.

| Aspect | Without Angular.js | With Angular.js |
| --- | --- | --- |
| Ease of Implementation  The development team has some experience with both options. We estimate that both options will take around the same amount of time. | + | + |
| Ease of Maintenance  When using any framework, the learning curve increases. This causes maintenance to be more expensive, and this cost will need to be weighed against the benefits. Documentation may ease the learning curve, but that does not remove it. | + | ± |
| Application Architecture  There is enough opportunity in both options to build a reliable, expandable and maintainable architecture. Using Angular might complicate the architecture, but it does provide a clear division between the server and client, making it more obvious where functionality needs to be implemented. | + | + |
| Page Reloading  When using just ASP.NET, pages need to be reloaded after navigation to a new page or when submitting a form. With a JavaScript framework, only a part of the page needs to be reloaded, decreasing load times and increasing responsiveness. Because functionality that would refresh pages is used so often, not having to reload would provide a very noticeable benefit. | ± | + |
| Overall Performance  The performance impact of both options might not be noticeable considering the amount of users. When using Angular, though, less server processing power should be needed because the template processing is mostly happening client side instead of server side. | + | + |
| Testability  Angular is built from the ground up to be testable. Every component should be testable in isolation. While this is also possible in ASP.NET, using Angular would provide a better testing experience because only business and data logic would reside on the server, while Angular would contain mostly view logic. | + | + |
| Experience  The development team has already built an application using ASP.NET, and also an application using Angular. We think it would provide a better learning experience for us if we were to combine the two. | ± | + |

## Conclusion

Provided CSi chooses to build the system with ASP.NET, and based on the above arguments, we think it would be a better option for this project to develop the application using ASP.NET in combination with Angular.js. It will provide a better experience for both the users and the development team. Choosing Angular will make the learning curve for maintainers higher, but we think that we can remedy most of that learning curve by writing proper documentation.

# Using an HTML Template

For this project, the ProUI HTML template (see <http://themeforest.net/item/proui-responsive-admin-dashboard-template/6769035>) is used. While there is already a style sheet which is used within CSi, there are several benefits to using this HTML template:

| Aspect | HTML template | Existing style sheet |
| --- | --- | --- |
| Uniform Styling  Both options provide many styles that would give the system a uniform look and feel. | + | + |
| Component Availability  The template includes several plugins and components, such as Google Maps integration and table filtering. Most of these components would take a lot of time to build from scratch. With a template, many components are included and styled for you already. | + | ± |
| Development Time  When developing with a template, there are many sample pages to take code from to adapt to the requirements. They provide a head start in solving the layout. With a style sheet, there aren’t as many pages to take samples from. | + | ± |

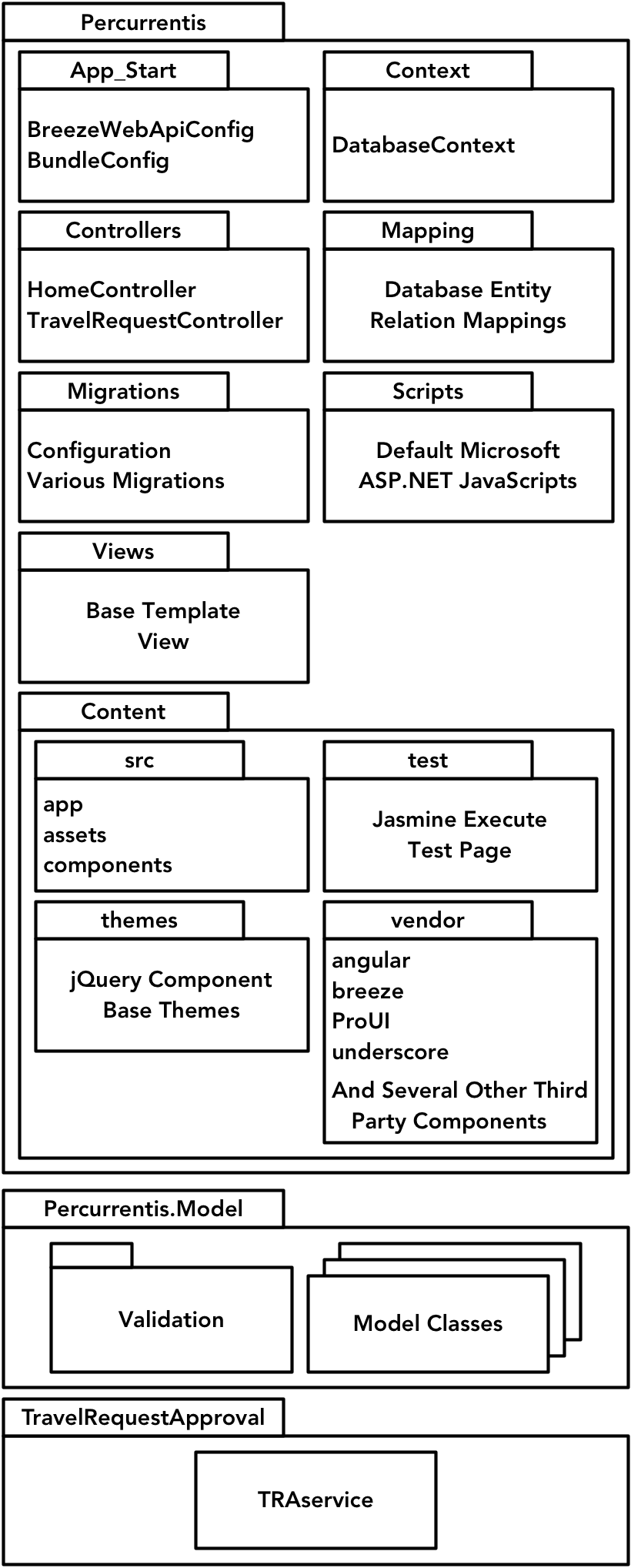
The costs of the template were 20 USD. These costs are far outweighed by the savings in development time, which is much more expensive. CSi has therefore chosen to accept our proposal for using an HTML template.

# Dependencies / Versions

The project has the following dependencies:

* **ASP.NET MVC 4**. This version is able to run on Windows Server 2008 R2, and it is possible to develop for MVC 4 using Microsoft Visual Studio 2010 and higher.
* **.NET 4.5**. ASP.NET MVC 4 is built on top of .NET 4.5.
* **Angular.js 1.2.13** [<https://angularjs.org/>]
* **Angular Localytics Chosen** [<https://github.com/localytics/angular-chosen>]
* **Angular UI Bootstrap 0.10.0** [<http://angular-ui.github.io/bootstrap/>]
* **Breeze.js 1.4.13** [<http://www.breezejs.com/documentation/download>]
* **Underscore.js 1.6.0** [<http://underscorejs.org/>]
* **Q.js 1.0.1** [<https://github.com/kriskowal/q>]
* **ProUI** [<http://themeforest.net/item/proui-responsive-bootstrap-admin-template/6769035>] (Ask Roel van Rijswijck for the license/download information)

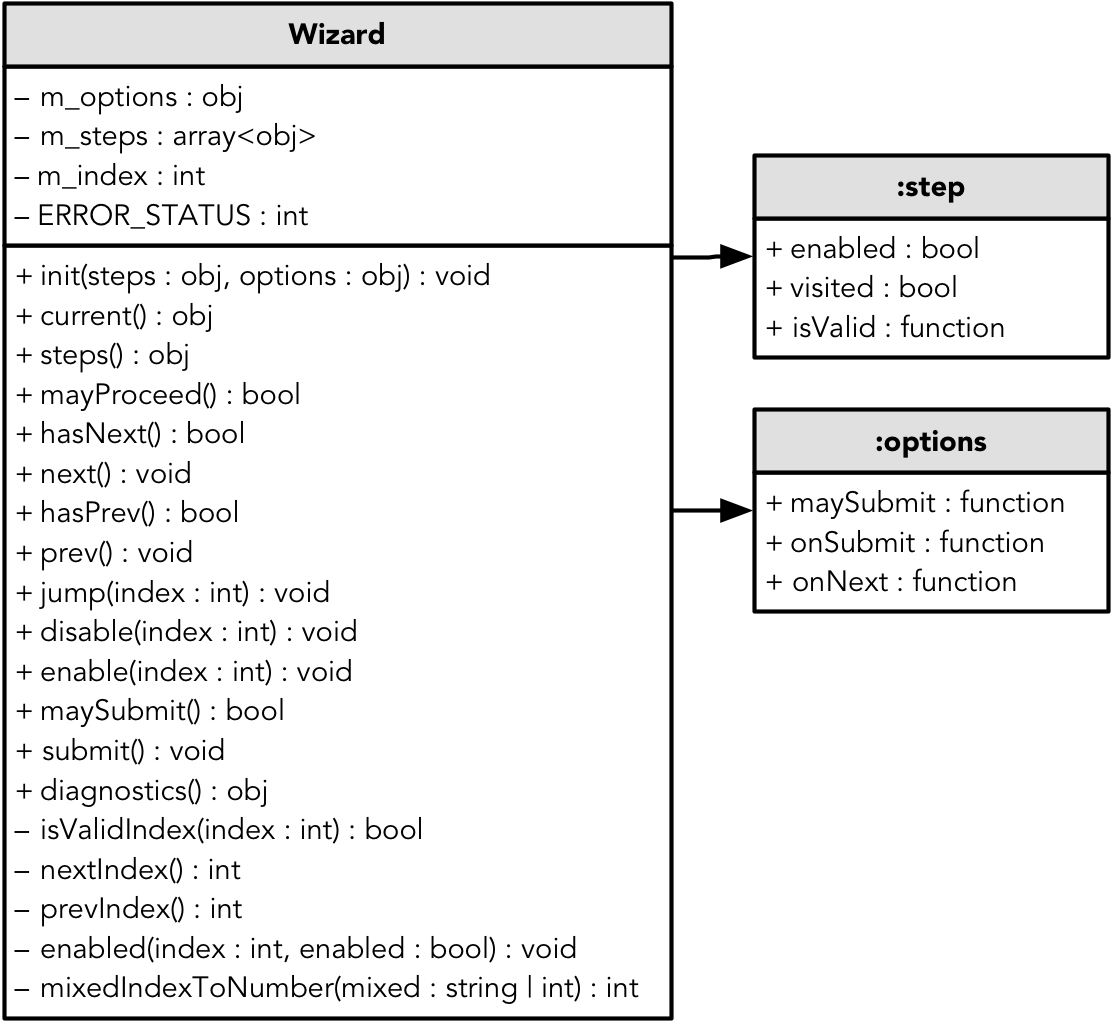
# Package Diagram

The system contains three seperate projects: **Percurrentis.Model** contains all of the model classes and validation logic; **TravelRequestApproval** contains a Windows Service which runs the process as noted in the section Travel Request Approval. **Percurrentis** contains the client application code and the controllers which supply data to the client.

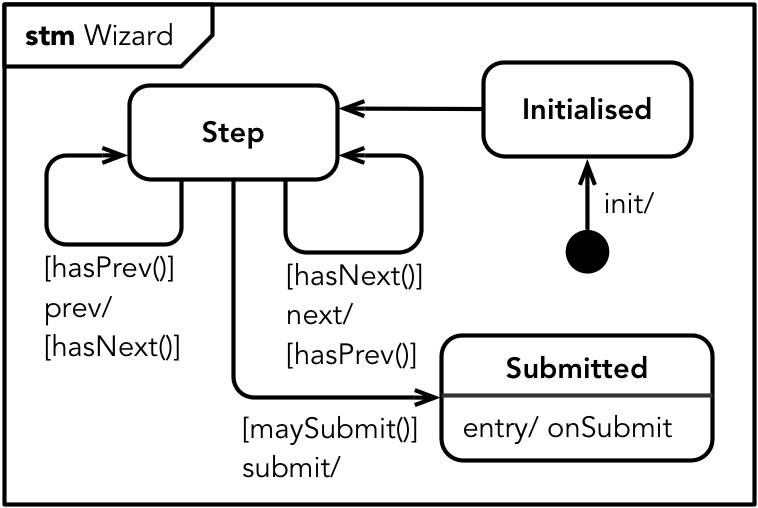
Breeze.JS is used as a data supplier, which takes the information from the Entity Framework. The controllers then use Breeze components on the server side in order to provide data to Breeze on the client side.

Client side code can be found within Percurrentis/Content/src.

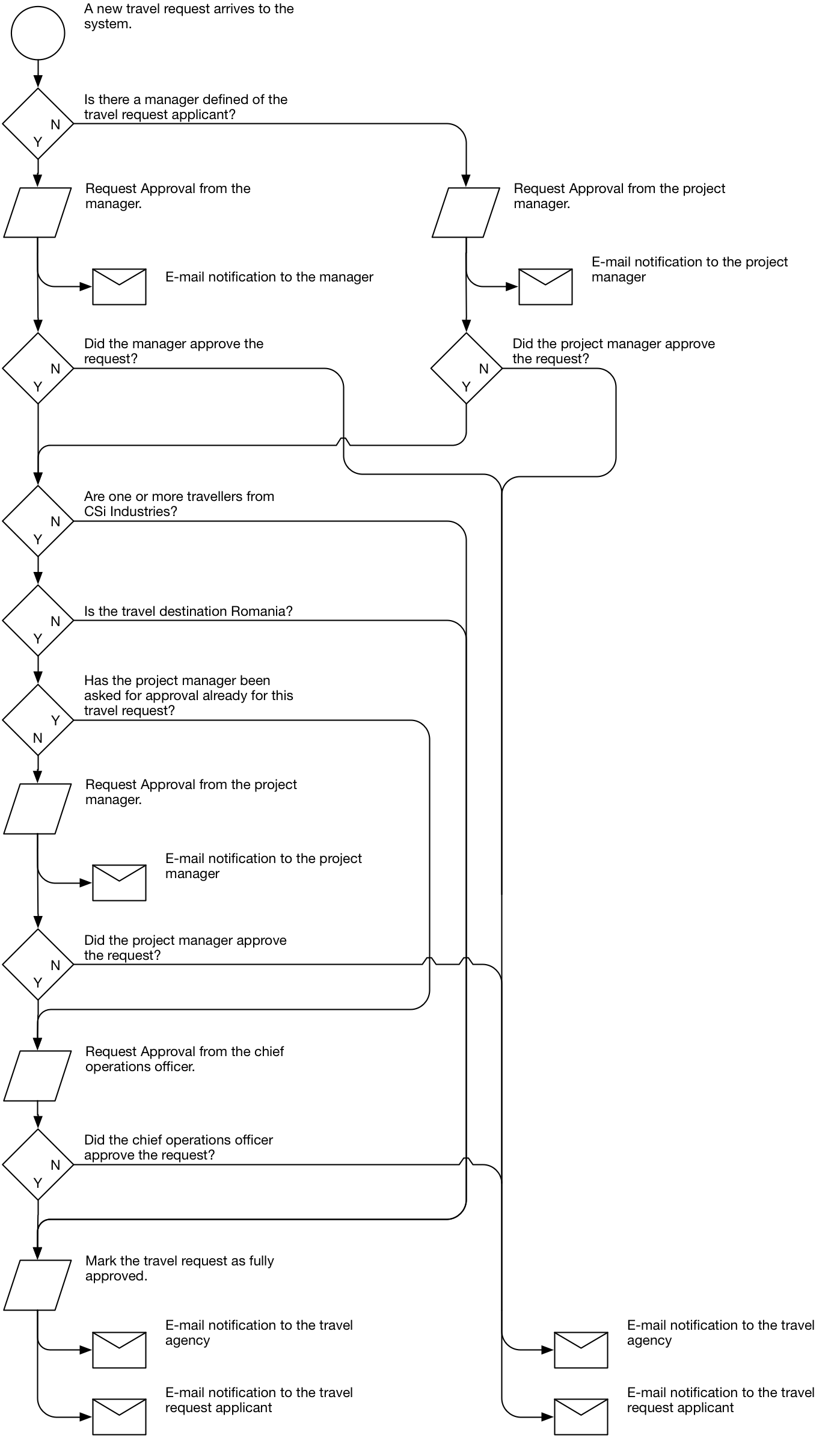
# Form Wizard



The form wizard is used in user story #1, File Travel Request. Unit tests for this component exist within the same folder. The entire mechanism can be described as a state machine:

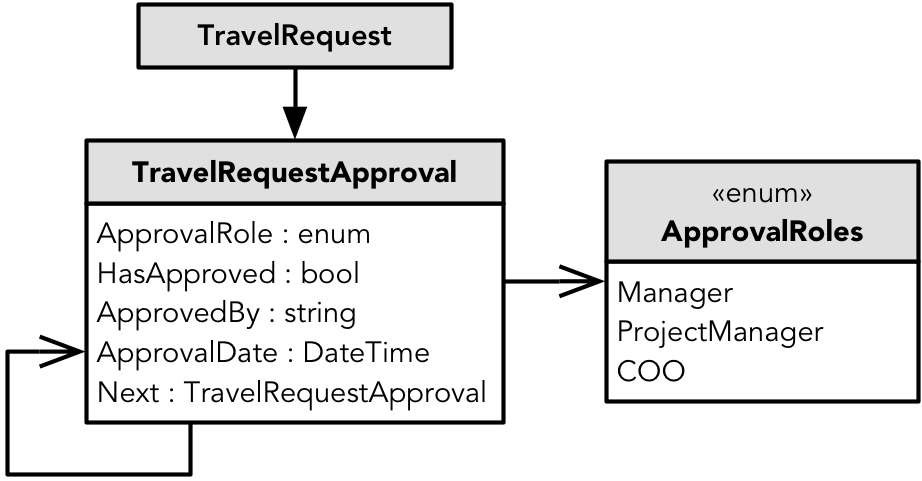
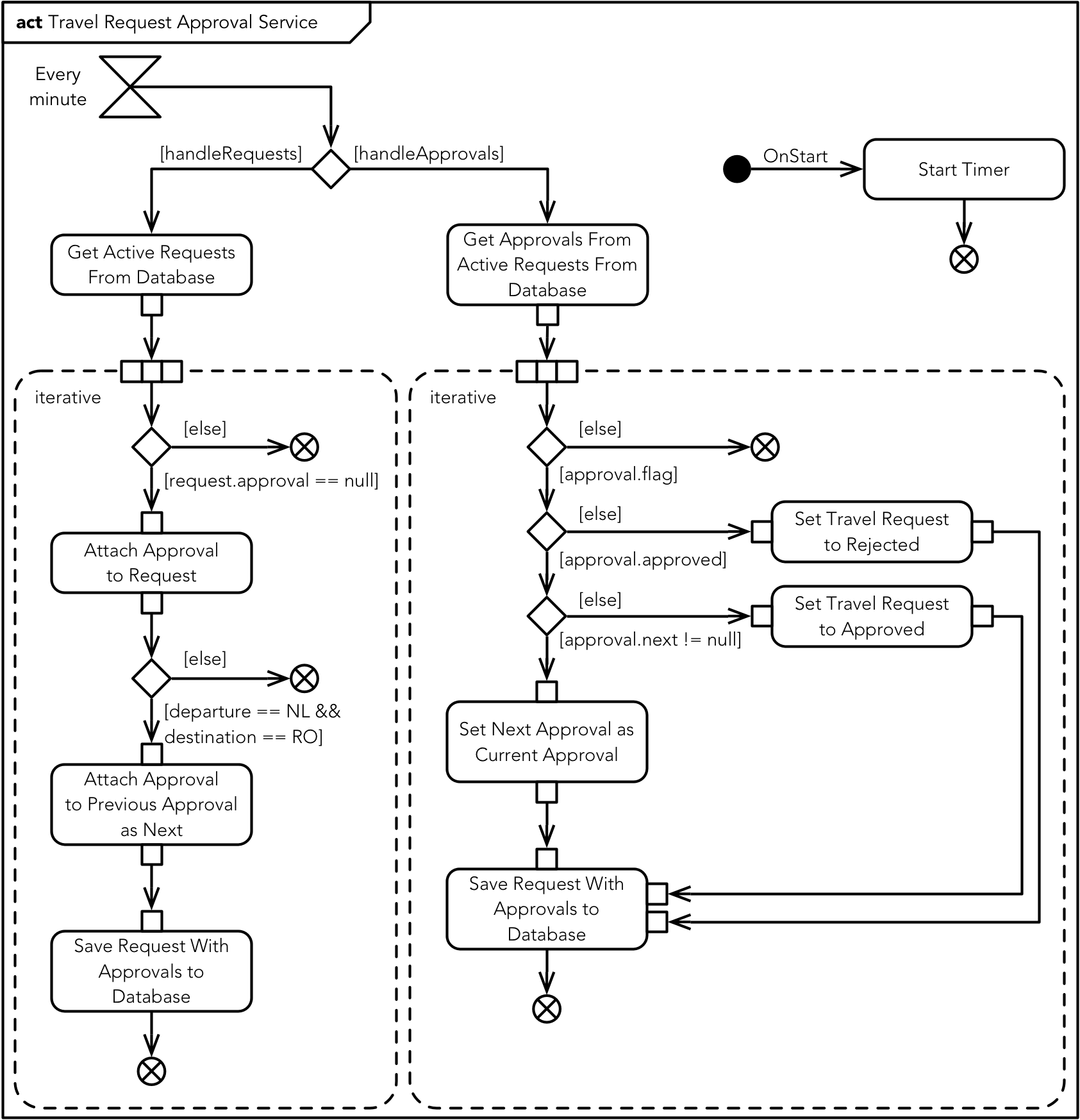


# Travel Request Approval



The previous diagram describes the requirements that the system has to fulfill regarding the approval of travel requests.

The diagrams below capture the internal process of the Travel Request Approval service, which periodically handle new requests, attaching the necessary objects to them.



# Server Technologies

This document describes the technological decisions made upfront for the back-end (server side) of the system. As will come forward in this document the decisions are chained to each other and one decision lead to other decisions.

## Data persistence

Data persistence is necessary in order to retrieve input and use resources. The user data has to be permanently stored and therefore a certain way of persistence is necessary.

For data persistence are two options. The first option is to load every piece of data from a file into the memory whilst the second choice is setting up a database. Since the database is very likely to become big and contain a lot of data, the first option is not possible. Therefore a database is necessary in order to meet the requirements.

The chosen database software is an SQL-database running on Microsoft SQL server. The software that the company uses also mainly runs on databases on a SQL server, so this fits in with the company’s knowledge. The IT department uses Microsoft SQL Server Management Studio to manage the database, which can also be used to manage the database that I choose.

## Database Modeling

Creating the model of the database can be done with a wide variety of technologies. The choices to which I limited the research involve the three technologies involving the Entity Framework (model-first, database-first and code-first), NHibernate and manual database creation.

**NHibernate** is a code-first database modeler and a good alternative for the commonly used Entity Framework (code-first). One of the disadvantages is that NHibernate has no integrated migration tool. This makes it necessary to use extra software, such as FluentMigrator, in order to migrate the model to the database. Another disadvantage is that my personal experience with NHibernate is very small compared to my experience with the Entity Framework.

**Manual database creation** is an option that is not a viable option since this means there is no knowledge about the model in the application. Due to the outcome of the research involving the database modeling, code-first was highly preferred. This relates to the chosen client-server technology, which involves a data model which is code first with Entity Framework of NHibernate.

The **Entity Framework** appeared to be the best option since the integration with the client-server technology is one of the better integrations and my experience with the Entity Framework is extensive. This technology has a built-in migration and update tool and allows a configuration to exist in the code. This means that no third-party tools are necessary in the process from code-first model creation to a real database deployed on the server.

The implementation of the Entity Framework will be executed by installing the Entity Framework from the NuGet package manager, which is built-in in Visual Studio, the development environment.

## Client-Server technology

The client-server technology has a wide variety of options. Microsoft offers the ASP.NET Web API to create an endpoint on the server which is easily accessed from the client. Another option I researched is the BreezeJS library for the client in combination with Breeze Server for the server.

The first option requires a lot of coding and raw data handling. ASP.NET Web API does have a lot of options, but the main function of the connection between the client and the server is to send data. This data is modeled by the code-first model and for this reason a framework which sends data as a full entity (with relations) and is also able sends the entity’s metadata is highly favored.

BreezeJS for the client and Breeze Server for the server supply a framework where the data retrieved from the database is processed as an instance of an entity. The built-in methods can make new entities, edit them or remove them. After this the framework can save the entities to the database. Since a lot of the logic has been simplified in this framework, a lot of intensive functions are easily handled. The client-side provides a manager that, after receiving the metadata, also approaches the data as instances of an entity. When the client makes edits to existing entities (sent to the client) or creates new entities according to the metadata, this can be easily sent back to the server which on its turn approaches it as instances of entities. By using Breeze, an integration of the model trough all the layers (database, server, and client) is consists.