|  |
| --- |
| Enterprise Tracking Application |
| Software Architecture Document |
| Prepared For: Perspecta  Location: Austin, TX |
| Date: 9/18/2020  Revised: 10/2/2020 |
| Angeleen Abesamis  Remnin Ferrer  James Fritz  Yeremy Joseph  Eric Leon |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Comments | Author |
| 1.0 | 10/2/2020 | Original Submission |  |
|  |  |  |  |
|  |  |  |  |

# Abstract

This document serves as the software architecture document for the Enterprise Tracking Application that is being developed by the University of Texas at Dallas project team for Perspecta Inc.

Table of Contents

[Revision History 2](#_Toc54690080)

[Abstract 3](#_Toc54690081)

[List of Figures 5](#_Toc54690082)

[List of Tables 5](#_Toc54690083)

[1. Introduction 6](#_Toc54690084)

[1.1 Purpose 6](#_Toc54690085)

[1.2 Scope 6](#_Toc54690086)

[1.3 Definitions, Acronyms, and Abbreviations 6](#_Toc54690087)

[1.4 References 6](#_Toc54690088)

[1.5 Overview 6](#_Toc54690089)

[2. Architectural Style 8](#_Toc54690090)

[3. Architectural Model 9](#_Toc54690091)

[3.1 4+1 Architecture View Model Diagram 9](#_Toc54690092)

[3.2 System Architecture Models/Views 10](#_Toc54690093)

[4. Technology, Software, and Hardware Used 25](#_Toc54690094)

[4.1 Technology 25](#_Toc54690095)

[4.2 Software 25](#_Toc54690096)

[4.3 Hardware 27](#_Toc54690097)

[5. Rationale for the Architectural Style and Model 28](#_Toc54690098)

[5.1 Data Flow Architectural Style 28](#_Toc54690099)

[5.2 Model-View-Controller Architectural Style 28](#_Toc54690100)

[5.3 Layered Architectural Style 28](#_Toc54690101)

[6. Evidence of Configuration Management 30](#_Toc54690102)

# List of Figures

|  |  |  |
| --- | --- | --- |
| Figure | Description | Page |
| 3-1 | 4+1 View Model Diagram | 9 |
| 3-2 | Perspecta ETA N-tier Architecture | 11 |
| 3-3 | User login activity diagram | 12 |
| 3-4 | Add source application activity diagram | 13 |
| 3-5 | List source applications activity diagram | 14 |
| 3-6 | Update source application activity diagram | 15 |
| 3-7 | Delete source application activity diagram | 16 |
| 3-8 | Add destination application activity diagram | 17 |
| 3-9 | List destination applications activity diagram | 18 |
| 3-10 | Update destination application activity diagram | 19 |
| 3-11 | Delete destination application activity diagram | 20 |
| 3-12 | ETA architecture model diagram | 21 |
| 3-13 | ETA component diagram | 22 |
| 3-14 | ETA development package diagram | 23 |
| 3-15 | ETA deployment diagram | 24 |
| 3-16 | Hardware structure diagram | 27 |

# List of Tables

|  |  |  |
| --- | --- | --- |
| Table | Description | Page |
| 5-1 | Architecture Style Tradeoffs. | 29 |

# 1. Introduction

The introduction of the Software Architecture Specification (SAS) provides an overview of the entire document including purpose, scope, definitions, acronyms, abbreviations, references and overview of the SAS. The aim of this document is to identify and evaluate candidate architectures for the Enterprise Tracking Application software and select the architecture that will provide the most stability as the most preferred.

### 1.1 Purpose

This document provides a comprehensive architectural overview of the Enterprise Tracking Application using several different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system. This document is intended to guide the detailed design phase of the project.

### 1.2 Scope

This section describes the boundaries of what the system will do/not do, from an architectural perspective. The Enterprise Tracking Application (ETA) will be composed of three main component categories; the presentation layer, the application layer, and the data layer.

The primary responsibility of the presentation layer will be to provide forward-facing user access to the system through a web browser. The data layer will store the application data. The application layer will provide the RESTful web services necessary for the presentation layer to utilize the information in the data layer.

### 1.3 Definitions, Acronyms, and Abbreviations

**CRUD:** Create, Retrieve, Update, Delete database operations.

**ETA**: Enterprise Tracking Application

**SAD**: Software Architecture Document

### 1.4 References

[KRU] P. B. Kruchten, "The 4+1 View Model of architecture," in IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995, doi: 10.1109/52.469759.

[RUPRSA] J. Maréchaux. “Developing a J2EE Architecture with Rational Software Architect Using the Rational Unified Process.” IBM Developer. https://www.ibm.com/developerworks/rational/library/05/0816\_Louis/index.html (accessed Sep. 23, 2020).

[SHAW] M. Shaw, "Comparing architectural design styles," in IEEE Software, vol. 12, no. 6, pp. 27-41, Nov. 1995, doi: 10.1109/52.469758.

### 1.5 Overview

To fully document all the architecture aspects, the software architecture document contains the following subsections:

* Section 1, Introduction: The introduction of the SAD provides an overview of the entire document. It includes the document introduction, purpose, scope, definitions, acronyms, and abbreviations, references, and the document structure of this Software Architecture Document.
* Section 2, Architectural Style: This section describes the architecture style and how it supports the various application features.
* Section 3, Architectural Model: The architectural model section includes the subsystem packages and views of the system.
* Section 4, Technology, Software, and Hardware Used: This section describes the technologies being used to implement the product including any software, hardware, components, connections, and constraints.
* Section 5, Rationale for the Architectural Style and Model: This section offers a brief description of the rationale for the system components, connections, and constraints.
* Section 6, Evidence of Configuration Management: This section contains a single image depicting evidence that the SAD has been placed under configuration management.

# 2. Architectural Style

The development team has chosen to use a layered architecture style to implement the Enterprise Tracking Application. The layered architecture style supports the application by utilizing an application logic layer to serve as the data controller. This will enable the application to use CRUD operations on the data within the data layer while remaining on a single page in the presentation layer.

# 3. Architectural Model

The architectural model used in this document is based on the 4+1 View Model of Software Architecture by Philippe B. Kruchten. The 4+1 View Model of Software Architecture is a model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views.

### 3.1 4+1 Architecture View Model Diagram

|  |
| --- |
|  |
| Figure 3-1. 4+1 View Model Diagram |

### 3.2 System Architecture Models/Views

The Perspecta Enterprise Tracking Application is divided into layers based on the N-tier architecture. The layering model is based on a responsibility layering strategy that associates each layer with a particular responsibility.

This structure has been chosen because it isolates various system responsibilities from one another in order to support the RESTful API as well as system development.

Each layer has specific responsibilities.

* The presentation layer deals with the presentation logic and with rendering the functions within the page.
* The application logic layer manages access to the data layer as well as the information contained within the data layer.
* The data layer deals with managing the database records and is being provided by the customer.

#### 3.2.1 Logical View

##### 3.2.1.1 Overview

The Logical View describes the design's object model.

|  |
| --- |
|  |
| Figure 3-2. Perspecta ETA N-tier Architecture |

#### 3.2.2 Process View

##### 3.2.2.1 Overview

The Process View is used to show the high-level structure and composition of the application processes that are available to the user.

##### 3.2.2.2 Login Process

|  |
| --- |
| Diagram  Description automatically generated |
| Figure 3-3. User login activity diagram |

##### 3.2.2.3 Add Source Application Process

|  |
| --- |
| Diagram  Description automatically generated |
| Figure 3-4. Add source application activity diagram |

##### 3.2.2.4 List Source Applications Process

|  |
| --- |
| Diagram  Description automatically generated |
| Figure 3-5. List source applications activity diagram |

##### 3.2.2.5 Update Source Application Process

|  |
| --- |
| Diagram  Description automatically generated |
| Figure 3-6. Update source application activity diagram |

##### 3.2.2.6 Delete Source Application Process

|  |
| --- |
| Diagram  Description automatically generated |
| Figure 3-7. Delete source application activity diagram |

##### 3.2.2.7 Add Destination Application Process

|  |
| --- |
|  |
| Figure 3-8. Add destination application activity diagram |

##### 3.2.2.8 List Destination Applications Process

|  |
| --- |
|  |
| Figure 3-9. List destination applications activity diagram |

##### 3.2.2.9 Update Destination Application Process

|  |
| --- |
|  |
| Figure 3-10. Update destination application activity diagram |

##### 3.2.2.10 Delete Destination Application Process

|  |
| --- |
|  |
| Figure 3-11. Delete destination application activity diagram |

#### 3.2.3 Development View

##### 3.2.3.1 Overview

The Development View describes the layers and subsystems of the application.

##### 3.2.3.2 Architecture Model Diagram

|  |
| --- |
| Diagram  Description automatically generated |
| Figure 3-12. ETA architecture model diagram |

##### 3.2.3.3 Component Diagram

|  |
| --- |
|  |
| Figure 3-13. ETA component diagram |

##### 3.2.3.4 Development Package Diagram

|  |
| --- |
|  |
| Figure 3-14. ETA development package diagram |

#### 3.2.4 Physical View

##### 3.2.4.1 Overview

The Physical View describes the mapping of the software onto the hardware and shows the system's distributed aspects.

##### 3.2.4.2 Physical Deployment Diagram

|  |
| --- |
|  |
| Figure 3-15. ETA deployment diagram |

# 4. Technology, Software, and Hardware Used

### 4.1 Technology

* Spring Boot – Spring Boot is an open source Java-based framework used to create Micro Services.
* Java – Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.
* RESTful Web Services – Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web.
* Angular – Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.
* Typescript – TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for development of large applications and transcompiles to JavaScript.
* JavaScript – JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm.
* Microsoft SQL Server – Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network.
* Structured Query Language (SQL) – SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.

### 4.2 Software

* Spring Boot – Spring Boot is an open source Java-based framework used to create Micro Services.
* Java – Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.
* RESTful Web Services – Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web.
* Angular – Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.
* Typescript – TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for development of large applications and transcompiles to JavaScript.
* JavaScript – JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm.
* Microsoft SQL Server – Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network.
* Structured Query Language (SQL) – SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.
* Node.js – Node.js is an open-source, cross-platform, back end, JavaScript runtime environment that executes JavaScript code outside a web browser.
* Angular CLI – The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications directly from a command shell.
* Eclipse IDE – Eclipse is an integrated development environment used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment.
* Maven – Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project.
* Microsoft SQL Server Management Studio (SSMS) – SSMS is a software application developed by Microsoft that is used to configure, manage, and administer all components within Microsoft SQL Server. This tool includes both script editors and graphical tools which work with objects and features of the server.

### 4.3 Hardware

|  |
| --- |
|  |
| Figure 3-16. Hardware structure diagram |

* Local System – User’s device with existing operating system and web browser to access the Enterprise Tracking Application website.
* Application Server – An existing web server provided by the client that provides the RESTful web service and front-end controls.
* Database Management System Server – An existing database server provided by the client.

# 5. Rationale for the Architectural Style and Model

### 5.1 Data Flow Architectural Style

This Architectural style consists of filters that perform transformations on data received and pipes which serve as connectors for the stream of the data being transformed. They have shown to be highly effective in Embedded system software design, where the system is manipulated by process control variable data, and applications that aim to maintain specified properties of the outputs of the process at given reference values. Although there are apparent advantages to this style, it is not suitable for this project because it only works in one direction and we would have to loop through the same algorithm which will cause extreme overhead of data transformation between filters, and implementation complexity. This style is not suitable for dynamic interactions that will be needed in this project therefore we have decided NOT to take this architectural style approach.

### 5.2 Model-View-Controller Architectural Style

The model-view-controller (MVC) has been the traditional architectural pattern for developing applications having user interface. MVC renders testability, maintainability and scalability. It also satisfies Single Responsibility Principle of SOLID. Some of the disadvantages, however, are that the View depends both on Controller and Model so when the updating the UI data the view might have to go to the model for some functionality and to the controller for other, and that the Model just does too much work from getting the data from the network/database to informing the Controller about whether it could get the data or not, and even preparing the result to be displayed on the View. Other architectural styles have come up with solutions to these issues. For these reasons and because the sponsor has advised us against this style, we decided not to implement the Model-View-Controller Architectural style.

### 5.3 Layered Architectural Style

The development team selected the Layered Architectural style to accommodate for functionalities and RESTful services that the utilized by the application. The architectural style is comprised of layers perform specific roles in the application and contain components and modules with similar functionalities. It provides an abstract view of the system and defines responsibilities and relationships between each layer. High level requirements can be easily implemented as separate layers, which will streamline development by minimizing dependencies, as well as improve testability, maintainability and reusability through layer isolation. This also allows high cohesion due to well-defined functional boundaries and low coupling from abstraction and event-based communication. While the architecture has innate performance issues, the trade-off for ease of implementation and improvement to many nonfunctional requirements puts it above the other architectural styles considered for this software.

#### 5.4 Reason for Architecture Selection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Data Flow | MVC | Layered Architecture |
| Modifiability | Algorithms | - | + | + |
| Data Representation | -- | + | ++- |
| Enhanceability | Add Function | +- | ++- | ++ |
| Performance | Space | ++ | + | - |
| Time | + | + | - |
| Reusability |  | -- | + | + |
| Intuitiveness |  | - | + | +- |
| Scalability |  | -- | - | ++ |
| Security |  | + | - | ++ |

Table 5-1. Architecture Style Tradeoffs.

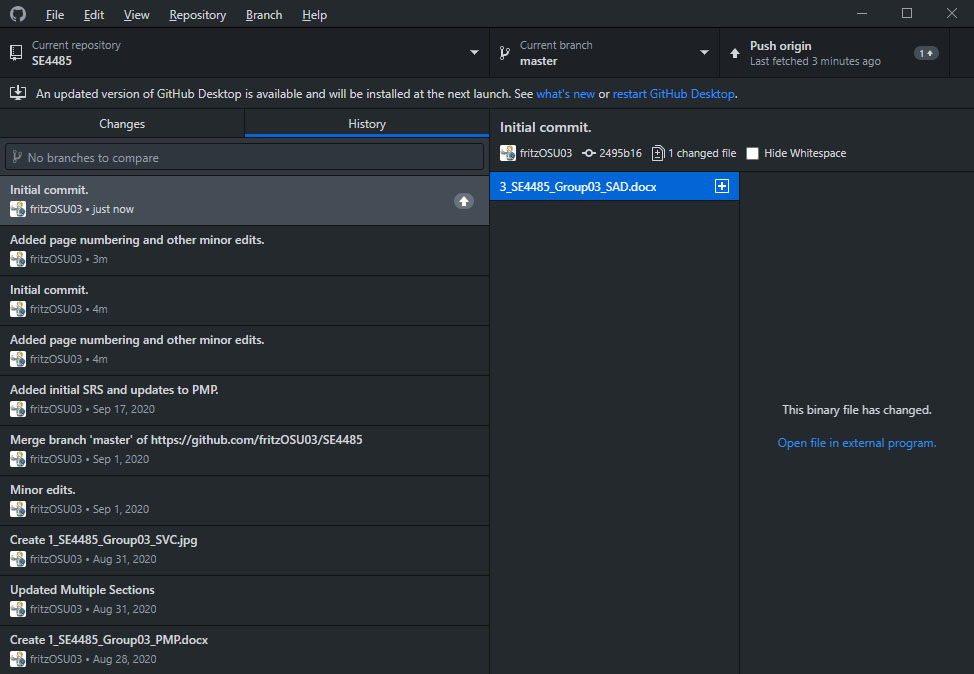
NFR1: Layered architecture is a key component of a RESTful architecture. Applying an Angular front end will ensure that the single-page application requirement is met.

NFR2 The system has been implemented using a RESTful architecture which will improve performance by minimizing the need to reload unnecessary web pages.

NFR3: This system has been implemented in Java and utilizes OOP with ADT which will allow it to function on different systems.

NFR4: A layered architecture implementation will allow the application to function securely by allowing users data access through a middleware layer. The single-page web design will also contribute to security by preventing sensitive data from being cached within the browser history.

# 6. Evidence of Configuration Management



END OF DOCUMENT