Architecture Vision

Tax Calculation

Version 0.1

For

Internal Customer

Contents

[1 Introduction 6](#_Toc509931823)

[1.1 Purpose 6](#_Toc509931824)

[1.2 Scope 6](#_Toc509931825)

[1.3 Definitions 6](#_Toc509931826)

[1.4 References 7](#_Toc509931827)

[1.5 Documentation Roadmap 7](#_Toc509931828)

[2 Executive Summary 9](#_Toc509931829)

[2.1 Key Decisions 9](#_Toc509931830)

[2.2 Key Risks and Open Issues 9](#_Toc509931831)

[3 Architectural Drivers 10](#_Toc509931832)

[3.1 Business Case 10](#_Toc509931833)

[**3.1.1** Business Goals 11](#_Toc509931834)

[**3.1.2** Major Features 11](#_Toc509931835)

[3.2 Service Level Agreement 11](#_Toc509931836)

[**3.2.1** Scope and Applicability 12](#_Toc509931837)

[**3.2.2** Service Quality Guarantees 12](#_Toc509931838)

[**3.2.3** Definitions 12](#_Toc509931839)

[3.3 Use Case Model 12](#_Toc509931840)

[**3.3.1** Use Case View <View Name> 12](#_Toc509931841)

[3.4 Domain Model 14](#_Toc509931842)

[**3.4.1** Domain View <View Name> 14](#_Toc509931843)

[3.5 Design Constraints 15](#_Toc509931844)

[3.6 Quality Attribute Scenarios 15](#_Toc509931845)

[4 Solution Architecture 17](#_Toc509931846)

[4.1 Big Picture 17](#_Toc509931847)

[**4.1.1** Solution Context 17](#_Toc509931848)

[**4.1.2** Solution Decomposition 19](#_Toc509931849)

[**4.1.3** Layered Application Structure 21](#_Toc509931850)

[4.2 Development Technology Stack 23](#_Toc509931851)

[**4.2.1** Development Languages, Frameworks, and Libraries 23](#_Toc509931852)

[**4.2.2** Development Tools 24](#_Toc509931853)

[**4.2.3** External Integration Points 25](#_Toc509931854)

[4.3 Architecture Part X 26](#_Toc509931855)

[**4.3.1** Decision View <View Name> 26](#_Toc509931856)

[5 Operation Plan 28](#_Toc509931857)

[5.1 Transition Phase 28](#_Toc509931858)

[**5.1.1** Infrastructure 28](#_Toc509931859)

[**5.1.2** Environments 29](#_Toc509931860)

[**5.1.3** Provisioning 29](#_Toc509931861)

[**5.1.4** Management 29](#_Toc509931862)

[**5.1.5** Monitoring 30](#_Toc509931863)

[5.2 Operation Phase 30](#_Toc509931864)

[**5.2.1** Service Operation Support 31](#_Toc509931865)

[**5.2.2** CMOD Activities 31](#_Toc509931866)

[**5.2.3** Continuous Improvement Process 31](#_Toc509931867)

[6 Implementation Roadmap 32](#_Toc509931868)

[6.1 Implementation Deliverables 32](#_Toc509931869)

[6.2 Implementation Milestones 32](#_Toc509931870)

[6.3 Estimate 33](#_Toc509931871)

[**6.3.1** Assumptions and Limitations 33](#_Toc509931872)

[**6.3.2** Estimate 33](#_Toc509931873)

[6.4 Team 33](#_Toc509931874)

[**6.4.1** Team Skillset 34](#_Toc509931875)

[**6.4.2** Team Structure 34](#_Toc509931876)

[7 Appendix A – Cross-Reference 36](#_Toc509931877)

[7.1 Driver Fulfillment Table 36](#_Toc509931878)

[7.2 View Reference Table 36](#_Toc509931879)

[7.3 Trade-off Reference Table 37](#_Toc509931880)

[7.4 Element Catalog 37](#_Toc509931881)

[8 Appendix B – Architecture Design Methodology 38](#_Toc509931882)

[9 Appendix C – How View is Documented 40](#_Toc509931883)

[9.1 View Structure 40](#_Toc509931884)

[9.2 Graphical Notation 44](#_Toc509931885)

[10 Appendix D – Estimation Methodology 46](#_Toc509931886)

[11 Guide to Architecture Vision Documentation – DELETE ME! 47](#_Toc509931887)

[11.1 Documentation Guidelines 47](#_Toc509931888)

[11.2 Architecture Vision Effort Estimation 52](#_Toc509931889)

[11.3 Styles to Use in the Document 53](#_Toc509931890)

[**11.3.1** Custom Section 55](#_Toc509931891)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 9/1/2023 | Initial | Filling section using AI | ishevt |
| 9/1/2023 | <x.x> | Approved | <name> |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

This Architecture Vision elicits the significant architecture drivers such as business, functional, nonfunctional requirements and constraints, defines architecture, and develops a roadmap for Single Entry implementation. The document is intended as a primary technical guidance into solution implementation for the development team.

The solution architecture is designed following the guiding principles outlined in [Appendix A – Architecture Design Methodology](#_Appendix_A_–).

## Scope

The document describes the proposed Single Entry architecture towards development of the solution that will satisfy business, functional, non-functional requirements and constraints provided by the Client. This Architecture Vision covers the following information:

* Significant architectural drivers for the Solution
* Proposed software architecture of the solution based on these drivers
* Technology stack and environment definitions
* Operations specific perspectives
* Development road map and high level estimates for effort, team size and skill sets.

## Definitions

The Definitions section lists the acronyms and terms used in this document which might possess lesser familiarity or double meaning to the reader.

|  |  |  |
| --- | --- | --- |
| # | Term | Definition |
|  | Financial Regulator Web Service Connector | A secure connector to interface with the Financial Regulator’s web service for transmitting tax data. |
|  | Identity and Access Management (IAM) | Manages secure user registration, authentication, and authorization. |
|  |  |  |
|  |  |  |
|  |  |  |

## References

The References section provides a complete list of all the documents referenced elsewhere in this document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Version | Date | Document Name | Published by |
|  | <x.x> | <mm/dd/yyyy> | <Document Name> | <Publishing Organization> |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Documentation Roadmap

This section guides into the Architecture Vision document structure to help find the information of interest.

* [Executive Summary](#_Executive_Summary) speaks about the proposed architecture on the highest level and covers:
  + [Key Decisions](#_Key_Decisions) about the architectural and operational choices made for the solution
  + [Key Risks and Open Issues](#_Key_Risks_and) detected with the analysis of the available context, requirements, and proposed decisions
* [Architectural Drivers](#_Architectural_Drivers) elicits the known essential requirements and constraints for the solution to design which play key role in forming archectural decisions and overall architecture.
  + [Business Case](#_Business_Case) describes the solution from the business standpoint including major features
  + [Service Level Agreement](#_Service_Level_Agreement) defines the important technical constraints and guarantees under which the solution will be serviced to its clients
  + [Use Case Model](#_Use_Case_Model) lists the key Use Cases for the designed solution
  + [Domain Model](#_Domain_Model) shows the key business entities with attributes and relationships between them.
  + [Design Constraints](#_Design_Constraints) include business, resource, technical and other constraints accounted for in the architecture of the solution.
  + [Quality Attribute Scenarios](#_Quality_Attribute_Scenarios) are a set of the testable scenarios clarifying non-functional requirements for the system quality attributes such as performance, maintainability, and others.
* [Solution Architecture](#_Solution_Architecture) defines the proposed architecture as a set of architectural views in the format defined in [Appendix C – How View is Documented](#_Appendix_B_–).
  + [Big Picture](#_Big_Picture) shows the solution architectural context, high level decomposition into components, and followed reference architecture.
  + [Development Technology Stack](#_Development_Technology_Stack) selects the tools, frameworks, libraries, external services and other technologies the solution implementation will rely on.
  + <Insert other sections of level 2>
* [Operation Plan](#_Operation_Plan) defines the environments, structures, toolsets, processes, and activites essential to build and operate the solution implementation.
  + [Transition Phase](#_Transition_Phase) talks about the part of the Operation Plan related to the solution implementation phase, including design and development of solution operation framework.
  + [Operation Phase](#_Operation_Phase) defines the processes to operate the solution in production using the framework built at the Transition Phase.
* [Implementation Roadmap](#_Implementation_Roadmap) proposes the plan for solution implementation including:
  + [Implementation Deliverables](#_Implementation_Deliverables) expected to be implemented and delivered at the implementation phase.
  + [Implementation Milestones](#_Implementation_Milestones) tied to the project timeline
  + [Estimate](#_Estimate_1) of complexity/size, effort, schedules for the implementation
  + [Team](#_Team) skillset and structure based on the technology, competence, and schedule requirements
* [Appendix A – Cross-Reference](#_Appendix_A_–_1) defines mappings between views, stakeholders, drivers, elements, etc.
* [Appendix B – Architecture Design Methodology](#_Appendix_A_–) briefly describes the used architecture design methodology.
* [Appendix C – How View is Documented](#_Appendix_B_–) explains the semantics and structure of an architectural view documentation
* [Appendix D – Estimation Methodology](#_Appendix_D_–) briefly describes the used estimation methodology.

# Executive Summary

The section Executive Summary highlights key architectural decisions made for the solution described in [Business Case](#_Business_Case). These decisions are defined and discussed in depth in [Solution Architecture](#_Solution_Architecture) while [Implementation Roadmap](#_Implementation_Roadmap) lays out the proposed milestones, estimates, and team to implement the provided architecture vision.

This section also summarizes the key business and technical risks related to the solution implementation. These risks are uncovered in depth in the rest of the document.

## Key Decisions

The section outlines key design decisions about the solution including the architecture big picture and most essential technologies and external services to rely on.

The solution will be implemented as a cloud based Java web service and deployed to the Cloud Platform Provider XYZ using it’s A, B, C services. The structured data will be stored in the scalable cloud RDBMS storage provided by the Cloud Platform Provider with multy-AZ replication to ensure data availability and backup as required by the solution’s SLA. The solution will be integrated as a REST-ful client with the third-party service ABC to store and retrieve the unstructured blob data.

## Key Risks and Open Issues

The section lists the key risks related to the solution design and implementation. It also lists key open issues where architectural decisions have not been made yet or are likely to change.

|  |  |
| --- | --- |
| Risk Description | Mitigation Strategy |
| Unauthorized data access | Employ robust IAM (Identity and Access Management) solutions and regular security audits. |
| Data leakage | Encrypt all sensitive information, both in transit and at rest. Use secure protocols for all data transmissions. |
| Downtime during high traffic (e.g., tax season) | Implement auto-scaling, load balancing, and have a fault-tolerance strategy, perhaps through the use of AWS SQS. |
| Slow response time during recalculation | Use caching solutions like Redis and optimize the tax calculation engine for performance. |
| Inaccurate tax calculations | Extensive unit testing and QA. Regular updates to the rule engine based on new or updated tax laws. |
| Complexity in updating tax rules | Use a well-documented rule engine like Drools, and provide sufficient training to administrators. |
| Data corruption | Implement ACID transactions in the database layer. Regular backups and integrity checks. |
| Inadequate audit trail | Integrate comprehensive logging and monitoring solutions like ELK Stack or AWS CloudWatch. |
| Vendor lock-in | Use containerization and microservices to ensure that services can be moved to another provider if necessary. |
| Data loss due to system failures | Regular data backups and implement disaster recovery strategies. |

# Architectural Drivers

The section captures significant requirements driving the solution architecture and road map. The requirements which are not influencing the solution architecture in major ways and low level requirement details and scenarios are typically excluded from this section and can be found in the requirement specification or the product backlogs.

## Business Case

The section lays out the business case for the solution Tax Calculation.



Figure 1. Business level view of Tax Calculation

The envisioned solution will enable the users to manage their payment transactions from their mobile devices whily allowing access to the rich reporting from the desktop browsers and backing up their transaction logs in the form of the pdf files to their accounts open with the third-party cloud storage services such as Dropbox for later access. The solution will be deployed on the Amazon cloud as SaaS web application.

### Business Goals

The section enumerates essential business goals for the solution

|  |  |
| --- | --- |
| # | Description |
| BG-1 | Modernize the company’s current approach of interfacing with our existing users |
| BG-2 | Achieve competitive advantage in TCO by moving to the Cloud based SaaS model |
| BG-3 | Enable access to the system from wider range of devices to target larger user base |
| <goal id> | <business goal description> |

### Major Features

The section enumerates solution major features.

|  |  |
| --- | --- |
| # | Description |
| F-1 | Centralized transaction data storage and management on the cloud |
| F-2 | Seamless integration with third-party cloud storage providers |
| F-3 | Access from the mobile devices and desktop browsers without loss of quality in user experience |
| F-4 | Secure data access, transmission, and storage protected from the unauthorized access |
| F-5 | Service uptime not less than 99.9% |
| <feature id> | <feature description> |

## Service Level Agreement

The SLA section provides key parts of the Service Level Agreement applicable to the designed solution ato be supported by the proposed solution architecture.

### Scope and Applicability

This Service Level Agreement (“SLA”) establishes the service quality parameters that are to be applied to the use of Tax Calculation services, and is part of the Tax Calculation Services Customer Agreement.

The terms and conditions specified in this agreement apply solely to the services provided as callable API hosted by the company, herein called “Covered Services.” This SLA applies separately to each service consumer (“Consumer”) that is using the Covered Services. Internal Customer reserves the right to change the terms of this SLA in accordance with the Tax Calculation Services Customer Agreement at any time.

### Service Quality Guarantees

The Covered Services will be operational and available to Consumers at least 99.95% of the time in any calendar month. If Internal Customer does not meet this SLA requirement while the Consumer succeeds in meeting its SLA obligations, the Consumer will be eligible to receive Financial Credits as compensation. This SLA states the Consumer’s exclusive right to compensation for any failure on Internal Customer’s part to fulfill the SLA requirements.

### Definitions

The following definitions are to be applied to Internal Customer’s SLA:

* “Unavailability” is defined as the entirety of the Consumer’s running instances as having no external connectivity for a duration that is at least five consecutive minutes in length, during which the Consumer is unable to launch commands against the remote API through either the Web application or Web service API.
* “Downtime Period” is defined as a period of five or more consecutive minutes of the service remaining in a state of Unavailability. Periods of “Intermittent Downtime” that are less than five minutes long do not count towards Downtime Periods.

## Use Case Model

### Use Case View <View Name>

#### View Context

The interaction outlined in the feature [F-3](#Feature_F_3) happens between the user and the user facing we application

#### Representation



Figure 2. <Use Case View Name>

This view defines primary Use Cases for the feature [F-3](#Feature_F_3), corresponding to the types of interaction between the user and the web application supported by the solution.

#### Element Catalog

|  |  |  |
| --- | --- | --- |
| # | Name | Description |
| ACT-1 | User | A business user of the system |
| UC-1 | Use Case1 | The user logs into the system. |
| UC-2 | Use Case2 | The user pays with the registered credit card. |
| <id> | <element name> | <Use Case description or scenario> |

## Domain Model

The section captures the solution domain model parts essential for the solution architecture including entities, their attributes, and the relationships between them. These elements are captured as part of the solution requirement analysis and do not directly map to the modules, class files, or other similar elements on the implementation level.

### Domain View <View Name>

#### View Context

User facing functionality involves the entities and relationships captured by this view.

#### Representation



Figure 3. <Domain View Name>

This view addresses the primary entities participating in interaction defined by the Use Cases [UC-1](#UseCase_UC_1) and [UC-2](#UseCase_UC_2). The important concern of is-a relationship between Class1 and Class3 is addressed with the logical extension on the view.

#### Element Catalog

The catalog defines the business entities and relationships included in the representation of this Domain View.

|  |  |
| --- | --- |
| # | Description |
| Class1 | Responsible for a, b, c |
| Class2 | Responsible for a, b, c |
| Class3 | Responsible for a, b, c |
| <entity name> | <entity description> |

## Design Constraints

The section lists the constraints accounted for in the designed solution. These can be of business, technical, resource, and other types.

|  |  |
| --- | --- |
| # | Description |
|  | A minimum of 1000 simultanious users must be supported |
|  | Time to market must be within one year |
|  | Java+RDBMS on Amazon technology stack must be used |
|  | The existing client’s dev team skill set is based on Spring framework and JPA |
| <constraint id> | <constraint description> |

## Quality Attribute Scenarios

A Quality Attribute Scenario is an unambiguous and testable requirement for one or more Solution Quality Attributes such as Performance, Usability, Maintainability and others. The scenario consists of six parts: Source of Stimulus, Stimulus, Environment, Artifact, Response, testable and accurate Response Measure.

This section lists and prioritizes the scenarios pertinent to the designed solution.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Quality Attribute | Scenario | Business Priority | Related  To |
|  | Security  (credentials transport security) | At all times the credentials entered by the user during log-in are transferred to the server over encrypted, secure channel without the chance of sniffing by third party. | High | [UC-1](#UseCase_UC_1) |
|  | Usability  (easiness of payment) | When logged in and navigated to the payment page it takes the user up to 3 clicks to pay with the preregistered valid credit card | Medium | [UC-2](#UseCase_UC_2) |
| <scenario id> | <attribute name  (scenario meaning)> | <quality attribute scenario description> | <priority level: High, Medium, Low> | <use case, feature, constraint ids> |

Or alternative notation:

**Scenario:** QA-1

**Quality Attributes:** Security

**Business Priority:** High

**Related To:** [UC-1](#UseCase_UC_1)

**Description:** At all times the credentials entered by the user during log-in are transferred to the server over encrypted, secure channel without the chance of sniffing by third party.

**Environment:** Normal operation conditions

Source

Login Form

Artifact

Client-Server Channel

Response Measure

Traffic encrypted and protected from sniffing

Stimulus

Response

Enter and submit credentials

Credentials transferred to server

**Environment**: The system is online and accessible from outside

# Solution Architecture

The section Solution Architecture is primary for the Architecture Vision document. It defines and reasons about the solution architecture design based on the architecturally significant requirements and constraints identified in the section [Architectural Drivers](#_Architectural_Drivers).

## Big Picture

The section includes a list of architectural views covering the designed solution along with the context it runs in on the high level.

### Solution Context

#### Intent

The view defines the primary solution components collaborating with the external systems and services. It is driven by the [Business Case](#_Business_Case).

#### Context

Context of the decision as diagram or text

#### Representation



Figure 4. Solution Context

Diagram and text documenting decision

#### Element Catalog

Table of annotated elements

|  |  |
| --- | --- |
| Name | Description |
| Cloud Based Solution | Responsible for implementing the REST API to serve data on request from the mobile and web clients |
| Element2 | Responsible for a, b, c |
| Element3 | Responsible for a, b, c |
| <element name> | <element description and responsibilities> |

#### Behavior

Behavior diagrams and scenarios

A screenshot of a computer

Description automatically generated

#### Variability

Points of decision’s configuration, customization, etc.

#### Reasoning

Reasoning

### Solution Decomposition

#### Intent

The view defines the runtime decomposition of the server-side part of the solution. It is driven by the [Business Case](#_Business_Case) and the architecture best practices applicable to the cloud-based applications.

#### Context

The view context is defined by the view [Solution Context](#_Solution_Context) where this section represents decomposition of [Cloud Based Solution component](#Element_Cloud_Based_Solution).

#### Representation

A screenshot of a computer

Description automatically generated

Figure 5. Cloud Solution Decomposition

The diagram presents a decomposed view of a tax calculation and submission system, comprised of several key components. The Web Interface includes a User Interface for tax form interaction and an Admin Dashboard for rule management. Various microservices, such as the Authentication & Authorization Service, Tax Form Service, Tax Calculation Engine, Rule Engine, and Audit Log Service, handle specialized functions like user authentication, form management, tax calculations, and logging. These microservices interact with a centralized Database for data storage and retrieval. Additionally, the Tax Form Service communicates with an external Financial Regulator Web Service for submitting finalized tax data. The components are designed for modular interaction, enabling flexible rule-based form generation and tax calculations, while also ensuring secure data handling and storage.

#### Element Catalog

Table of annotated elements

|  |  |
| --- | --- |
| # | Description |
| User Interface (UI) | Responsible for rendering tax forms, displaying quick calculations like Forecasted Refund, and capturing user input. |
| Admin Dashboard | Provides an interface for administrative tasks such as rule management and direct database access for advanced functions. |
| Authentication & Authorization Service | Manages user login and authorization procedures, verifying user credentials against the database. |
| Tax Form Service | Manages the generation, fetching, and storing of tax forms; uses rules from the Rule Engine and submits data to external Financial Regulator Web Service. |
| Tax Calculation Engine | Performs quick and final tax calculations based on user input and rule sets provided by the Rule Engine; may store certain calculation data in the database. |
| Rule Engine | Enables addition, modification, and fetching of tax rules; acts as the back-end logic provider for tax forms and calculations. |
| Audit Log Service | Records crucial events and transactions, storing logs in the database for future reference and compliance. |
| Database | Serves as the central repository for storing user credentials, tax forms, rules, and audit logs. |
| Financial Regulator Web Service | External service to which finalized tax data is submitted by the Tax Form Service. |

#### Behavior

Behavior diagrams and scenarios

A screenshot of a computer

Description automatically generated

#### Variability

Points of decision’s configuration, customization, etc.

1. **Rule Engine Configuration**:
   * **Decision**: Ability to update tax rules without requiring a system reboot.
   * **Reasoning**: Compliance with frequently changing tax laws.
2. **UI Form Elements**:
   * **Decision**: Dynamic rendering of UI form elements based on rules.
   * **Reasoning**: To adapt to additional questions triggered by certain responses.
3. **Scalability of Microservices**:
   * **Decision**: The system can scale individual microservices independently.
   * **Reasoning**: To handle peak times during tax season.
4. **Database Encryption**:
   * **Decision**: Choice of encryption methods for stored financial data.
   * **Reasoning**: To comply with financial data protection regulations.
5. **Authentication Methods**:
   * **Decision**: Choice between basic authentication, OAuth, or multi-factor authentication.
   * **Reasoning**: Security concerns and user convenience.
6. **Data Backup and Recovery**:
   * **Decision**: Configurable backup intervals and recovery methods.
   * **Reasoning**: To minimize data loss in the event of a failure.
7. **Audit Logging Levels**:
   * **Decision**: Granularity of audit logs.
   * **Reasoning**: Compliance and debugging requirements.
8. **Response Caching**:
   * **Decision**: Whether to use caching for quick calculations and form rendering.
   * **Reasoning**: Performance optimization.

#### Reasoning

1. **Rule Engine Configuration**:
   * **Justification**: Changing tax rules should not require a system rebuild or downtime, thus promoting system availability.
2. **UI Form Elements**:
   * **Justification**: Dynamic forms allow the system to adapt to legislative changes or special circumstances without requiring major code changes.
3. **Scalability of Microservices**:
   * **Justification**: Independent scalability allows for resource optimization and better handling of load.
4. **Database Encryption**:
   * **Justification**: Encryption is non-negotiable for compliance with regulations like GDPR and for building user trust.
5. **Authentication Methods**:
   * **Justification**: Multiple authentication methods give a balance between security and user convenience.
6. **Data Backup and Recovery**:
   * **Justification**: Regular backups minimize the risk and impact of data loss, ensuring data reliability.
7. **Audit Logging Levels**:
   * **Justification**: Different levels of logging help in debugging while complying with data storage regulations.
8. **Response Caching**:
   * **Justification**: Caching frequently requested data can significantly improve system responsiveness, especially under high load.

### Layered Application Structure

#### Intent

The view defines the runtime decomposition of the server-side part of the solution. It is driven by the [Business Case](#_Business_Case) and the architecture best practices applicable to the cloud-based applications.

#### Context

The view context is defined by the view [Solution Context](#_Solution_Context) where this section represents decomposition of [Cloud Based Solution component](#Element_Cloud_Based_Solution).

#### Representation

A screenshot of a computer

Description automatically generated

Figure 6. Layered Application Structure

The Layered Application Structure diagram illustrates the architecture of the tax calculation system, broken down into four primary layers.

1. **Presentation Layer**: This is the front-end layer consisting of the User Interface (UI) and Admin Dashboard. It's responsible for rendering the forms and other UI elements for the end-users and administrators.
2. **Business Logic Layer**: This layer contains the core services responsible for executing the application's business logic. These services include Authentication & Authorization, Tax Form Service, Tax Calculation Engine, and Rule Engine. They handle tasks like user verification, form rendering, tax calculation, and rule management.
3. **Data Access Layer**: This encapsulates the Database component, which is accessed by services in the Business Logic Layer for data storage and retrieval. It acts as a bridge between the business logic and data storage, ensuring that data queries are managed in a centralized location.
4. **External Services**: This layer represents any external systems that the application interacts with, such as the Financial Regulator Web Service for submitting tax forms.

The arrows indicate the flow of interaction between these components. For example, the User Interface communicates with the Authentication & Authorization Service for user verification. Each layer is modular and abstracted, allowing for easier maintainability and scalability.

#### Element Catalog

Table of annotated elements

|  |  |
| --- | --- |
| # | Description |
| User Interface (UI) | Responsible for rendering the tax forms and other UI elements to the end-users, capturing user input. |
| Admin Dashboard | Provides an interface for administrators to manage tax rules and other configurable aspects of the system. |
| Authentication & Authorization Service | Handles user verification, session management, and authorization of roles for accessing different parts of the system. |
| Tax Form Service | Manages the rendering, saving, and submission of tax forms based on user inputs and tax rules. |
| Tax Calculation Engine | Executes the core tax calculations based on the tax rules and user-provided information. |
| Rule Engine | Manages the dynamic tax rules, and allows for rule modification without requiring a system rebuild. |
| Database | Responsible for storing user data, tax forms, and tax rules; accessed by the Business Logic Layer for data queries. |
| Financial Regulator Web Service | External service for confirming and processing the submitted tax forms. |

#### Behavior

Behavior diagrams and scenarios

A diagram with many arrows

Description automatically generated with medium confidence

#### Variability

Tax Rules in Rule Engine

Decision: The Rule Engine allows administrators to update tax rules dynamically.

Configuration: Via Admin Dashboard.

Customization: Allows adding, deleting, or modifying tax rules and related logic.

User Interface (UI) Components

Decision: The UI is designed to be modular, supporting customizable components.

Configuration: Managed via front-end frameworks or UI libraries.

Customization: Change form layouts, add or remove form fields, and toggle visibility of certain UI elements.

Authentication Methods

Decision: The architecture supports multiple forms of authentication.

Configuration: Via Auth Service settings.

Customization: Allows the addition of other authentication methods like social login, 2FA, etc.

Tax Calculation Algorithms

Decision: The Tax Calculation Engine can implement multiple algorithms.

Configuration: Via the Rule Engine or directly in the Tax Calculation Engine.

Customization: Add new calculation methods, algorithms, or formulae as tax laws change.

Database Schema

Decision: Database schema is designed for easy modifications.

Configuration: Managed via database management system (DBMS) or Object-Relational Mapping (ORM).

Customization: New tables, fields, or relationships can be added to accommodate new features or requirements.

External Service Providers

Decision: Architecture allows the integration of different external financial or regulatory services.

Configuration: Via the Business Logic Layer, typically in the Tax Form Service.

Customization: Switch between different service providers or integrate multiple services for redundancy.

Scaling Strategy

Decision: The microservices architecture allows for easy scaling of individual components.

Configuration: Managed via container orchestration systems like Kubernetes.

Customization: Scale up/down services independently based on load requirements.

Data Privacy Features

Decision: The system can adopt various data privacy and encryption mechanisms.

Configuration: Via the Data Access Layer or the Business Logic Layer.

Customization: Implement different data anonymization, encryption, or retention policies.

#### Reasoning

The reasoning behind incorporating these variability points is to make the system more adaptable, maintainable, and extendable. Tax laws and regulations can frequently change, and the system needs to be agile enough to accommodate these updates without requiring a complete overhaul. For instance, the Rule Engine is designed to be dynamic so that administrators can update tax rules in real-time, mitigating the need for software redeployment.

Similarly, modular UI components provide flexibility in adapting to user feedback or changing user interface design trends. Multiple authentication methods and scalable microservices architecture are integrated to meet growing demands and to adapt to evolving security requirements. The Database Schema and External Service Providers are designed for easy modifications to ensure the system remains future-proof.

Lastly, we have incorporated points for scaling and data privacy to meet not just the current requirements but to also prepare the system for future challenges like increased load or stricter privacy laws. In summary, each variability point is strategically planned to provide the system with the adaptability it requires to sustain and grow in a dynamic environment.

## Development Technology Stack

The section includes a list of architectural views covering <List of Concerns, Solution Part, etc.>.

### Development Languages, Frameworks, and Libraries

#### Intent

The view lists the set of programming languages, frameworks, and libraries the solution implementation will depend on.

#### Context

The context is provided by the view [Solution Context](#_Solution_Context).

#### Element Catalog

Table of annotated elements

|  |  |  |
| --- | --- | --- |
| Name | Version | Description |
| Node.js | 14.x | Backend runtime for Tax Form Service and Authentication Service. |
| Python | 3.9.x | Backend language for the Tax Calculation Engine. |
| JavaScript/TypeScript | ES6/4.x | Frontend development language for building the user interface. |
| Express.js | 4.x | Backend framework used for building RESTful APIs and routing in Node.js-based services. |
| React.js | 17.x | Frontend library for building user interfaces. |
| Material UI | 4.x | UI framework for React, responsible for the look and feel of the frontend. |
| RESTful API/GraphQL | N/A | API protocol for communication between services and with the frontend. |
| OpenAPI | 3.x | Specification for RESTful API documentation and external integration points. |
| PostgreSQL | 13.x | Relational database management system for storing user and tax-related data. |
| MongoDB | 4.x | Document-oriented database for storing dynamic or hierarchical data. |
| Passport.js | 0.4.x | Authentication middleware for Node.js, responsible for user authentication. |
| Drools | 7.x | Rule engine library for defining and managing tax rules. |
| OpenSSL | 1.1.x | Encryption library for securing sensitive financial data. |
| Visual Studio Code | Latest | IDE for code development. |
| Git | 2.x | Version control system for tracking changes in code. |
| Docker | 20.x | Containerization tool for packaging and deploying services. |
| Kubernetes | 1.2x.x | Container orchestration for automating deployment, scaling, and management of application. |
| Jenkins | 2.x | CI/CD tool for automating build and deployment pipelines. |
| Jest | 26.x | JavaScript testing framework for unit and integration tests. |
| Pytest | 6.x | Testing framework for Python-based services. |
| Prometheus | 2.x | Monitoring tool for capturing metrics and system performance. |
| Grafana | 7.x | Visualization tool for monitoring data from Prometheus. |
| ELK Stack | 7.x | Logging and analytics suite (Elasticsearch for search, Logstash for processing, Kibana for visualization). |

#### Variability

The technology stack is selected with variability in mind to ensure the system remains flexible and adaptable. For example, the use of containerization technologies like Docker and orchestration tools like Kubernetes allows for scalability and simplifies the process of updating or replacing individual services. Additionally, the stack employs open standards like RESTful APIs and OpenAPI, enabling seamless integration with a variety of external services and making it easier to switch between different service providers if needed. The frontend is built using a modular framework, React.js, which can be easily extended or modified to accommodate evolving user interface requirements. Overall, the technology stack is crafted to be as variable as possible to cater to both current and future needs.

#### Reasoning

The reasoning behind the chosen technology stack and architectural elements is primarily driven by the need for scalability, maintainability, and flexibility. Given the dynamic nature of tax laws, the system incorporates a configurable rule engine, enabling quick updates without downtime. Technologies like Node.js and Python are selected for their ease of use, extensive community support, and proven robustness for web services and computational tasks, respectively. The adoption of containerization and orchestration tools allows for greater operational efficiency and the ability to easily adapt to increasing workloads. Security and data privacy are also key considerations, leading to the use of trusted libraries and protocols for authentication and encryption. Ultimately, the architecture is planned to be as resilient and adaptable as possible to meet both current and future demands.

### Development Tools

#### Intent

The view lists the set of tools the development team will rely upon in solution implementation.

#### Context

The context is provided by the view [Solution Context](#_Solution_Context).

#### Element Catalog

Table of annotated elements

|  |  |  |
| --- | --- | --- |
| Name | Version | Description |
| Visual Studio Code | Latest | IDE for code development, debugging, and source code management. |
| Git | 2.x | Version control system, responsible for tracking changes in the codebase and facilitating team collaboration. |
| GitHub/GitLab | N/A | Code hosting platform, used for repository management, issue tracking, and CI/CD pipelines. |
| Docker | 20.x | Containerization tool, used for packaging applications and dependencies for consistent deployment and scaling. |
| Kubernetes | 1.2x.x | Container orchestration platform, responsible for automating application deployment, scaling, and management. |
| Jenkins | 2.x | Continuous Integration/Continuous Deployment tool, automates the build and deployment process. |
| Postman | Latest | API testing tool, used for developing and testing RESTful APIs. |
| Jest | 26.x | JavaScript testing framework, responsible for running unit and integration tests for the frontend. |
| Pytest | 6.x | Testing framework for Python, used for running unit tests and integration tests for backend services. |
| ESLint | 7.x | JavaScript linter, enforces coding standards and identifies potential errors in the code. |
| Pylint | 2.x | Python linter, enforces coding standards and identifies potential errors in Python code. |
| Webpack | 5.x | Module bundler for JavaScript applications, responsible for optimizing frontend resources. |
| Prometheus | 2.x | Monitoring tool, captures metrics from the system for performance monitoring. |
| Grafana | 7.x | Visualization tool, used for displaying Prometheus metrics and creating dashboards for monitoring. |
| ELK Stack (Elasticsearch, Logstash, Kibana) | 7.x | Logging and analytics suite, responsible for collecting, processing, and visualizing logs. |

#### Variability

The choice of development tools is highly adaptable to ensure that the development process is as agile and flexible as possible. Tools like Visual Studio Code and Git are widely supported and have vast ecosystems, allowing for easy integration of additional extensions or plugins based on project-specific needs. Jenkins provides a robust foundation for CI/CD but can be swapped for alternatives like GitLab CI or GitHub Actions without significant overhead. Similarly, the monitoring and analytics suite (Prometheus, Grafana, ELK Stack) is modular, so individual components can be replaced as needed. The use of containerization through Docker and Kubernetes further adds a layer of flexibility, enabling the team to switch underlying technologies with minimal disruption. Overall, the development toolset is configured to be as variable as needed to adapt to both current tasks and future expansions or changes.

#### Reasoning

The reasoning behind the selection of these development tools revolves around optimizing for scalability, collaboration, and future-proofing. Visual Studio Code and Git were chosen for their universal acceptance and extensive community support, which ensures ease of onboarding for new developers and robustness in code management. The adoption of Docker and Kubernetes aligns with the architectural emphasis on microservices and scalability, allowing for a more streamlined deployment and management process. Jenkins provides a mature, customizable solution for CI/CD, fitting well into the DevOps culture. The combination of Prometheus, Grafana, and the ELK Stack offers a comprehensive monitoring and logging solution, crucial for maintaining high availability and performance. Overall, the toolset is tailored to meet the project's technical demands while providing enough flexibility to adapt to future needs or changes.

### External Integration Points

#### Intent

The view lists the set of programming languages, frameworks, and libraries the solution implementation will depend on.

#### Context

The context is provided by the view [Solution Context](#_Solution_Context).

#### Element Catalog

Table of annotated elements

|  |  |  |
| --- | --- | --- |
| Name | Version | Description |
| Financial Regulator API | v2.0 | API for submitting tax calculations to the financial regulatory authority, ensuring compliance with laws. |
| Payment Gateway API | v3.1 | API for processing payments, responsible for handling refunds and additional payment requirements. |
| OAuth2.0 Identity Provider | Latest | Service for third-party authentication, used for securing user access. |
| CRM System | v5.0 | Customer Relationship Management system, responsible for storing customer information and interaction logs. |
| ERP System | v7.0 | Enterprise Resource Planning system, responsible for financial management and reporting. |
| SMTP Service | N/A | Simple Mail Transfer Protocol service, responsible for sending notifications and alerts to users. |
| SMS Gateway | v1.0 | Service for sending SMS messages for two-factor authentication and alerts. |
| Cloud Storage API | v2.0 | API for storing and retrieving digital documents, like income statements and tax forms. |

#### Variability

The variability of integration points is designed to accommodate evolving business requirements and technological advancements. For example, the Financial Regulator API version could be updated to a newer one as tax rules and compliance standards evolve. Similarly, the Payment Gateway API is modular and can be swapped for another service with minimal disruption if the business requires it. OAuth2.0 provides a framework for integrating multiple identity providers, offering flexibility in user authentication methods. The CRM and ERP systems are also selected with interoperability in mind, enabling a transition to different platforms if needed. This flexible approach ensures that the system can easily adapt to changes or integrate new services without a complete overhaul.

#### Reasoning

The reasoning behind selecting these specific integration points centers on compliance, flexibility, and user experience. The Financial Regulator API is crucial for ensuring that the application meets all regulatory and legal requirements for tax filing. The Payment Gateway API is chosen for its robust security measures and broad feature set, ensuring a smooth payment process. OAuth2.0 is implemented to offer secure and familiar login options to users, improving the overall user experience. The CRM and ERP systems are integral for business operations, enabling better customer relationship management and financial reporting. Finally, SMTP and SMS services are incorporated for effective communication with users, enhancing engagement and security. Each integration point is not only essential for the application's current functionalities but is also flexible enough to accommodate future changes or upgrades.

## Architecture Part X

The section includes a list of architectural views covering <List of Concerns, Solution Part, etc.>.

### Decision View <View Name>

#### Intent

Use Cases, QAS, etc

#### Context

Context of the decision as diagram or text

#### Representation

Diagram and text documenting decision

A diagram of a company

Description automatically generated with medium confidence

#### Element Catalog

Table of annotated elements

|  |  |
| --- | --- |
| # | Description |
| Decision 1: Use of Microservices | Determines the architectural style, specifically whether to use microservices to encapsulate different functionalities like tax calculation, authentication, etc. |
| Decision 2: Language Selection | Selects the programming languages to be used for developing the tax calculation engine and user interface. |
| Decision 3: Data Storage | Specifies the type of database or storage solution that will be used to securely store user data and tax rules. |
| Decision 4: API Integration | Chooses the method for integrating with external services like Financial Regulator API and Payment Gateway. |
| Decision 5: Web Framework | Picks the web framework to build the user interface of the tax calculation tool. |
| Decision 6: Mobile Support | Decides the approach for making the web application mobile-responsive. |
| Decision 7: Dynamic Forms | Determines how to implement dynamic form fields that can change based on previous inputs or tax rules. |
| Decision 8: OAuth2.0 | Specifies the authentication mechanism, particularly the use of OAuth2.0 for secure logins. |
| Decision 9: Two-Factor Authentication | Decides whether to implement an additional layer of security through two-factor authentication. |
| Decision 10: Data Encryption | Chooses the algorithms and methods for encrypting sensitive user data. |
| Decision 11: Regulatory Compliance | Ensures that the system is in compliance with all applicable tax laws and data protection regulations. |
| Decision 12: Load Balancing | Decides the strategy for distributing incoming web traffic across multiple servers. |
| Decision 13: Caching Strategy | Specifies the caching mechanisms for improving system performance. |

#### Interfaces

Element interfaces

#### Behavior

Behavior diagrams and scenarios

#### Variability

Points of decision’s configuration, customization, etc.

#### Reasoning

Reasoning

# Operation Plan

Typically, each service has two key phases of its lifecycle: Transition Phase and Operation Phase.



The state while the solution stays in active development mode is called **Transition Phase**. During this period main activities related to Infrastructure management will be performed by system engineering team, especially by Configuration Managers (CMs). Their goal is to implement all technical solutions to make the product ready to enter the Operation Phase.

Transition Phase ends up with the Production Release.

Operations Team will be responsible for performing the activities during the **Operation Phase**. Key goal at this phase is to maintain the desired quality of service and Service Level Agreement.

## Transition Phase

The section includes a list of architectural views covering <List of Concerns, Solution Part, etc.>.

### Infrastructure

#### Hosting Platform

Text

#### Hardware Resources

Text

#### Virtualized Resources

Text

### Environments

Text

#### Development Environment

Text

#### Continuous Integration Environment

Text

#### Testing Environment

Text

#### Staging Environment

Text

#### Production Environment

Text

### Provisioning

#### Packaging

Text

#### Deployment

Text

#### Undeployment

Text

### Management

#### Application Management

Text

#### Data Management

Text

#### Infrastructure Management

Text

#### System Security Management

Text

#### Backup and Restore

Text

#### Disaster Recovery

Text

#### Incident Management

Text

### Monitoring

#### Availability and Capacity Monitoring

Text

#### Logging and Audit

Text

#### Performance Monitoring

Text

#### Health Monitoring

Text

## Operation Phase

All activities during Operation Phase are performed in cycle.

Some of the Transition Phase tasks can be transferred to Operation Phase if they are considered as not vitally important to start Service Operation Phase.

### Service Operation Support

Text

### CMOD Activities

Text

### Continuous Improvement Process

Text

# Implementation Roadmap

Implementation Roadmap defines the solution implementation road map including list of implementation deliverables, major milestones, effort estimates, and recommended team skillset, structure, and size.

## Implementation Deliverables

The implemented solution will include the following parts as deliverables:

|  |  |  |
| --- | --- | --- |
| Name | Refer to Requirements | Refer to Design |
| Application 1 |  |  |
| Service 2 |  |  |
| Mobile Client 3 |  |  |
| Production Deployment 4 |  |  |

## Implementation Milestones

The section proposes the major milestones to guide the solution implementation.

|  |  |  |
| --- | --- | --- |
| Milestone | Description | Outcomes |
| M0 | Project bootstrap | * Dev Team ramped up * Dev and Testing Environment set up * Project Skeleton built |
| M1 | Main application with web front-end development | * Alpha version of the web application * Deployment to the staging |
| M2 | Development and stabilization of the web application, development of the deployment and monitoring procedures, release | * Release 1.0 of the application prepared and stable * DevOps framework functional * Release 1.0 goes to production |

## Estimate

The Estimate section provides the estimates for the solution implementation based on the proposed architecture and selected estimation methodology.

### Assumptions and Limitations

This section describes the known limitations imposed by selected platform, technology, hardware, operating system, third party components, etc. which might affect implementation effort and schedule estimates.

|  |  |  |
| --- | --- | --- |
| # | Description | Responsible |
| A1 | Assumption 1 | <Client Name> |
| A2 | Assumption 2 | SoftServe Team |
| L1 | Limitation 1 | <Third Party Name> |

### Estimate

The Estimate section provides effort and, optionally, schedule estimates for the implementation based on the architecture proposed in this Vision document and the selected estimation methodology defined in [Appendix C](#_Appendix_C_–).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Item | Complexity | Min Effort | Max Effort |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | **Total:** |  |  |  |

## Team

The Team section outlines the proposed team skillset and structure based on the proposed technology stack and estimates.

### Team Skillset

The section defines skills required from the different team member roles to implement the solution.

|  |  |
| --- | --- |
| Role | Skillset |
| Business Analyst | * Business domain 1 * Business domain 2 |
| Backend Developer | * Programming Language 1 * Technology 2 * Framework 3 * Tool 4 * Standard 5 |
| UI Developer | * Programming Language 1 * Technology 2 * Framework 3 * Tool 4 * Standard 5 |
| DevOps Engineer | * Programming Language 1 * Technology 2 * Framework 3 * Tool 4 * Standard 5 |

### Team Structure

The section proposes a team/sub-team structure and work allocation for the implementation phase.

#### Core Team

The core team is responsible for:

* Development of <Component 1>
* Development of <Component 2>
* Support of <Application 1>

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Responsibility | Count | FTE |
| Project Manager | * Project management (coordinate the team, status reporting, communications with the Client team) * Gap analysis | 1 | 0.5 |
| Business Analyst | * Business analysis * Business requirements specification * Software requirements specification | 1 | 1.0 |
| Solutions Architect | * System analysis and design * System requirements specification | 1 | 0.2 |
| Technical Leader | * Technical leadership and communication * SCRUM Master * Code Reviews * Backend implementation | 1 | 1.0 |
| Sr. Backend Developer | * Technical communication * Code reviews * Backend implementation | 2 | 1.0 |
| Jr. Backend Developer | * Backend implementation | 3 | 1.0 |
| Int. UI Developer | * Web frontend design and implementation * UX prototyping | 3 | 1.0 |

#### DevOps Team

The DevOps team is responsible for:

* Configuration of <Environment 1>
* Configuration of <Environment 2>
* Deployment and runtime monitoring of <Application 1>

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Responsibility | Count | FTE |
| DevOps Architect | * Operations analysis and design | 1 | 0.2 |
| Int. DevOps Engineer | * System deployment * Environment setup * Production operations support | 3 | 1.0 |

# Appendix A – Cross-Reference

The Cross-Reference section provides convenient lookup tables mapping views, drivers, and risks.

## Driver Fulfillment Table

The section maps the architectural drivers elicited in this document to the decision views. When a driver is not addressed by the Vision it is marked as not fulfilled.

|  |  |  |
| --- | --- | --- |
| Driver | Fulfilled? | Decision Views |
| <driver hyperlink> | <Yes/No> | <Links to relevant decision views> |

## View Reference Table

The section lists the documented requirement and architectural views of the solution accompanied with the brief description and the stakeholders who should be interested in using this view at the implementation phase. The views are documented following the architecture documentation guidelines defined in [Appendix B – How View is Documented](#_Appendix_B_–).

|  |  |  |
| --- | --- | --- |
| View Name | Description | Concerned Stakeholders |
| [Use Case View <View Name>](#_Use_Case_View) |  |  |
| [Domain View <View Name>](#_Domain_View_<View) | <view brief description> | <stakeholder roles> |
| [Decision View <View Name>](#_Decision_View_<View) | The view defines this and that architectural decision targeting those and those drivers | Product Manager, Architect, Dev Team |
| [Implementation Milestones](#_Implementation_Milestones) |  |  |
| [Estimate](#_Estimate) |  |  |
| [Team Structure](#_Team_Structure) |  |  |
| <view hyperlink> | <view brief description> | <stakeholder roles> |

## Trade-off Reference Table

The section maps trade-offs, risks, and sensitivity points to the decision views where they are analyzed.

|  |  |  |
| --- | --- | --- |
| Point | Description | Decision Views |
| <point hyperlink> | <type: Risk, Nonrisk, Sensetivity Point, Trade-off and description> | <Links to relevant decision views> |

## Element Catalog

The section maps elements defined in the documentation to the views where they can be found.

|  |  |  |
| --- | --- | --- |
| Driver | Description | Decision Views |
| <element hyperlink> |  | <Links to relevant decision views> |

# Appendix B – Architecture Design Methodology

The architectural design of the solution follows the formal methodology called “Augmented Attribute Driven Design” (Augmented ADD). This methodology is developed by Carnegie-Mellon Software Engineering Institute to standardize on the industry best practices and to build the software architectures robustly and rationally within practicable schedules.

The figure defines Augmented ADD in detail.



Figure A-1. Augmented Attribute Driven Design

There are a few major principles guiding this process:

* The solution architecture design must be driven and rationalized by the explicit set of the functional and business requirements, non-functional qualities of the system (such as performance, availability, usability, maintainability, etc.) defined by means of the Quality Attribute Scenarios, and various design constraints. All these are called Architectural Drivers. The architectural drivers are elicitated, prioritized, and approved by the solution stakeholders.
* The process is iterative. Each iteration starts with selection of the subset of architectural drivers to address and completes with documenting the design decisions made to satisfy those.
* The design decisions are based on selection among design concepts such as Reference Architectures, Patterns, Frameworks, Tools, etc. Selection relies on reasoning how well aparticular candidate concept matches the architectural drivers, what pros, cons, and risks it brings to the table in comparison with other candidates.
* The selected design concepts are instantiated, clarified, and transformed to seamlessly work together within the frame of the solution design and implementation.
* The results of the design process are recorded with the appropriate level of detail in the documentation called “Architecture Vision” which serves as an architectural entry point into the solution implementation.

Architectural Driver Elicitation is one of the most critical parts in preparation and execution of this process. While designed product features, functional requirements, and various business, resource, and other constraints are typically well understood by the stakeholders the non-functional requirements concerning System Quality Attributes such as performance, capacity, resource utilization, reusability of the specific components, maintainability of the solution implementation, its deployability and others can remain more obscure. At the same time these Quality Attributes drive much of the design decisions about the future solution architecture.

To improve the process of the quality attribute related requirements elicitation the Quality Attribute Workshop (QAW) methodology is used. In essence, QAW is the process of collaborative generation and prioritization of the scenarios defining how well the solution or its parts are expected to work under certain conditions executed by the architects along with the group of the solution stakeholders. Each scenario must be specific enough and provide a testable measure of response by the system.

# Appendix C – How View is Documented

## View Structure

A view is the main unit of the Architecture Vision . Its purpose is to document, explain, and rationalize about a particular design decision (or multiple related design decisions). In its simplified form the view is also used to document a set of requirements in the form of Use Case and Domain Model diagrams.

The view typically includes these seven parts, some of them can be omitted:

Intent

The information about the purpose of this view and the list of the architectural drivers addressed by this view provide the Intent of the view. The architectural driver references link to those defined in the [Architectural Drivers](#_Architectural_Drivers) section and can include features, use cases, quality attribute scenarios, constraints, etc. This section can also directly describe the problem(s) to solve with the view.

Context

The context of the view means the larger part of the designed solution within which this view exists. For instance, the view can define the decomposition of the application source code into layers. In this case the context of this view will be the solution itself or its subsystem including the application as a black box. This section would often link to another view defining this context or in other cases could directly embed the diagram and textual description of the context.

Representation

The Representation defines and depicts the design decisions made to address the specified intent in the specified context. It usually includes a diagram drawn with UML or another graphical notation and textual description of the design decisions made. In some cases a table or some other form of representation can be used instead of or in addition to the the diagram.

Element Catalog

The Element Catalog lists the elements defined in the view defining their types, responsibilities, semantics, constraints, and other important traits. When needed the relationships, operations, and attributes can be defined as well. When an element is defined on more than one view there is no need to duplicate its definition. A link to its original definition can be used instead.

Interfaces

The section defines important element interfaces and integration points in as much detail as needed to enforce the clear contracts between elements and subsystems. These interfaces are typical candidates for inclusion:

* REST API
* SOAP Web Services via linked or embedded WSDL and XSD
* Message Broker queue and exchange names to bind
* Message formats for XML, JSON, CSV, and other types of messages
* RPC
* Important code module (classes, java interfaces) interfaces

The documented interfaces can be provided and/or required.

This is an example what the documentation for a REST resource might look like.

|  |  |
| --- | --- |
| Title | Start indexing documents. Indexing will be performed asynchronously. Use status call to get status of the index |
| URL | /documents/index |
| Method | POST |
| Request Body | {“startDate”: “2012-01-30”, “endDate”: “2013-01-30” }  Also you can miss date parameters and just use: {} |
| Success Response | Response Headers:  Status Code: 200 OK |
| Error Response | Code: 401 UNAUTHORIZED |
| Sample Call | http://host/documents/index |

These parts pertinent to the REST web service definition cen be included into the resource documentation:

* Resource path sans the base path and version (<http://host/application/v1>)
* HTTP method
* Headers
* Cookies (optional)
* Query (optional)
* JSON passed in the request
* Response code. For more details, please refer to [section 6 of RFC 2616](http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1)
* Response error codes
* JSON passed back in the response (optional)

Variability

The Variability section documents possible ways to configure, customize, and substitute elements and relationships which are parts of this view in response to potential requirement and external dependency changes as well as in the interest of deployment, integration, and other operations on the solution. Typical variability candidates:

* Configuration options in files, data stores, etc.
* Scripting
* Substitution of components, classes, third-party services
* Using alternative frameworks, libraries, servers, pieces of infrastructure, and other technologies.
* Limits within which the resource amounts can vary to provide for the system requirements

Reasoning

The Reasoning section rationalizes about the decisions taken and described in the view. It can also document the alternative decisions and compare them to the selected ones to show why they were discarded.

The section points out the sensitivity points, risks, trade-offs, unmade decisions and more.

Typical approaches to Reasoning section completion:

* Alternative options analysis
* Trade-off Analysis
* References to authoritative sources
* Calculations
* Cost-Benefit Analysis
* Reference to Proofs of Concept, Prototypes, Simulations
* Reference to previous experience

**Alternative options analysis** can be presented in the form:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Alternatives | Driver 1 | Driver 2 | Driver 3 | Reason for discarding |
| Alternative1 | ++ | - | + | Accepted |
| Alternative2 | + | o | o | Reasons a, b, c |
| Alternative3 | **--** | **+** | **o** | Reasons x, y, z |
| <alternative> |  |  |  | <reasons why alternative is discarded> |

Where:

* ++: a very positive influence on the quality goal can be expected
* +: a positive influence on the quality goal can be expected
* o: no influence on the quality goal can be expected
* -: a negative influence on the quality goal can be expected
* --: a very negative influence on the quality goal can be expected

**Trade-off analysis** is another way to analyze the decisions for risks and trade-offs. Typically a form like the one below is used to capture its results in the reasoning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scenario #: Number | Text of quality attribute scenario | | | |
| Architectural Decisions | Sensitivity | Tradeoff | Risk | Nonrisk |
| Architectural decisions relevant to this scenario that affect quality attribute response | Sensitivity Point # | Tradeoff Point # | Risk # | Nonrisk # |
| ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... |
| Reasoning | Qualitative and/or quantitative rationale for why the list of architectural decisions contribute to meeting each quality attribute requirement expressed by the scenario | | | |

The method is described in depth in this document:



Other types of reasoning are expected to come in the form suited to a specific case.

## Graphical Notation

The section defines the graphical and color notation used in the document to draw the diagrams in the views.

This Vision primarily uses **UML 2.4.1** documented at [www.uml.org](http://www.uml.org/) as the graphical modeling language of choice.

These online resources can be used to look up the common ways of UML modeling:

* [UML 2 Tutorial](http://www.sparxsystems.com/resources/uml2_tutorial/index.html) from Sparx Systems
* [www.uml-diagrams.org](http://www.uml-diagrams.org/)

Additionally, these graphical elements are used through the documentation:

|  |  |
| --- | --- |
| Graphics | Meaning |
|  | Mobile Device with the client application installed |
|  | Third-party components are marked with blue color |
| <image> | <image meaning> |

# Appendix D – Estimation Methodology

This section refers to the estimation methodology used to produce the implementation estimates.

The approach to the estimation in the phase of initial solution architecture generation involves several steps:

1. Decompose the solution into sufficiently small and well-defined parts.
2. Estimate their size and/or relative complexity by analogy and involving, when possible, other experts.
3. Translate the obtained estimates into the effort estimates for an “average” developer of a specified level of knowledge and experience. This translation accounts for the company historical log of similar implementations.
4. Adjust the estimates for the identified key risks which can potentially affect the required effort or schedule.
5. Based on the project business priorities and the estimated effort
   1. either plan the Milestone delivery schedule (usually in project iterations or months) and deduce the proposed team size and structure from it
   2. or define the team size and structure and plan the schedule based on the amount of effort the team can cover over an iteration.
6. Project and approve with the stakeholders the iterative implementation process and delivery based on the above estimates.
   * <Amend or replace the methodology description if the different one was used>