CPSA-A Abschlussaufgabe

BigSpender

ChatGPT, Ralf D. Müller

Version 0.1, 11.11.2023

Inhalt

Liste der Abbildungen

Liste der Tabellen

Chapter 1. Introduction

This document describes the solution for the BigSpender iSAQB example certification for the advanced level. It was mainly created by ChatGPT.

Sources for the certification task where taken from the iSAQB website: isaqb.org

The ChatGPT model used was GPT-4 with a context size of 4096 tokens. A model with a larger context size (128.000 tokens) was available through the API but not in the chat application.

Chapter 2. BigSpender System Architecture - Quality Requirements

2.1. Subtask 1 – Quality requirements

The objective is to derive quality scenarios from the case study and allocate content-related importance to them. We create a utility tree showing the connection with the relevant business features. We then filter out the most important quality goals and give these a weighting. The expected results are: - A table with 3-5 quality goals, including the quality feature for each goal, a brief motivation, and the weighting of the goal. - A formulated quality scenario per quality goal, each with business importance. - A utility tree as a graphic.

This approach ensures that the most critical aspects of system quality are identified, prioritized, and addressed in the architectural design. The quality scenarios are formulated to align with the business's needs and are weighted to reflect their relative importance to the company's operations.

2.2. Key Quality Goals

ID	Quality Goal	Weight
Q1	Data Integrity and Compliance	40%
Data integrity is essential for accurate audit trails and adherence to German data protection laws, ensuring that financial data is processed and reported correctly.		
Q2	System Performance and Scalability	30%
Performance impacts user satisfaction and productivity, and the system must support many users concurrently, which is vital for BIG PLC's operations.		
Q3	Security of Personal Data	20%
Security is a high priority due to strict data protection laws in Germany. Protecting personal data maintains user trust and ensures legal compliance.		
Q5	System Reliability and Availability	10%
System availability is crucial for continuous business operations, minimizing downtime, and ensuring access for users whenever needed.		

2.3. Quality Scenarios

ID	Quality Scenario	Importance
Q1	To maintain a high level of data integrity, the system must ensure that expense entries cannot be altered by unauthorized persons.	High

ID	Quality Scenario	Importance
Q2	The system should handle up to 10,000 users concurrently without performance degradation.	High
Q3	Personal data must be encrypted in transit and at rest to protect against unauthorized access.	High
Q4	The user interface must allow users to submit expenses in no more than three clicks from the dashboard.	Medium
Q5	The system's annual downtime should not exceed 4 hours to ensure continuous business operations.	High

2.4. Quality Features Influenced by Scenarios

Quality Feature	Influenced by Quality Scenario
Performance	Q2 - Handling up to 10,000 users concurrently.
Security	Q1 - Protecting data integrity.
	Q3 - Encrypting personal data.
Usability	Q4 - Intuitive UI with minimal interaction.
Reliability	Q5 - Limited annual downtime.

2.5. Utility Tree

```
Failed to generate image: undefined method `to_sym' for nil:NilClass
@startmindmap
+ [BigSpender Utility Tree]
++ [Performance]
+++ [Q2 - Handle 10,000 concurrent users]
++ [Security]
+++ [Q1 - Protect data integrity]
+++ [Q3 - Encrypt personal data]
++ [Usability]
+++ [Q4 - Intuitive UI with <=3 clicks]
++ [Reliability]
+++ [Q5 - Max 4 hours downtime annually]
@endmindmap
```

Chapter 3. BigSpender System Architecture - Solution Strategy

3.1. Subtask 2 – Solution strategy

The goal is to outline a solution strategy for the BigSpender system, communicating key constraints, assumptions, and basic principles guiding the design decisions. This is crucial for aligning stakeholders on the architectural vision and approach. The strategy has been formulated considering the provided clarifications and aligning with the quality requirements.

3.2. Solution Strategy

3.2.1. Key Constraints

- **Technology Stack**: Preference for Java or Groovy-based technologies, with a focus on Spring Boot or Grails.
- **Deployment Environment**: The system will be deployed in a cloud environment.
- **Security Protocol**: All connections must be secured using TLS.

3.2.2. Assumptions

- **Integration Points**: Assumed integration with payroll, user directory, and other external systems via secure APIs.
- **Scalability**: Based on the requirement to support up to 10,000 concurrent users, scalable architecture is assumed to be a priority.
- **Modulith Architecture**: The system will adopt a modular monolith (modulith) approach, aligning with the quality requirements.

3.2.3. Basic Principles

- **Modularity**: The system will be designed with clear module boundaries, promoting maintainability and ease of future scaling or modifications.
- **Security**: Adhering to the principle of secure by design, security will be integrated into the architecture from the outset, particularly focusing on data protection and compliance with German law.
- **Performance and Scalability**: Ensuring responsiveness and stability under high user load.
- **Usability**: Focused on providing an intuitive user experience, with simplicity in

navigation and interaction.

This solution strategy has been crafted to ensure that the BigSpender system is robust, secure, and scalable, adhering to the specified technological preferences and deployment environment. The modular approach aligns with the modulith architecture, ensuring ease of maintenance and future enhancements. Security and usability have been prioritized in line with the stated quality goals and business requirements.

Chapter 4. BigSpender System Architecture - Technical Context

4.1. Subtask 3 – Technical context

Create a technical context for the BigSpender system, identifying all users, devices, and external systems, including those technically motivated and missing in the business context view. Decide which parts to develop in-house and which to buy or use as open source, distinguishing these parts accordingly. Additionally, state the communication mechanism for each connection between technical nodes.

This solution strategy provides a comprehensive overview of the technical context for the BigSpender system. It identifies all actors and external systems that interact with BigSpender, and the communication protocols used. This approach ensures that the technical architecture aligns with the business needs and quality requirements of the system.

4.2. Context Diagram

```
Failed to generate image: undefined method `to_sym' for nil:NilClass

@startuml BigSpender_Context
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-
PlantUML/master/C4_Container.puml

LAYOUT_WITH_LEGEND()

Person(squanderer, "Squanderer", "Employee or consultant submitting expenses")
Person(receiptScanner, "Receipt Scanner", "Person or system scanning receipts")
Person(approver, "Approver", "Supervisor or manager approving expenses")
Person(businessAdmin, "Business Admin", "Administrator for business rules")
Person(auditor, "Auditor", "Responsible for auditing expense claims")

System_Boundary(bigSpender_sys, "BigSpender System") {
    Container(bigSpender, "BigSpender", "Java/Groovy application", "Manages and
processes expense claims")
}

System_Ext(payroll, "Payroll System", "Manages employee payments")
System_Ext(userDirectory, "User Directory", "Manages user authentication")
System_Ext(documentArchive, "Document Archive", "Stores receipts and documents")
System_Ext(invoicesSystem, "Invoices System", "Manages invoicing")
System_Ext(inTray, "In-tray System", "Manages incoming receipts/documents")

Rel(squanderer, bigSpender, "Uses")
Rel(receiptScanner, bigSpender, "Uses")
Rel(approver, bigSpender, "Uses")
Rel(businessAdmin, bigSpender, "Uses")
Rel(auditor, bigSpender, "Audits")

Rel(bigSpender, payroll, "Integrates via", "HTTPS REST API")
Rel(bigSpender, userDirectory, "Authenticates via", "LDAP over TLS")
Rel(bigSpender, documentArchive, "Accesses", "WebDAV over HTTPS")
Rel(bigSpender, invoicesSystem, "Integrates via", "SOAP over HTTPS")
Rel(bigSpender, inTray, "Receives data via", "HTTPS REST API")

@enduml
```

4.3. External Systems and Protocols

External System	Communication Protocol
Payroll System	HTTPS REST API
User Directory	LDAP over TLS
Document Archive	WebDAV over HTTPS
Invoices System	SOAP over HTTPS

External System	Communication Protocol
In-tray System	HTTPS REST API

Chapter 5. BigSpender System Architecture - Business Structuring

5.1. Subtask 4 – Business structuring

The objective is to develop and visualize a rough business structuring of the BigSpender system. This involves naming the individual system components, defining their responsibilities, and explaining how they interact. This approach provides a clear and modular view of the system, facilitating understanding and communication among stakeholders.

5.2. Component Diagram

```
Failed to generate image: undefined method `to_sym' for nil:NilClass
@startuml BigSpender_Business_Structure
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-
PlantUML/master/C4_Component.puml

LAYOUT_WITH_LEGEND()

Person(squanderer, "Squanderer", "Employee or consultant submitting expenses")
Person(receiptScanner, "Receipt Scanner", "Person or system scanning receipts")
Person(approver, "Approver", "Supervisor or manager approving expenses")
Person(businessAdmin, "Business Admin", "Administrator for business rules")
Person(auditor, "Auditor", "Responsible for auditing expense claims")

System_Boundary(bigSpender_sys, "BigSpender System") {
    Container(expenseManagement, "Expense Management Module", "Java/Groovy",
"Manages and processes expense claims")
    Container(userManagement, "User Management Module", "Java/Groovy", "Handles
user authentication and authorization")
    Container(reporting, "Reporting Module", "Java/Groovy", "Generates expense
reports and analytics")
    Container(integration, "Integration Module", "Java/Groovy", "Integrates with
external systems")
}

System_Ext(payroll, "Payroll System", "Manages employee payments")
System_Ext(userDirectory, "User Directory", "Manages user authentication")
System_Ext(documentArchive, "Document Archive", "Stores receipts and documents")
System_Ext(invoicesSystem, "Invoices System", "Manages invoicing")
System_Ext(inTray, "In-tray System", "Manages incoming receipts/documents")

Rel(squanderer, expenseManagement, "Uses")
Rel(receiptScanner, expenseManagement, "Uses")
Rel(approver, expenseManagement, "Uses")
Rel(businessAdmin, userManagement, "Uses")
Rel(auditor, reporting, "Audits")

Rel(expenseManagement, payroll, "Integrates via Integration Module")
Rel(userManagement, userDirectory, "Authenticates/authorizes via Integration
Module")
Rel(reporting, documentArchive, "Accesses via Integration Module")
Rel(integration, invoicesSystem, "Integrates")
Rel(integration, inTray, "Integrates")

@enduml
```

5.3. Components and Their Interactions

Component	Responsibility	Interaction
Expense Management Module	Manages and processes expense claims	Interacts with Squanderer, Receipt Scanner, and Approver for managing expenses. Integrates with Payroll System via Integration Module.
User Management Module	Handles user authentication and authorization	Used by Business Admin. Authenticates and authorizes via User Directory.
Reporting Module	Generates expense reports and analytics	Used by Auditor. Accesses data from Document Archive.
Integration Module	Integrates with external systems	Connects Expense Management, User Management, and Reporting Modules with external systems like Payroll, User Directory, Invoices System, and In-tray System.
Payroll System	Manages employee payments	Receives processed expense claims from the Expense Management Module.
User Directory	Manages user authentication	Provides authentication and authorization services to the User Management Module.
Document Archive	Stores receipts and documents	Archives documents and receipts accessed by the Reporting Module.
Invoices System	Manages invoicing	Integrated by the Integration Module for managing invoicing related to expenses.
In-tray System	Manages incoming receipts/documents	Provides receipts and documents to the Expense Management Module.

This business structuring of the BigSpender system provides a comprehensive view of the system components and their roles. The modular approach aligns with the system's quality requirements, ensuring scalability, maintainability, and clear delineation of responsibilities. The inclusion of external systems highlights the system's integration points, essential for a holistic understanding of the system architecture.

Chapter 6. BigSpender System Architecture - Technology Decisions

6.1. Subtask 5 – Technology decisions

Define the technology stack mapping the BigSpender design onto real IT infrastructure, describing programming languages, operating systems, database technologies, and communication mechanisms. Then, explain and rationalize how this technology stack ensures the quality goals identified in subtask 1, focusing on the user interaction of "Submitting an Expense Claim." This will include a tabular step-by-step presentation of the workflow, showing the contribution of system parts towards achieving the quality goals.

6.2. User Interaction: Submitting an Expense Claim

The interaction chosen to demonstrate the technology stack and its alignment with quality goals is "Submitting an Expense Claim."

6.2.1. Technology Stack Overview

- Programming Language: Java with Spring Boot
- Operating System: Linux
- Database Technologies: PostgreSQL and Redis
- Communication Mechanisms: RESTful APIs, HTTPS, and TLS
- Front-End Technologies: React and Bootstrap
- Cloud Infrastructure: AWS
- DevOps Tools: Docker and Jenkins
- Security: Spring Security, OAuth 2.0, JWT
- Monitoring and Logging: ELK Stack

6.2.2. Workflow of Submitting an Expense Claim

What Happens	System Parts Involved	Contribution to Quality Goals
User logs in to submit an expense	React Front-End, Spring Security, User Directory (LDAP over TLS)	Ensures secure access (Security), User-friendly UI (Usability)
User inputs expense details	React Front-End, Bootstrap	Provides an intuitive interface, ensuring data accuracy (Usability)

What Happens	System Parts Involved	Contribution to Quality Goals
Expense data submitted to server	Spring Boot Application, RESTful API over HTTPS	Secure transmission of data (Security), Efficient processing (Performance)
Data validated and processed	Spring Boot Application, Business Logic Layer	Ensures data integrity and compliance with business rules (Data Integrity)
Data stored in PostgreSQL	PostgreSQL Database, Transaction Management	Maintains data integrity and consistency (Data Integrity), Reliable data storage (Reliability)
Cache updated in Redis	Redis	Enhances retrieval speed for frequently accessed data (Performance)
Confirmation sent to user	Spring Boot Application, React Front-End	Provides immediate feedback to user (Usability), Ensures system responsiveness (Performance)
Notification sent to Approver	Spring Boot Application, RabbitMQ	Decouples message delivery (Reliability), Asynchronous communication (Performance)
Data available for auditing	Reporting Module, ELK Stack	Facilitates audit processes (Compliance), Effective logging (Reliability)

This workflow demonstrates the "concert of system parts" used to ensure the quality goals. The technology stack has been selected and configured to address the identified quality goals of Security, Usability, Performance, Data Integrity, and Reliability, as defined in subtask 1.

Chapter 7. BigSpender System Architecture - Evaluation

7.1. Subtask 6 – Evaluation

The task is to identify the top five riskiest and most important quality scenarios from subtask 1, providing a rationale for their selection and discussing how the architecture addresses these scenarios. This includes referring to central decisions in the architecture, submitting any new decisions, countering risks, and highlighting trade-offs. This approach ensures that the architecture is robust against key risks and aligns with critical quality goals.

7.2. Top 5 Riskiest and Most Important Quality Scenarios

1. **Data Integrity and Compliance (Q1)**
    - Rationale: Essential for legal compliance and integrity of financial data. Any breach could lead to legal issues and loss of trust.
    - Architectural Decision: Implementing strict access control and audit trails within the system. Using Spring Security for robust authentication and authorization, ensuring that only authorized users can modify data.
    - Trade-offs: This may slightly increase system complexity but is necessary for ensuring compliance and data integrity.
2. **System Performance and Scalability (Q2)**
    - Rationale: Critical for user experience and system reliability, especially during peak times.
    - Architectural Decision: Using a cloud-based infrastructure (AWS) for scalability. Implementing caching mechanisms with Redis to enhance performance.
    - Trade-offs: Additional cost for cloud infrastructure and complexity in cache management, but essential for maintaining performance.
3. **Security of Personal Data (Q3)**
    - Rationale: Data breaches can have severe legal and reputational consequences.
    - Architectural Decision: Encryption of data in transit and at rest, using HTTPS and TLS for secure communication. Implementation of OAuth 2.0 and JWT for secure handling of authentication and authorization tokens.
    - Trade-offs: These security measures may add latency to the system, but they are non-negotiable for ensuring data security.
4. **System Reliability and Availability (Q5)**

    - Rationale: System downtime can disrupt business operations and damage the company’s reputation.
    - Architectural Decision: Adopting a microservices architecture for fault isolation, using Docker and Jenkins for consistent deployment, and AWS for high availability.
    - Trade-offs: This approach requires more effort in monitoring and management but is crucial for high availability.
5. **Usability (Q4)**
    - Rationale: Essential for user adoption and productivity. Poor usability can lead to inefficiencies.
    - Architectural Decision: Using React and Bootstrap for creating an intuitive and responsive user interface, ensuring ease of use.
    - Trade-offs: May require more upfront development effort for a user-friendly design, but it is critical for user satisfaction and system adoption.

These scenarios have been selected based on their potential impact on the system’s operational effectiveness, legal compliance, user satisfaction, and the business’s reputation. The architectural decisions made in each case are aimed at mitigating the risks associated with these scenarios, ensuring the system is robust, compliant, and user-friendly.