On Dynamic Record Layout

and a naive graph database

Fabian Klopfer
Theodoros Chondrogiannis

Database and Information Systems Group
Department of Computer Science
University of Konstanz

Abstract

Contents

Chapter 1

User Guide

1.1 Intorduction

1.2 Record Layout

1.3 Architecture Overview

1.4 Conclusion

Bibliography

Chapter 2

Specifications

2.1 Appendix A: Roadmap & Milestones

2.2 Appendix B: Software Requirement Specification

2.2.1 Introduction

2.2.1.1 Purpose

The purpose of this Software Requirements Specification is to describe the features, constraints and demands of a Zoo Management Tool in detail. This document is intended for both the stakeholders and the developers of the system and will be proposed to the zoo director Dr. Susan Seuss and her Staff.

2.2.1.2 Scope

This Software system shall implement a Zoo Management Tool, that assists the zoo-keepers in ordering feed, managing and visualizing the feeding costs of each animal , the administration in managing the budgets, preferenced food dealers, communication, staff and working hours and the system itself.

2.2.1.3 Overview

In the second chapter several conditions, assumptions and circumstances will be mentioned, that help charachterizing the software's special use case. In the thrid chapter the concrete requirements are listed.

2.2.2 Overall description

2.2.2.1 Product Perspective

The product shall support three main tasks of the daily work in the zoo: 1. Budget management, 2. Feed management and 3. Scheduling management. This shall be done by a database, that shall save related information and functions and methods, that operates on this data.

2.2.2.2 Product Functions

The system shall store information on the budget and take care of balancing each month. It shall support the zoo staff in communicating request for additional funds, by providing forms, notifications and historical information on the requests. The system shall store information on available feeds, their expiration date, orders and invoices. It shall also store information about employees and schedules of holidays and working hours.

2.2.2.3 User Characteristics

The users are the staff of a zoo. Therefore standard user will not have advanced technical knowldege, but they are able to visit websites to order food and to read from the graphic user interface of a tablebased database.
There are three different type of users:

- Zoo keepers: responsible for animals and their feed, including the ordering of new feed. They have the right to view animal budgets, they have the right to view and alter the feed information. They don't have the right to view employee information or other budget information than animal budget information.

- Secretary: Cares about scheduling working hours, vacation, the employee budget, delivery issues and probably other occuring problems. Therefore she needs access on Employee information and the employee budget, orders, invoices, and eventually more. So there shall be a usergroup, that is easy adjustable, so that the secretary can be granted temporary rights for things she only need to handle once or to change her permanent access and alter rights quickly. This usergroup needs to be more secure.

- Zoo director: The administrator of the zoo and the future system. Needs full access and altering rights; has to be most secure.

2.2.2.4 Constraints

- Security critical accounts for usergroups 2 and 3, due to personal and financial critical data.

2.2.2.5 Assumptions and Dependencies

The tablets for the software to be run on are present and provide an operating system, that implements a messaging system and is compatible to the C programming language (and by that C based languages).

2.2.3 Specific Requirements

2.2.3.1 External interfaces

Tablets and their os, a fitting database management system (details omitted, no implementation/concrete facts about this)

2.2.4 Functional requirements

1 Order- and feed-management

- 11 The system shall highlight entries that passed their expiration date.
- 12 The system shall be able to check if a planned order's invoice total will exceed the monthly budget of an animal.
- 13 The system shall only deduct the invoice total from the budget if an order arrived.
- 14 The system shall provide a arrived checkbox for each order to mark it as arrived with a date stamp.
- 15 If an order does not arrive in time, the system shall mark it "overdue" and highlight it.
- 16 The system has to be implemented so, that the list of authorized feed dealers can be edited by a user with sufficient rights in the system.
- 17 The system shall provide a request for additional funds form for the corresponding user if the invoice total of an order would exceed budget or if the planned payment of a temporary worker would exceed the budget.

2 Budget-management

- 21 The system shall be so implemented, that at the end of a month only the spent amount of the budget shall be deducted from the total zoo budget.
- 22 The system shall reserve the sum of the animal budgets from the overall budget.
- 23 The system shall make it possible to alter the monthly budget and the monthly reserved budgets. A user with sufficient rights in the system shall be able to write the new amounts and confirm them with its log-in data. The person or persons that are concerned with the altered budget shall be informed by the system.
- 24 The system shall include a graphical representation of the total monthly budget, money already spent in the current month, reserved monthly sub-budgets (such as budgets for feed or salaries and wages) and money not spent or reserved for this month.
- 25 The system shall provide a graphical representation of: total animal budget of the month, amount spent already, amount reserved for pending deliveries, amount not spent for each animal.
- 26 The system shall balance the account monthly and carry the result to the next month.
- 27 The system shall be able to store and print invoices.
- 28 The system shall highlight the balance green if it's positive and red if it's negative. If it's negative the system shall issue a warning.

29 The system shall send a message to the zoo director if a request for a budget increase is submitted; there shall be a link to the corresponding budget, the stock of the corresponding animal's feed and an average cost per order.

210 The system shall include an overview on the budget spanning the last 6 month. This overview shall include: money spent for every month, average money spent, number of budget increase requests, number of requests granted, number of requests denied, links to the corresponding statements. This overview shall be generated and linked in the message to the zoo director for each request that is submitted.

211 If the zoo director grants an increase request, the system shall make it possible to enter the amount by which the budget is increased. This amount shall be added to the current month's reserve. A message shall be send to the applicant, containing the information that the request was granted and the amount by which the budget was increased. In case of the applicant being a zookeeper a message shall be send to all zookeepers that are responsible for the animal or animals affected by this request. The affected budget shall be increased. In the overview this shall be marked as "additional funds granted on request".

212 If the request is denied, a message that contains a short statement written by the zoo director shall be send to the applicant.

3 Schedule-management

31 The system shall, if a temporary worker is needed, check if there is enough money left in the employee budget.

32 If there is a delay in delivery the system shall inform the secretary.

33 The system shall be able to sort all personnel by tasks, working hours and availability.

34 The system shall warn the user if one tries to include a worker into the schedule during their planned holiday.

35 If the secretary is able to resolve the problem with the food dealer, the system shall send a message to all zoo keepers of the corresponding animal, including a statement and if available the new delivery date. The new delivery date shall be updated to the corresponding order and marked as new delivery date.

36 The system shall inform the zoo director if a temporary worker needs to be hired more than 40 hours in one month.

37 If an user wants to plan a vacation, this user has to get a permission by the secretary. The system shall check for conflicts with other planned vacation of personnel with the same task or tasks and if there are enough unused holidays. If the secretary grants the request for holidays, the system shall send a message to the employee and the planned vacation shall be entered into the employee information. If she does not grant the request, a notification with a short statement shall be send to the applicant by the system. In the employee information the system shall remark that "a request for holidays was denied" with a link to the request or requests.

2.2.5 Nonfuntional Requirements

2.2.6 Datebase requirements

1 The system shall provide a separate budget for every animal.

2 The system shall store information about the available feeds including their quantity and their expiration dates.

3 The system shall provide the information how much of the monthly budget was already spent on an animal.

4 The system shall store data on the budget, containing how much was spent, for what and when, how much was earned, by what and when, how much was reserved, for what and when, money left/balance.

5 The system shall store employee information containing the following: salary or wage, working hours, unused holidays, planned holidays, requested holidays, denied holiday requests, availability for overtime and in case of temporary workers availability and hours worked in the current month, hours worked total and the tasks that they are able to fulfill.

6 The system shall be able to store orders, containing its invoces and an expected delivery date.

7 Orders shall be deleteable.

2.2.7 Software System requirements

1 Maintainability

2 Security

3 Stability

4 Cost efficience

2.3 Appendix C: Software Design Document

2.3.1 Introduction

2.3.1.1 Purpose

This document is written, to document the proposed architecture of a zoo management system. The document is inteded for the developers implementing the system, the persons deploying the final system and the other stakeholders to see the process and decide, if the developed system is according to the specifications.

2.3.1.2 Scope

The System developed is a zoo management system, that assists the zoo keepers in ordering food, managing and visualizing the feed cost of each animal, the administrators in managing budgets, preferred feed dealers and the staff. The software requirements specification is already present.

2.3.1.3 Definitions, Acronyms, Abbreviations

term	**short form**	**description**
Java	-	A portable programming language
Java Virtual Machine	JVM	The software system nessecary to run java applications.
Database Management System	DBMS	A software system used to manage, create and maintain databases.
relational DBMS	-	DBMS storing data in tables referencing each other.
Transport Control Protocol / Internet Protocol	TCP/IP	Low level Protocols to transfere data over the internet.
Secure Hypertext Transfer Protocol	HTTPS	The standard protocol to transfer data over the internet in a secure fashion.

2.3.1.4 References

1. SRS Document
2. Enumerated Requirements Document

2.3.1.5 Overview

The proposed architecture will be shown from a static point of view, eg. the decomposition into subsystems and their structure. Then the final deployment will be shown. This includes the distribution of the software across the different hardware used. Furthermore the data management is visible. Therefore the database design is modeled and the used DBMS is mentioned. The next part contains a description of the security features of the system and how the users are managed. The dynamic aspects of the system are following. This contains the description of interfaces between components, the behavior of the system and information flow. Finally, used design patterns are mentioned.

2.3.2 Proposed Software Architecture

2.3.2.1 Overview

The Zoo management system is a client - server system. It is composed out of the following components.

Client The client module is the complete system running on the client computer. It bundles the UI, navigation and can obtain an access token for the inner components.

Server The server module is the system running on the server computer. It hosts the database and provides user authentification capabilities.

FeedManager (FM) The FM system coordinates all requests concerning the supply, storage and management of feeds and the according feed dealers. It is mainly used by zoo keepers for buying feeds and checking the current quantity and expiration date of stock feeds.

EmployeeManager (EM) The EM system is mainly used by the secretary to create employee schedules, organize temporary workers, check the vacation requests of employees and manage the associated budget.

BudgetManager (BM) The BM system organizes the budget of the zoo. It is used to generate reports, get an overview of the current budget and provide information to prevent negative balances. It is mainly used by the zoo director to organize the subbudgets for feeds and employees as well as granting or dening requests for additional funding created by other employees.

UserManager (UM) The UM system is used to create and manage user accounts of the system. It is exclusively used by the zoo director.

Messaging subsystem (MSS) The MSS is used by the system to deliver messages to special user accounts. It is an external system, which is used by the zoo management system.

MessageManager (MM) The Message manager is a client component, which is used to display messages that are received from the MSS.

AuthUtility (AU) The AU is a compontent running on the server. It is the only way to access the system internals. This component checks, if provided login credentials are valid, and if the access level of the user is high enough.

Database subsystem (DBMS) The DBMS is the subsystem used for persistent data storage. It is used as a transaction based system.

2.3.2.2 Subsystem decomposition

2.3.2.2.1 Client The client component contains the main initialisation code. It provides a host window for the other components to draw their UI and navigational elements to switch the used component. In addition the component is used to obtain an access token from the server, which allows other components to comunicate with the server.

2.3.2.2.2 Server The server component contains the initialisation code to start the DBMS and routes incoming connections to the AuthUtility.

2.3.2.2.3 Manager Components The different manager components are implemented in a similar fashion: By using the *AbstractFactory* Pattern to get instances. The following two models figure and emphasis this pattern and the implementaion style.

The *BudgetManager* is instantiated by the *BudgetManagerFactory*. The budgets are modeled as *CompositeBudget*s and *SubBudget*s, both supporting a specific set of operations. In addition animal implements the *Budget* interface, because each *Animal* has an associated budget.

2.3.2.2.4 MessagingSubSystem The messaging susbsystem allows the zoo management system to send messages to specific user accounts and devices. The system has to guarantee the delivery of the messages. The architecture isn't specified any further, because the MSS is an external system and therefore not part of this document.

2.3.2.2.5 AuthUtility The AuthUtility is used to access the database. For every request the AuthUtility checks, if the login credentials are valid and not expired. Then the privilege level for the requested action is checkt. If the user has enough permissions the request is passed to the DBMS. The answer is returned to the client.

2.3.2.2.6 DatabaseManagementSystem The DBMS used is MariaDB. It is a high performance relational database management system. It is used, because it is a open source software and therefore free of costs. In addition it is widely used and tested. Therefore it is reliable and it may be assumed, that it is supported for the entire lifetime of the zoo management system.

2.3.2.3 Hardware/Software mapping

Figure shows the planned hardware software mapping of the zoo management system. The System is divided into a server and a client part. The server part will run on a server computer, availiable 24/7. The server is responsible for user authentication and the storage of data. The client side consists of tablets and personal computers. These will run the client application, which is used to contact the server via a HTTPS (on top of TCP/IP). The client application requires the Java Runtime Envirement to be installed on personal computers and tablet clients. The client provides the user interface and uses the messaging subsystem to deliver and receive messages to/from users of the system. The hardware allocation of the MSS is not part of this document.

2.3.2.4 Persistent data management

For persistent data management on the server side the open source relational DBMS MariaDB is used. Data is organized in tables according to the scheme in figure. Users have multiple activity logs associated. Each employee has exactly one user account for the system. For each employee, the associated vacation requests and a schedule is stored. Employees are workers of the zoo, so some data is shared with temporary workers (which are workers, too), for example name and wage/salary. For each worker there is possibly a WorkerFundRequest, if the budget was too small to hire the worker. This, in turn, is a specialization of a general FundRequest, which is associated with a given budget. Each budget consists of multiple subbudgets and may contain FundRequests. Another possible FundRequest is the FeedFundRequest, which is issued, if an order exceeds the feed budget. These requests and the associated orders are stored and linked. An order can be delayed and the delivery may be updated, so the possible DeliveryUpdates are stored. The order itself contains one or more Feed offerings. These

offerings are stored as an association between FeedDealer and Feed, because the FeedDealer can make an offer for a feed. If a FeedDealer is removed from the system, all the associated feed offerings not contained in an order are deleted too. In addition the system stores all animals and the feeds they are eating. If an animal is removed from the system, the feeds consumed only by this one animal are deleted. If a feed is deleted, the associated offerings are removed, if they aren't contained in any order.

On the client side no DBMS is neseccary, because all used data is loaded at runtime. The only presistent data is the username, which is stored for the next usage of the system.

2.3.2.5 Access control and security

2.3.2.5.1 Network location restrictions The first layer of security is, to grant access to the system only out of the local network in the zoo. If this is too restrictive, it may be dropped.

2.3.2.5.2 Network protocol restrictions The client may only connect to the server using a secure HTTPS connection. This require the server to have a valid SSL certificate. The server will drop every connection attempt using insecure protocols.

2.3.2.5.3 User account restrictions The next layer is the AuthUtility. Each user of the system has a user account with a unique user ID and a password. These accounts are managed by the zoo director. When a user tries to contact the application server, he has to provide a valid access token and a valid user account id. The access token is obtained by sending the login credentials to the authentification server. If they are valid an access token is returned. This token has only limited validitiy and has to be obtained again, if timed out. To restrict access for zoo keepers and the secretary, every user account has an integer value associated, which represents the users access level. Each module can check the current access level of the user to determine which functionality is provided.

2.3.2.5.4 Storage policies The user passwords aren't stored in plain text. They are hashed using a state of the art hash funktion (PBKDF2 with iteration count larger than 20000) and stored together with the used hash function, salt, and function parameters (iteration count). Because a lot of people use the same password for different services, this prevents potential intruders from taking the password list and using the passwords on other web services or obtaining passwords from other services and using them for the zoo management system.

2.3.2.5.5 Logging As an additional security feature the system logs the account accessing the system for every access. These logs are visible for the zoo director.

2.3.2.6 Global software control

2.3.2.6.1 Startup behaviour **Server:** When the server machine is booted, the operating system automatically invokes a startup handler of the system. This handler will boot up the database, the AuthUtility and then it will enable the request handling.
Client: The client application starts on user demand. The client will show a login form for the user. When the user entered his login data the client contacts the server to obtain an access token. The client creates different instances of *UIComponentFactory* for each UI component. The components are initialized with the access token and these factory objects.

2.3.2.6.2 Interfaces The `MessagingService` is provided by the MSS. It provides a possibility to notify users with messages. If the notifyUser function is invoked, it is ensured, that the user is notified and the message is stored.

```
Interface MessagingService {
  public void sendMessage(string sender,
      string accesstoken, string recipient, Message message) {
    require accessValid(username, accessToken);
    require messageHandle != null;
    require message.length > 0;
    ensure getMessages(recipient, database.getUser(recipient)
        .getAccessToken()).contains(message);
  }

  public Message[] messages getMessages(string username, string accessToken) {
    require accessValid(username, accessToken);
    require messageHandle != null;
    ensure messages != null;
  }

  public void messageRead(String username, string accessToken Message message) {
    require accessValid(username, accessToken);
    require message != null;
    require getMessages(username, accessToken).contains(message);
    ensure !getMessages(username, accessToken).contains(message);
  }
}
```

The `AccessWebService` interface provides three functions: *getAccessToken*, *accessValid* and *access*. *getAccessToken* is used to obtain access tokens. *accessValid* checks, if the provided access token is valid. This function is used by the MSS to authenticate users. The function *access* is used to execute operations on the database. In before a authentification check is made and it is tested if the user is allowed to execute the given operation.

```
Interface AccessWebService {
  public string accesstoken getAccessToken(string username, string password) {
    require username.length > 0 && password.length > 0;
    ensure database.getUser(username).accesstoken == accesstoken &&
    database.getUser(username).expirationDate;
    ensure containsLogEntry(user, system.getCurrentDate());
  }

  public boolean valid accessValid(string username, string accesstoken) {
  require database.existsEntry(username);
  ensure valid == database.getUser(username).isIsValidToken(accesstoken);
  }

  public Response r access(string username,
                           string accesstoken, Operation op) {
    require database.entryExists(username);
    ensure iff accessValid(username, accesstoken) then
```

```
      iff database.getUser(username).isLocked() then
        operation == new PasswordChangeRequiredResponse();
      else
        iff database.getUser().getPrivilegeLvl() > getPrivilegeLvl(operation) then
          operation is executed and r is the response of the operation;
        else
          r == new AccessDeniedResponse();
    ensure if accesstoken is not valid operation is not executed and r is null;
  }
}
```

2.3.2.6.3 Sequences table

Sequence Name	**accessValid**
Preconditions	◦ username is contained in the database ◦ accesstoken is not null
Event order	1.0 initial call 1.1 AuthUtility requests the given User from the DB 1.2 The DB allocates and populates a User Object 1.3 The BD returns the User Object 1.4 AuthUtility ask the User object, if the token is valid 1.5 The User Object returns a boolean 1.6 This boolean is passed to the client
Postconditions	◦ the database is not modified
Quality requirements	◦ The accessValid call shall finish within at most 3 seconds.

Sequence Name	**getAccessToken**
Preconditions	◦ username is contained in the database ◦ password is not null
Event order	1.0 initial request 1.1 AuthUtility requests the given User from the DB 1.2 The DB allocates and populates a User Object 1.3 The BD returns the User Object 1.4 The AuthUtility generates the password hash using HashImpl 1.6 The AuthUtility gets the original password hash from the User object 1.8 The hashes are compared 1.9 The hashes were equal so an access token is requested from the user object 1.10 The user object checks, if the currently active access token is valid 1.11 The token wasn't valid, so a new token is generated 1.12 The new Token is saved in the Database 1.13 Now the access token is valid and therefore returned to the AuthUtility 1.14 From there on it is passed to the callee 1.15 The password was false so null is returned
Postconditions	◦ the returned access token is valid ◦ the returned access token is saved persistently in the DB
Quality requirements	◦ The getAccessToken call shall finish within at most 3 seconds.

Sequence Name	**access**
Preconditions	◦ username is contained in the database ◦ access token is not null ◦ the operation is not null
Event order	1.0 initial request 1.1 AuthUtility requests the given User from the DB 1.2 The DB allocates and populates a User Object 1.3 The BD returns the User Object 1.4 it is checked, if the access token is valid (using the User object) 1.6 the token was valid, so the access level of the user is checked 1.8 the access level of the operation and the user are compared 1.9 the user is allowed to execute the operation, so the operation is send to the DB 1.10 The DB Response is passed to the AuthUtility 1.11 The Response is passed to the callee 1.12 the user had no sufficient access permissions so null is returned to the callee 1.13 the access token was not valid, so null is returned
Postconditions	◦ the database is not modified
Quality requirements	◦ The accessValid call shall finish within at most 3 seconds.

2.3.2.7 Boundary conditions

1. MSS exists already.
2. The zoo has a sufficiently fast network connection availible to use the system without maintaining full disk caches.

2.3.2.8 Patterns

2.3.2.8.1 AbstractFactory The abstract factory pattern is used, to allow the client component to initialize the UI without knowing the exact implementation. The client com-

ponent provides different interfaces (figure) which are implemented by UI components (for example figure and figure).

2.3.2.8.2 Composition The composition pattern is used by the BudgetManager (figure). The interface *Budget* specifies common operations possible on composed budgets and atomic budgets. It is implemented in different subclasses for example *SubBudget* and *Composite-Budget*. This is useful, because the controller doesn't have to know if a budget is composed out of different budgets or if it is an atomic budget. This simplifies the implementation of the overview screen and other parts using the budget.

Chapter 3

Evaluation

3.1 Appendix E: Tests & Results

3.1.1 Data Sets

3.1.2 Tests

3.1.3 Benchmarks

3.1.4 Visualizations