Relational Database Project Report

Table of Contents

Introduction 1
Requirements 1
  Working Hypotheses 1
Specification 2
  Actors and Use Cases 2
  Application Dataflow 3
  Software Architecture 4
  Class diagram 4
Database Conceptual Design 5
  Entities Definition 5
  Entities Attributes 5
  E-R Diagram (with Generalizations) 7
  Generalizations Resolutions 8
  Database Operations 9
  Database Volume Table 10
  E-R Diagram (final) 11
Database Logical Design 12
  Database Tables 12
  Database Normalization 13
  Database Schema 14
Class Diagram (Final) 14

Introduction

The following paper illustrates the design steps and the choices made for the development of an application based on a relational database devised to meet the data management requirements of a fictional tech company named *Innovative Solution*.

The first part of the paper covers the application's functional and non-functional requirements from which the application's specifications are derived, consisting among other things of the actors involved and their use cases, the application dataflow, and an overall description of the software architecture.

The second part of the paper focuses on the theoretical and logical steps in database theory that led us to the design choices made in the development of the database as an integral part of the application.

Requirements

Innovative Solutions is a company whose core business consists in the retailing of electronic IoT-oriented products assembled in-house by teams of employees from components purchased from various suppliers.

The company requires a software solution which should be used by its system administrator to oversee and control all the information concerning the company, by its team leaders, who should be able to add or remove members from their teams and add assembled products, and by the company's customers, who will be allowed to purchase the products and review their orders.

The design of the application will also rely on the following:

Working Hypotheses

- *Customers* may purchase any *Products* up to their available quantities
- Each *Employee* in the company may belong to up to one *Team*
- *Teams* are assigned to the assembly of the *Products* offered by the company, each *Product* being assembled by a single team
- *Products* are composed of one or more *Components*
- *Components* are purchased from a list of *Suppliers*, where different suppliers may offer the same component at different prices

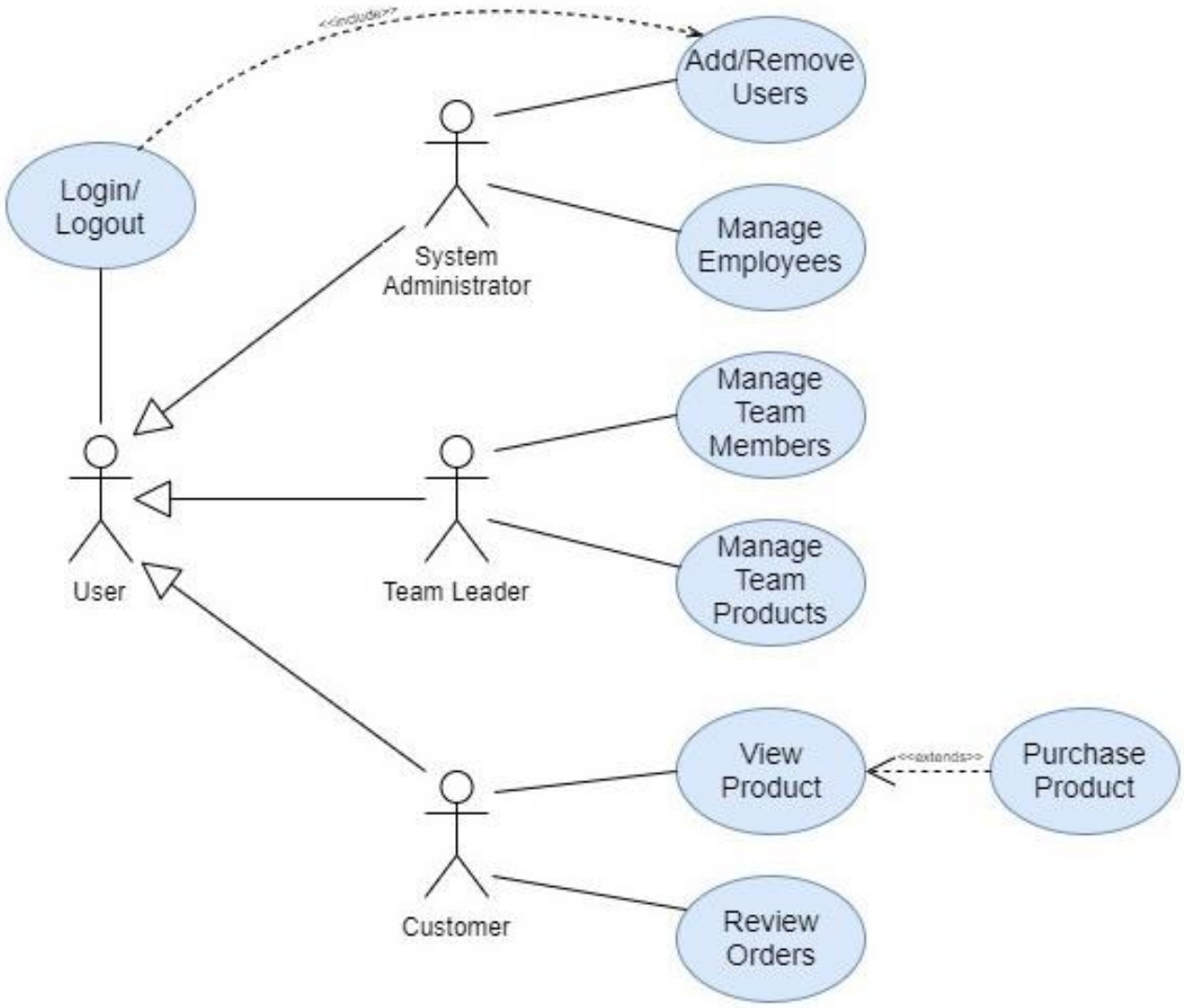
Specification

Actors and Use Cases

Given the software requirements, the application is meant to be used by three different actors, each being allowed to perform a different set of operations:

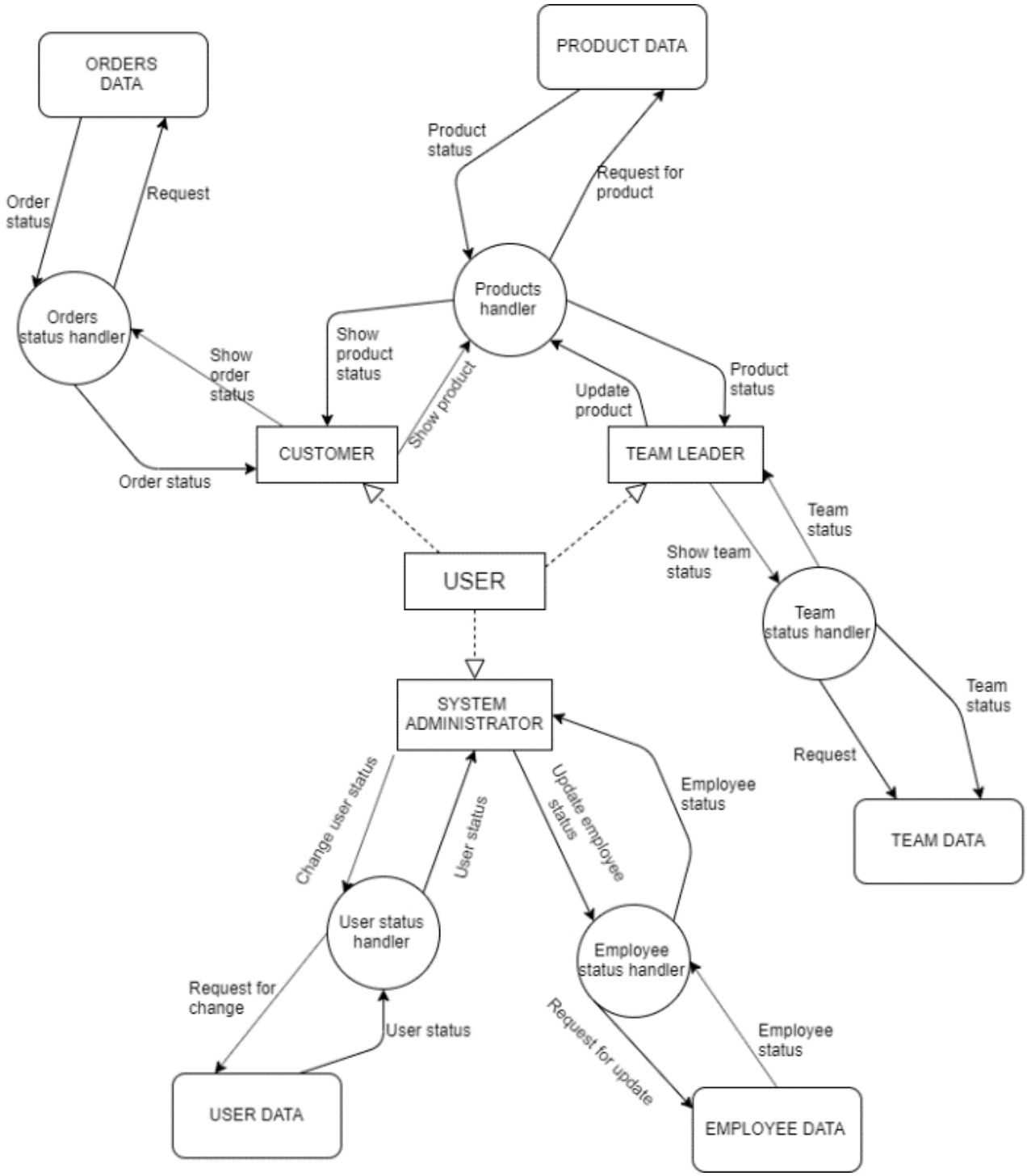
- The company's system administrator, who is permitted to perform a number of actions requiring a high level of privilege.
- The team leaders, who are allowed to manage the members and the products assigned to their teams.
- The customers, who may view and purchase the products on offer, as well as review their past orders.

Hence, the diagram of the application's use cases appears as follows:



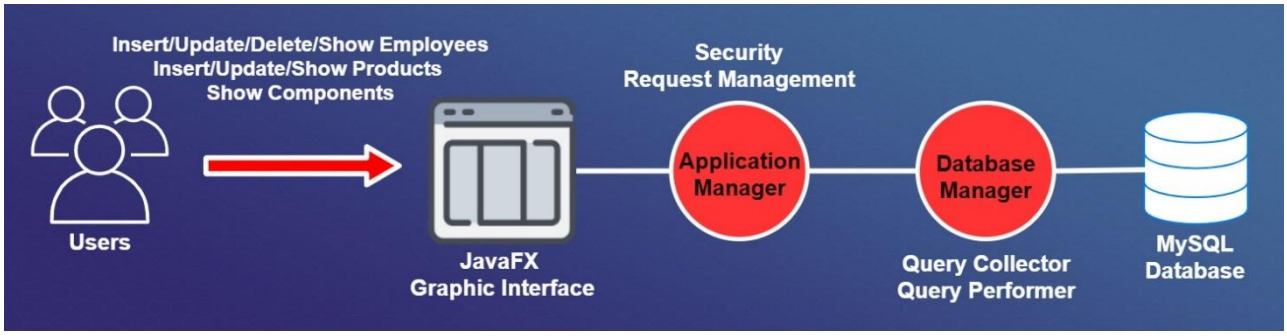
Application Dataflow

The projected dataflow of the application is outlined below:



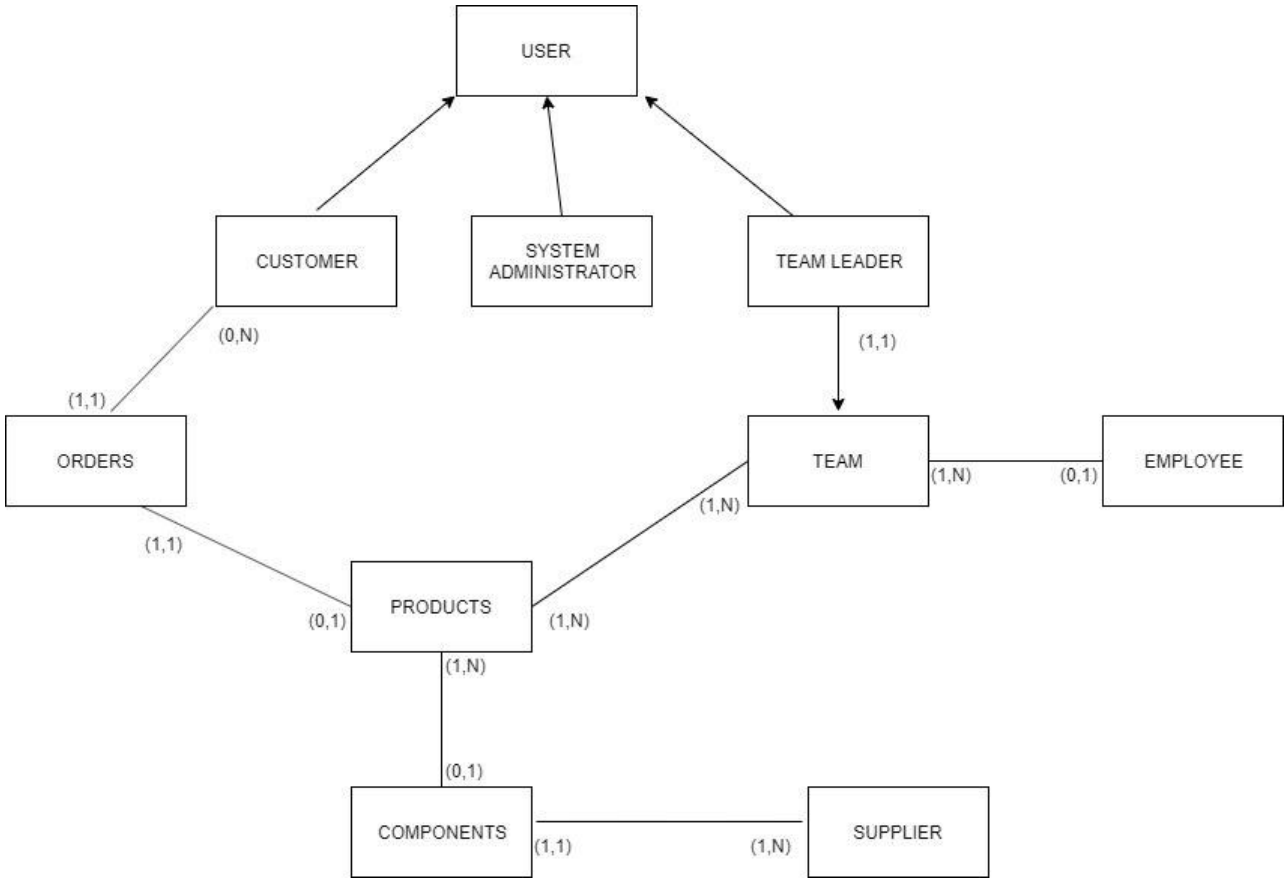
Software Architecture

Since the application should not require computer expertise to be used, to enhance the non-functional requirement of usability the software solution will include a front-end module written in Java providing a graphical interface through which authenticated users will connect to and interact with an underlying MySQL database according to their level of privilege.



Class diagram

the front-end Java module will be developed according to the following architecture:



Database Conceptual Design

Entities Definition

From the application's requirements and the additional working hypotheses the following entities are identified as the first step in the conceptual design of the database:

- **USER**: A generic user of the application, that is the system administrator, the team leaders or the company's customers
- **CUSTOMER**: The company's customers, who may purchase products and review their orders
- **EMPLOYEE**: The company's employees, where each employee may belong to a single team
- **TEAM:** The company's teams, which are composed of a team leader and other members, and assemble products from their base components
- **ITEM**: The items used by the company in its production process, which may consist in products or components
- **PRODUCT**: The list of products offered by the company , which can be bought by customers
- **PRODUCT_STOCK**: Represents the products the company has in stock, and so available for purchase by the customers
- **COMPONENT**: The list of components used by the company to assemble its products
- **COMPONENT_STOCK**: Represents the components the company has in stock, and so available for assembly by the teams
- **SUPPLIER**: The suppliers the components are purchased from, where different suppliers may offer the same product at different prices

Entities Attributes

The list of the attributes related to each entity, including its identifiers, is outlined in the table below:

ENTITY	ATTRIBUTES
USER	username, name, surname, password, mail
CUSTOMER	IDcostumer, address
EMPLOYEE	IDemployee, role, salary
TEAM	IDteam, location, teamLeader
ITEM	itemType, itemName, itemDescription, itemAvailability
PRODUCT	productType, productPrice
PRODUCT_STOCK	IDproduct, productType
COMPONENT	componentType
COMPONENT_STOCK	IDcomponent, componentType
SUPPLIER	IDsupplier, companyName, supplierMail

- An instance of the USER entity is uniquely identified by its username, which in conjunction with its password allows a user to access the application.
  Other information related to a user consists in its real name and surname and its email.

- An instance of the CUSTOMER entity is uniquely identified by its IDcustomer, where the customer's address is also recorded for product shipping purposes.

- An instance of the EMPLOYEE entity is uniquely identified by its IDemployee, and additional information related to an employee is given by his role in the company and the team he belongs to, if any.
- An instance of the TEAM entity is uniquely identified by its IDteam, where the working location and the leader of each team is also recorded.
- An instance of the ITEM entity is uniquely identified by its itemType, and the other attributes are used to track his name, a brief description, and its availability in the company's stock.
- An instance of the PRODUCT entity is uniquely identified by its productType, where the productPrice attribute represents the company's current asking price for the product.
- An instance of the PRODUCT_STOCK entity is uniquely identified by its IDproduct, which identifies a single product in stock.
- An instance of the COMPONENT entity is uniquely identified by its componentType.
- An instance of the COMPONENT_STOCK entity is uniquely identified by its IDcomponent, which identifies a single component in stock.
- An instance of the SUPPLIER entity is uniquely identified by its IDsupplier, where the supplier's company name and email are also recorded.

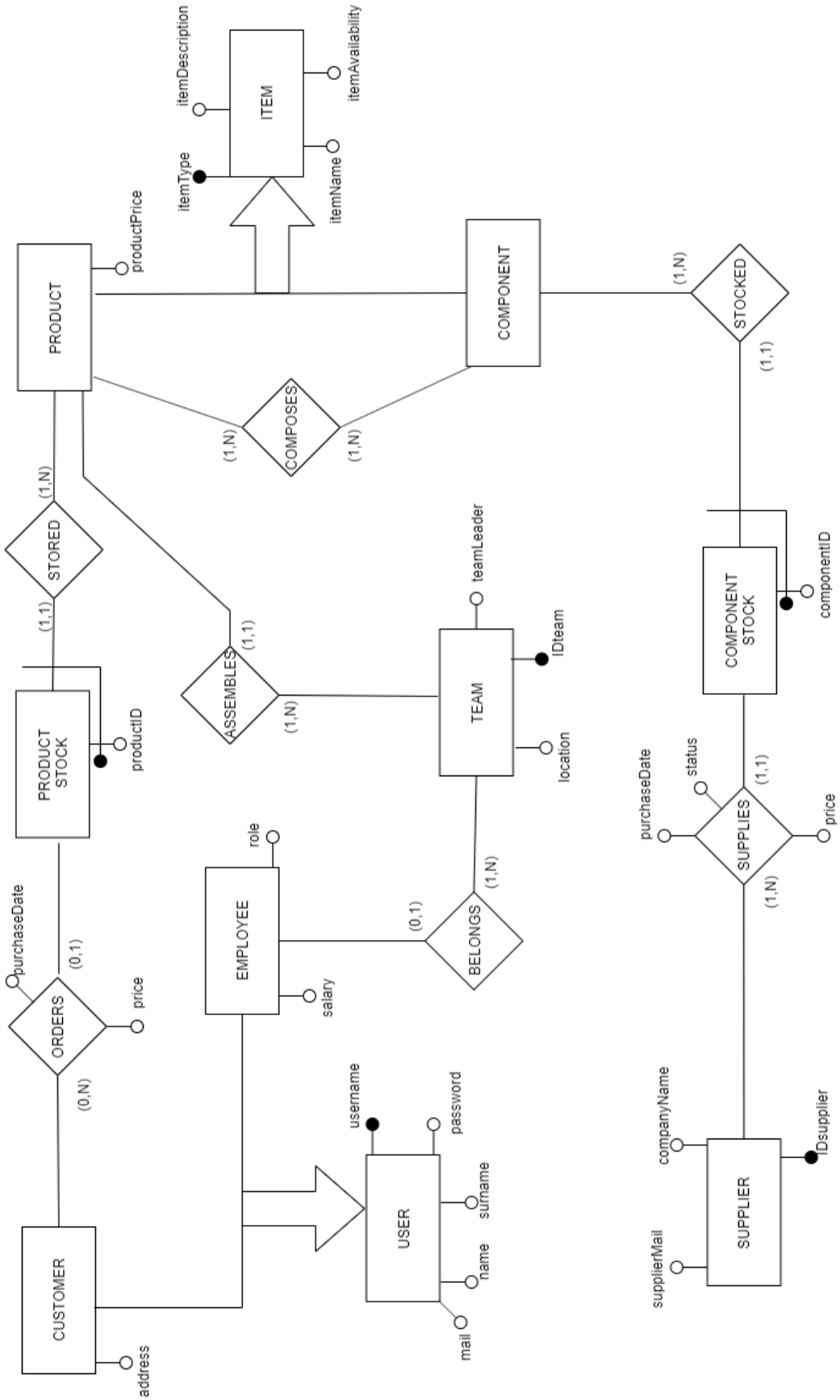
Entity Relationships

The following table describes the relationships between the entities identified in the previous step, along with their cardinality and description

RELATION	ENTITY A (cardinality)	ENTITY B (cardinality)	DESCRIPTION
BELONGS	EMPLOYEE (0,1)	TEAM (1,N)	An Employee may belong to up to one team, while a team is composed of at least one member (its team leader)
ASSEMBLES	TEAM (1,N)	PRODUCT (1,1)	A Team assembles one or more different types of products, while each type of product is assembled by a single team
ORDERS	CUSTOMER (0,N)	PRODUCT_STOCK (0,1)	A Customer may purchase zero or more products in stock, while each single Product in stock may be sold to a single Customer
STORED	PRODUCT_STOCK (1,1)	PRODUCT (1,N)	Each Product in stock belongs to a single product type, while for each product type there may be one or more products in stock
COMPOSES	PRODUCT (1,N)	COMPONENT (1,N)	Each Product is assembled from one or more components, while each Component may be part of one or more products
STOCKED	COMPONENT_STOCK (1,1)	COMPONENT (1,N)	Each Component in stock belongs to a single component type, while for each component type there may be one or more components in stock
SUPPLIES	COMPONENT_STOCK (1,N)	SUPPLIER (1,N)	Each component type is supplied by one or more Suppliers, while each supplier provides one or more components

E-R Diagram (with Generalizations)

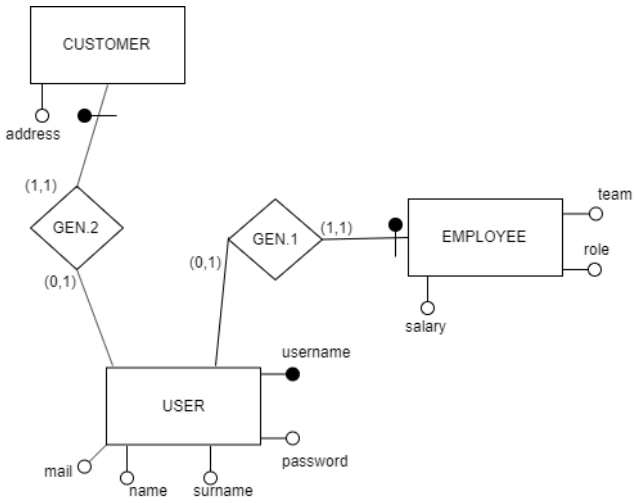
As for the entities and their relationships identified in the first part of the database conceptual design, the database E-R Diagram appears as follows:



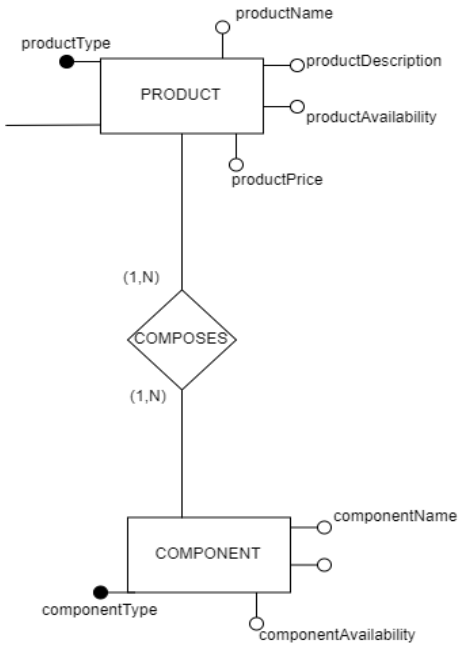
Generalizations Resolutions

We can observe two generalizations in the E-R diagram:

1) USER – CUSTOMER, EMPLOYEE
   The USER entity represents a generalization of the CUSTOMER and EMPLOYEE entities, and our design choice was to resolve this generalization by not aggregating the father entity (USER) into the two children entities (CUSTOMER, EMPLOYEE).
   This solution, though involving an increase in the average number of accesses to the database, makes it possible to avoid the use in the USER table of NULL values for the attributes relative to the employees for the customers and for the attributes relative to the customers for the employees
   Thus the resolved generalization appears as follows:



2) ITEM – PRODUCT, COMPONENT
   The ITEM entity represents a generalization of the CUSTOMER and EMPLOYEE entities, where in this case we have decided to resolve the generalization by aggregating the father entity (ITEM) into the two children entities (PRODUCT, COMPONENT), i.e. removing the father entity while transferring its attributes to both children.



Database Operations

In compliance with the application's requirements and the use cases we have defined, the following operations on the database are taken into account:

OPERATION	ACTOR	DESCRIPTION	EXPECTED FREQUENCY (times/day)
ADD USER	Administrator	Allows the administrator to add a User, whether an Employee or a Customer, to the database	3
REMOVE USER	Administrator	Allows the administrator to remove a User from the database	1
UPDATE SALARY	Administrator	Allows the administrator to modify the salary of an Employee	1
SHOW TEAM PRODUCTS	Team Leaders	Allows a team leader to view the list of Products assembled by his Team	200
SHOW TEAM MEMBERS	Team Leaders	Allows a team leader to view the members of his Team	6
ADD PRODUCTS	Team Leaders	Allows a team leader to add one or more assembled products to the stock	60
SHOW PRODUCTS	Customers	Allows a Customer to view the list of products currently available for purchase	200
ADD ORDER	Customers	Allows a Customer to purchase an available Product	20
SHOW ORDERS	Customers	Allows a Customer to review his past orders	100

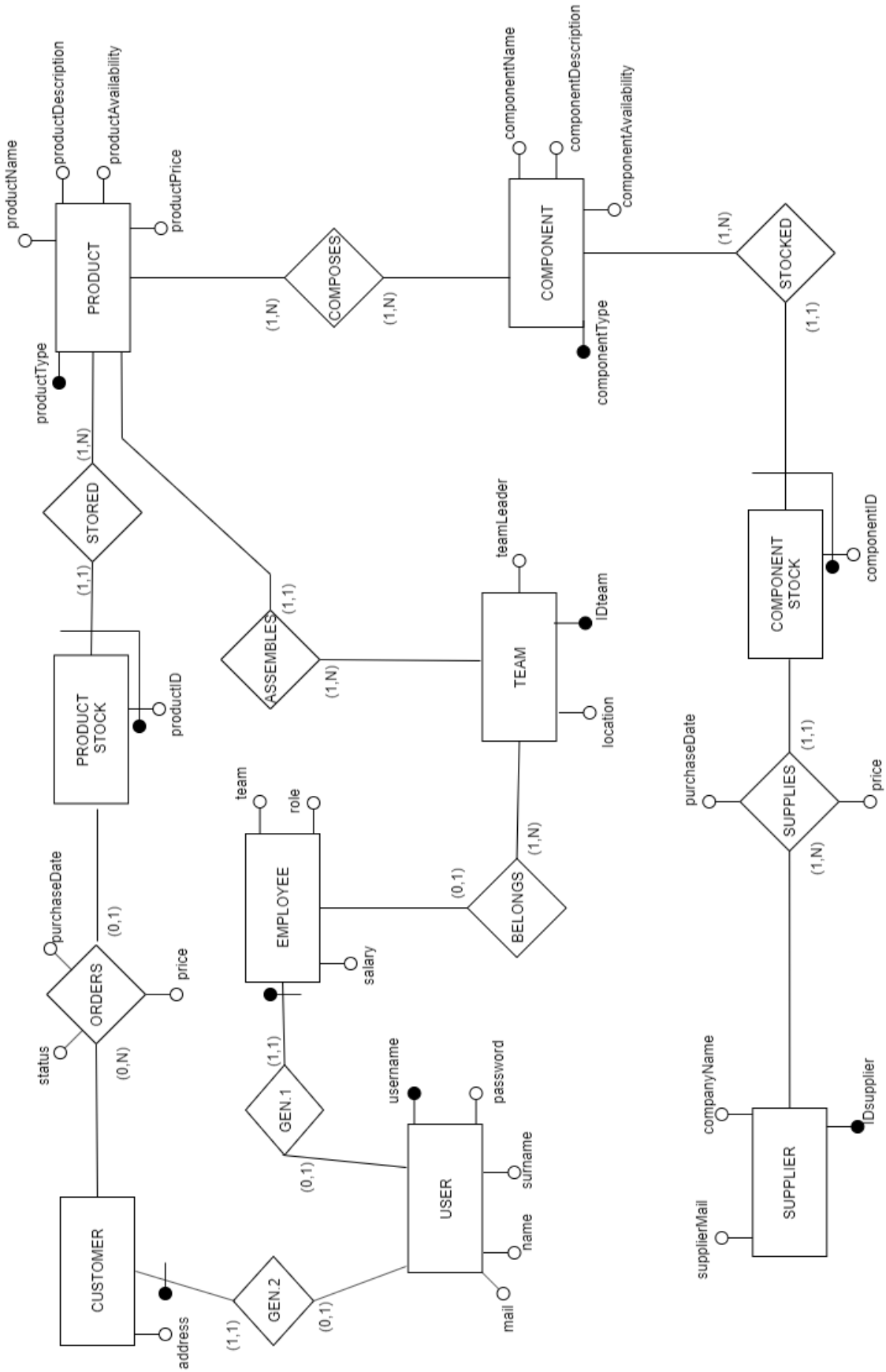
Database Volume Table

The following table shows the expected number of instances of the entities and relationships previously identified in the E-R diagram based on a projected usage of the application:

NAME	E/R	EXPECTED INSTANCES	MOTIVATION
USER	E	100	Initial Hypothesis
GEN.1	R	30	We assume 30% of the users to be Employees
EMPLOYEE	E	30	(1,1) cardinality with GEN.1
GEN.2	R	70	We assume 70% of the users to be Customers
CUSTOMER	E	70	(1,1) cardinality with GEN.2
BELONGS	R	27	We assume 90% of the Employees to belong to a Team
TEAM	E	6	We assume the presence of 6 teams in the company
ASSEMBLES	R	30	We assume that each team assembles 5 products on average
PRODUCT	E	20	We assume the company to offer 20 different types of products
STORED	R	600	We assume the company to keep an average stock of 30 instances for each product type
PRODUCT_STOCK	E	600	(1,1) cardinality with STORED
ORDERS	R	350	We assume each customer to place 5 orders on average
COMPOSES	R	100	We assume products to be assembled on average from 5 components
COMPONENT	E	100	(1,1) cardinality with COMPOSES
STOCKED	R	3000	We assume the company to keep an average stock of 30 instances for each component type
COMPONENT_STOCK	E	3000	(1,1) cardinality with STOCKED
SUPPLIES	R	3000	(1,1) cardinality with COMPONENT_STOCK
SUPPLIER	E	10	We assume each supplier to provide 300 components on average

E-R Diagram (final)

The final E-R Diagram result of the database conceptual design is outlined below:



Database Logical Design

Database Tables

The entities and relationships outlined in the final E-R diagram are derived in the logical design into the following tables:

TABLE	ATTRIBUTES
USER	username, name, surname, password, mail
CUSTOMER	IDcostumer, address
EMPLOYEE	IDemployee, role, salary, team
TEAM	IDteam, location, teamLeader
ASSEMBLES	team, product
PRODUCT	productType, productName, productPrice, productDescription, productAvailability
PRODUCT_STOCK	IDproduct, productType
ORDERS	customer, product, purchaseDate, price, status
COMPOSES	product, component
COMPONENT	componentType, componentName, componentDescription, componentAvailability
COMPONENT_STOCK	IDcomponent, componentType
SUPPLIES	component, supplier, purchaseDate, price, status
SUPPLIER	IDsupplier, companyName, supplierMail

We can also identify the following referential integrity constraints between the tables:

- Between the "username" attribute of the USER table and the "IDemployee" attribute of the EMPLOYEE table
- Between the "username" attribute of the USER table and the "IDcustomer" attribute of the CUSTOMER table
- Between the "team" attribute of the EMPLOYEE table and the "IDteam" attribute of the TEAM table
- Between the "IDteam" attribute of the TEAM table and the "team" attribute of the ASSEMBLES table
- Between the "productType" attribute of the PRODUCT table and the "product" attribute of the ASSEMBLES table
- Between the "productType" attribute of the PRODUCT_STOCK table and the "productType" attribute of the PRODUCT table
- Between the "IDcustomer" attribute of the CUSTOMER table and the "customer" attribute of the ORDERS table
- Between the "productid" attribute of the PRODUCT_STOCK table and the "product" attribute of the ORDERS table

- Between the “productType” attribute of the PRODUCT table and the “product” attribute of the COMPOSES table
- Between the “componentType” attribute of the COMPONENT table and the “component” attribute of the COMPOSES table
- Between the “componentType” attribute of the COMPONENT_STOCK table and the “componentType” attribute of the COMPONENT table
- Between the “componentID” attribute of the COMPONENT_STOCK table and the “component” attribute of the SUPPLIES table
- Between the “IDsupplier” attribute of the SUPPLIER table and the “supplier” attribute of the SUPPLIES table

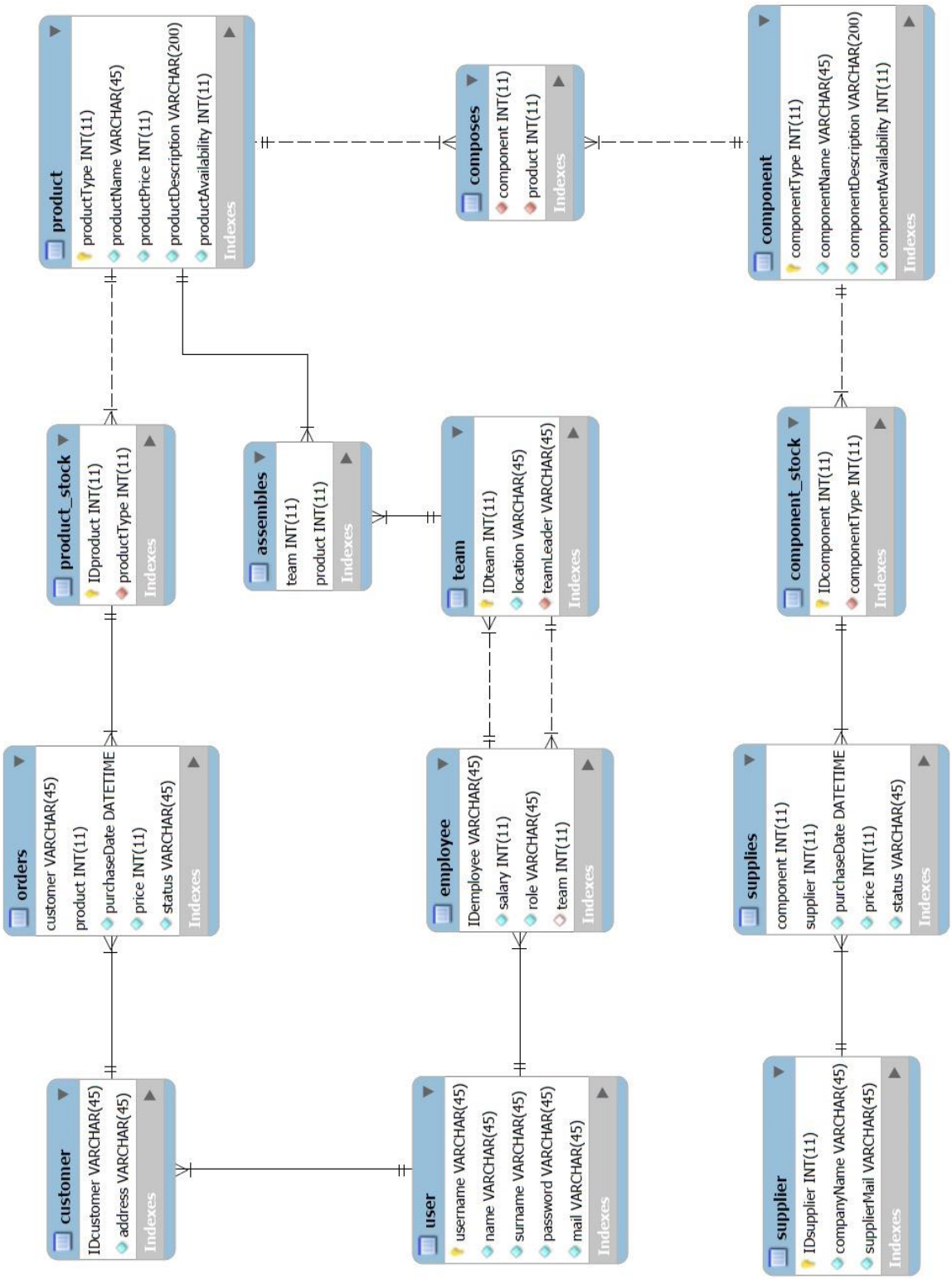
Database Normalization

All the tables in the database as presented attune to the Boyce-Codd normal form since, for every functional dependency X→Y in a table, X represents a superkey for the table, thus guaranteeing the absence of redundancies related to functional dependencies.

TABLE	ATTRIBUTES
USER	username → name, surname, password, mail
CUSTOMER	IDcostumer → address
EMPLOYEE	IDemployee → role, salary, team
TEAM	IDteam → location, teamLeader
ASSEMBLES	team, product
PRODUCT	productType → productName, productPrice, productDescription, productAvailability
PRODUCT_STOCK	IDproduct → productType
ORDERS	customer, product → purchaseDate, price, status
COMPOSES	product → component
COMPONENT	componentType → componentName, componentDescription, componentAvailability
COMPONENT_STOCK	IDcomponent → componentType
SUPPLIES	component, supplier → purchaseDate, price, status
SUPPLIER	IDsupplier → companyName, supplierMail

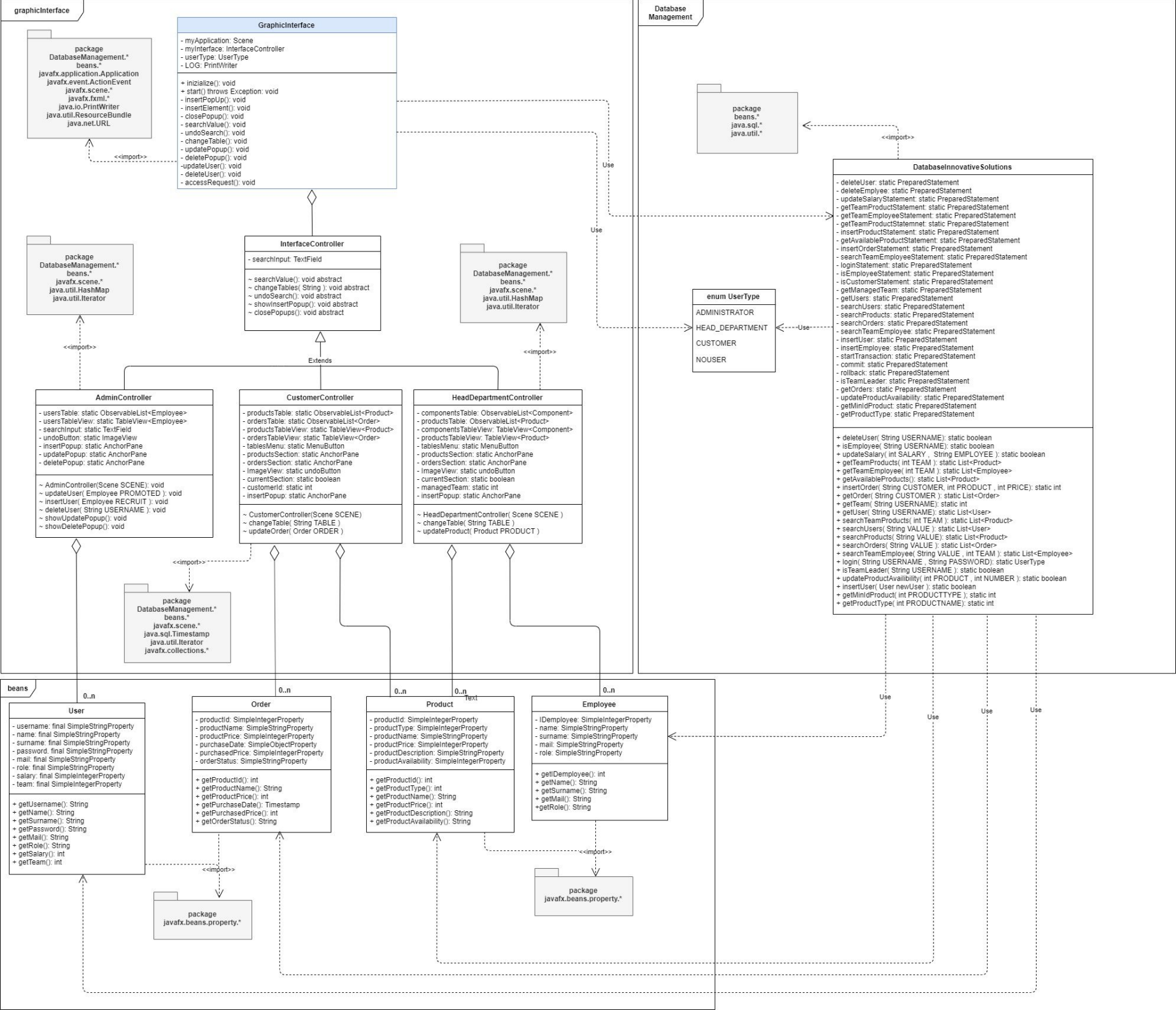
Database Schema

The schema resulting from the database logical design is shown below



Class Diagram (Final)

We also present the final class diagram relative to the front-end Java module


graphicInterface
GraphicInterface
- myApplication: Scene
- myInterface: InterfaceController
- userType: UserType
- LOG: PrintWriter
+ inizialize(): void
+ start() throws Exception: void
- insertPopUp(): void
- insertElement(): void
- closePopup(): void
- searchValue(): void
- undoSearch(): void
- changeTable(): void
- updatePopup(): void
- deletePopup(): void
-updateUser(): void
- deleteUser(): void
- accessRequest(): void
package DatabaseManagement.* beans.* javafx.application.Application javafx.event.ActionEvent javafx.scene.* javafx.fxml.* java.io.PrintWriter java.util.ResourceBundle java.net.URL
<<import>>
Use
InterfaceController
- searchInput: TextField
~ searchValue(): void abstract
~ changeTables(String): void abstract
~ undoSearch(): void abstract
~ showInsertPopup(): void abstract
~ closePopups(): void abstract
Extends
package DatabaseManagement.* beans.* javafx.scene.* java.util.HashMap java.util.Iterator
package DatabaseManagement.* beans.* javafx.scene.* java.util.HashMap java.util.Iterator
AdminController
- usersTable: static ObservableList<Employee>
- usersTableView: static TableView<Employee>
- searchInput: static TextField
- undoButton: static ImageView
- insertPopup: static AnchorPane
- updatePopup: static AnchorPane
- deletePopup: static AnchorPane
~ AdminController(Scene SCENE): void
~ updateUser(Employee PROMOTED): void
~ insertUser(Employee RECRUIT): void
~ deleteUser(String USERNAME): void
~ showUpdatePopup(): void
~ showDeletePopup(): void
CustomerController
- productsTable: static ObservableList<Product>
- ordersTable: static ObservableList<Order>
- productsTableView: static TableView<Product>
- ordersTableView: static TableView<Order>
- tablesMenu: static MenuButton
- productsSection: static AnchorPane
- ordersSection: static AnchorPane
- ImageView: static undoButton
- currentSection: static boolean
- customerId: static int
- insertPopup: static AnchorPane
~ CustomerController(Scene SCENE)
~ changeTable(String TABLE)
~ updateOrder(Order ORDER)
HeadDepartmentController
- componentsTable: ObservableList<Component>
- productsTable: ObservableList<Product>
- componentsTableView: TableView<Component>
- productsTableView: TableView<Product>
- tablesMenu: static MenuButton
- productsSection: static AnchorPane
- ordersSection: static AnchorPane
- ImageView: static undoButton
- currentSection: static boolean
- managedTeam: static int
- insertPopup: static AnchorPane
~ HeadDepartmentController(Scene SCENE)
~ changeTable(String TABLE)
~ updateProduct(Product PRODUCT)
<<import>>
package DatabaseManagement.* beans.* javafx.scene.* java.sql.Timestamp java.util.Iterator javafx.collections.*
Database Management
package beans.* java.sql.* java.util.*
<<import>>
DatabaseInnovativeSolutions
- deleteUser: static PreparedStatement
- deleteEmplyee: static PreparedStatement
- updateSalaryStatement: static PreparedStatement
- getTeamProductStatement: static PreparedStatement
- getTeamEmployeeStatement: static PreparedStatement
- getTeamProductStatemnet: static PreparedStatement
- insertProductStatement: static PreparedStatement
- getAvailableProductStatement: static PreparedStatement
- insertOrderStatement: static PreparedStatement
- searchTeamEmployeeStatement: static PreparedStatement
- loginStatement: static PreparedStatement
- isEmployeeStatement: static PreparedStatement
- isCustomerStatement: static PreparedStatement
- getManagedTeam: static PreparedStatement
- getUsers: static PreparedStatement
- searchUsers: static PreparedStatement
- searchProducts: static PreparedStatement
- searchOrders: static PreparedStatement
- searchTeamEmployee: static PreparedStatement
- insertUser: static PreparedStatement
- insertEmployee: static PreparedStatement
- startTransaction: static PreparedStatement
- commit: static PreparedStatement
- rollback: static PreparedStatement
- isTeamLeader: static PreparedStatement
- getOrders: static PreparedStatement
- updateProductAvailability: static PreparedStatement
- getMinIdProduct: static PreparedStatement
- getProductType: static PreparedStatement
+ deleteUser(String USERNAME): static boolean
+ isEmployee(String USERNAME): static boolean
+ updateSalary(int SALARY , String EMPLOYEE): static boolean
+ getTeamProducts(int TEAM): static List<Product>
+ getTeamEmployee(int TEAM): static List<Employee>
+ getAvailableProducts(): static List<Product>
+ insertOrder(String CUSTOMER, int PRODUCT , int PRICE): static int
+ getOrder(String CUSTOMER): static List<Order>
+ getTeam(String USERNAME): static int
+ getUser(String USERNAME): static List<User>
+ searchTeamProducts(int TEAM): static List<Product>
+ searchUsers(String VALUE): static List<User>
+ searchProducts(String VALUE): static List<Product>
+ searchOrders(String VALUE): static List<Order>
+ searchTeamEmployee(String VALUE , int TEAM): static List<Employee>
+ login(String USERNAME , String PASSWORD): static UserType
+ isTeamLeader(String USERNAME): static boolean
+ updateProductAvailibility(int PRODUCT , int NUMBER): static boolean
+ insertUser(User newUser): static boolean
+ getMinIdProduct(int PRODUCTTYPE): static int
+ getProductType(int PRODUCTNAME): static int
enum UserType
ADMINISTRATOR
HEAD_DEPARTMENT
CUSTOMER
NOUSER
Use
beans
User
- username: final SimpleStringProperty
- name: final SimpleStringProperty
- surname: final SimpleStringProperty
- password. final SimpleStringProperty
- mail: final SimpleStringProperty
- role: final SimpleStringProperty
- salary: final SimpleIntegerProperty
- team: final SimpleIntegerProperty
+ getUsername(): String
+ getName(): String
+ getSurname(): String
+ getPassword(): String
+ getMail(): String
+ getRole(): String
+ getSalary(): int
+ getTeam(): int
0..n
Order
- productId: SimpleIntegerProperty
- productName: SimpleStringProperty
- productPrice: SimpleIntegerProperty
- purchaseDate: SimpleObjectProperty
- purchasedPrice: SimpleIntegerProperty
- orderStatus: SimpleStringProperty
+ getProductId(): int
+ getProductName(): String
+ getProductPrice(): int
+ getPurchaseDate(): Timestamp
+ getPurchasedPrice(): int
+ getOrderStatus(): String
0..n
Product
- productId: SimpleIntegerProperty
- productType: SimpleIntegerProperty
- productName: SimpleStringProperty
- productPrice: SimpleIntegerProperty
- productDescription: SimpleStringProperty
- productAvailability: SimpleIntegerProperty
+ getProductId(): int
+ getProductType(): int
+ getProductName(): String
+ getProductPrice(): int
+ getProductDescription(): String
+ getProductAvailability(): String
0..n
Text
Employee
- IDemployee: SimpleIntegerProperty
- name: SimpleStringProperty
- surname: SimpleStringProperty
- mail: SimpleStringProperty
- role: SimpleStringProperty
+ getIDemployee(): int
+ getName(): String
+ getSurname(): String
+ getMail(): String
+getRole(): String
0..n
<<import>>
package javafx.beans.property.*
<<import>>
package javafx.beans.property.*
Use
Use
Use
Use