Design document for «Almaty Flowers» project

Central Park Team

Sandibekova Aizhana

Suindikova Zhuldyz

Tunbay Aidana

Taspolat Madina

Version	Date	Author	Change
0.1	05/03/16	Central Park Team	Initial document
0.2	12/04/16	Central Park Team	Complete document

Table of contents

1 Introduction

1.1 Purpose

The purpose of this document is to explain and show the design and architecture of the Almaty Flowers Application.

1.2 Scope

This document explains interfaces, system decomposition and relation, as well as design rationale.

1.3 Definitions, Acronyms, Abbreviations

1.4 Design goals

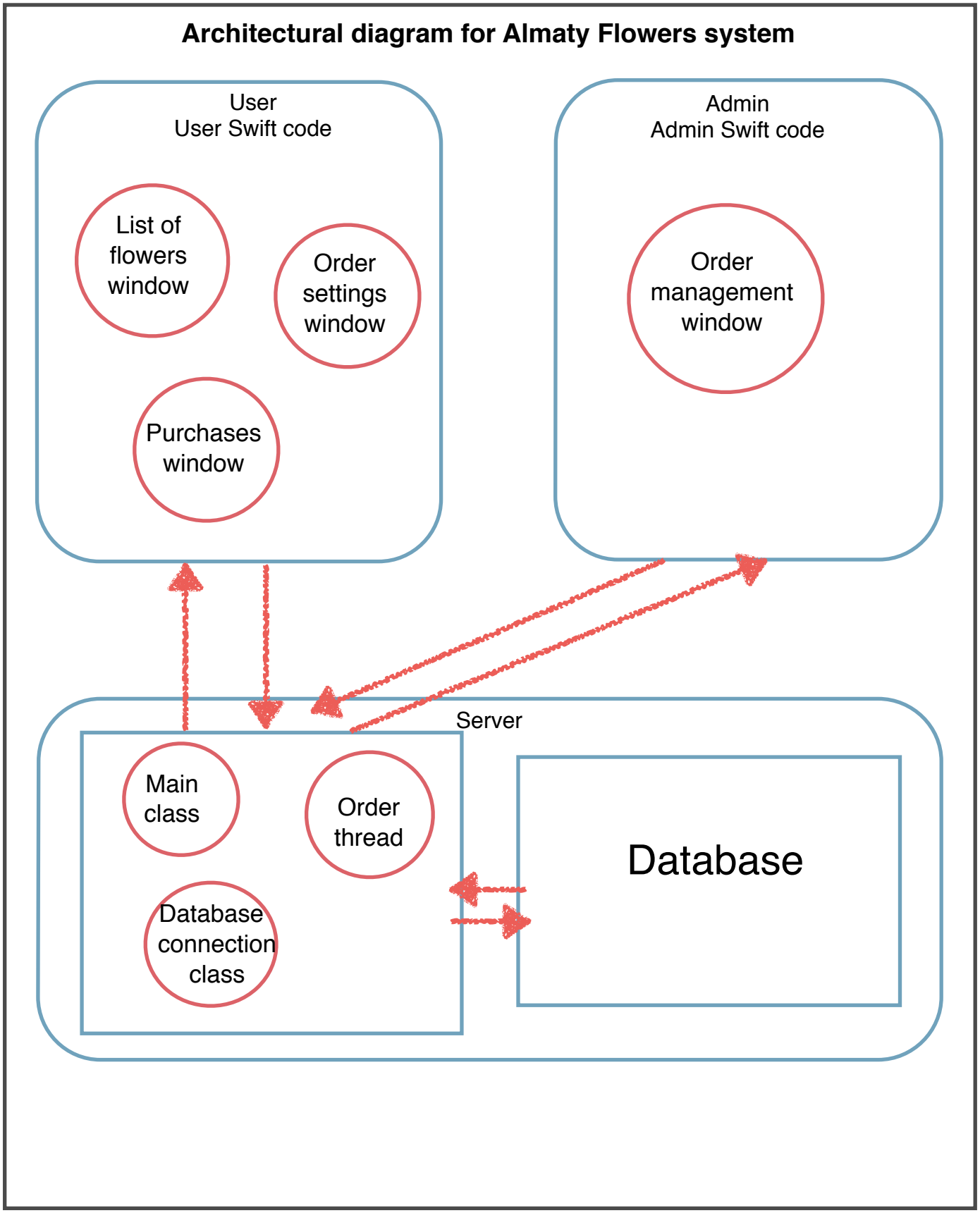
- Allow users to schedule access to their workstation
- Use unused computing cycles on workstations
- Allow multiple jobs to run at the same time versus having to sequence jobs
- Ensure user and data security
- Has to be extensible
- Has to be easy to administer and do much automatically

2. References

[NONE]

3. Decomposition Description

3.1 Module decomposition



Architecture type of this project is the a client-server architecture. User and Administrator modules will represent graphic presentation of user work for user administrator. There is a database where orders will be saved and server will work with these database and event. User, Administrator and Server will have Swift code modules.

3.1.1 User Swift code description

User Swift code is consist of 3 main classes that graphically represents User work area where he/she can order products, pick some products as favorite and add them to busket, configure their order adding information about themselves and product. They end data to the Server Swift code module.

3.1.2 Admin Swift code description

Admin code is consist of only 2 main classes. It graphically presents the work and sends data to Server.

3.1.3 Server Swift code description

Server Swift code is consist of Main class which declare main application; Order thread class which declare how work on each user or admin connection; Database connection class which is accountable for work with database.

3.2 Concurrent process

3.2.1 User Main window description

User Main window is flowers list, where user can see list of flowers.

Pick as favorite: User can pick some product as favorite.

Add to basket: User can add some product (flowers) to the basket.

3.2.2 User Favorite window description

User Favorite window is list of favorite flowers. When user presses icon «Favorite», it will be added to favorites list. It will help to user to save already liked products.

3.2.3 User Basket (Purchases) window description

In this window will be represented list of products that user would buy in a feature.

Delete: User can delete some product from this list pressing button «Delete».
Add: There is «-» and «+» icons around count of flowers, pressing them user can increase or decrease sum of flowers.

3.2.4 Order configuration window description

Order configuration: In the configuration window after defining count of flowers etc. user will leave information about themselves as name, telephone number, address.

3.2.5 Administrator Order Management window description

There will be list of orders that admin can search and admin will control orders.

3.2.6 Server Main class description

Server main class represents main application logic. It will provide events, create connections between client and database.
Order thread: Thread will work with one exactly order connection.
Database main class: Main thread accepts new connections and creates order threads.

3.3 Data decomposition

3.3.1 `Orders` database table.

order_id INTEGER AUTO_INCREMENT

title VARCHAR (255)

flower_id INTEGER NOT NULL REFERENCES flowers.flower_id

Primary Key(order_id)

3.3.2 `Flowers` database table.

id INTEGER AUTO_INCREMENT NOT NULL

title VARCHAR(255) NOT NULL

color TEXT NOT NULL

sum_of_flowers INT NOT NULL

Primary key (id)

3.3.3 `Users` database table

id INTEGER AUTO_INCREMENT NOT NULL

name VARCHAR(255) NOT NULL

surename VARCHAR(255) NOT NULL

number INT NOT NULL

address VARCHAR(255) NOT NULL

user_id INTEGER REFERENCES users_.id order_id INTEGER NOT

Primary key (id)

3.4 STATES

4 Dependency Description

4.1 Intermodule Dependencies

User Swift code module is dependent on Server Swift code module as they are making socket connection with each other.
Admin Swift code module is also dependent on Server Swift code module the same as User's one. Database and Server modules able to connect with each other.

4.2 Interprocess Dependencies

See sequence diagrams in SRS document.

4.3 Data Dependencies

[NONE]

5 Interface Description

5 .1User:

5.1.1 List of flowers window:

5.1.1.1 public void flowerList();
This method shows list of products or flowers.

5.1.1.2 public void addToFavourites();
This method adds selected product to the list of favorites.

5.1.1.3 public void order();
This method will send this product to the basket.

5.1.2 Order settings

5.1.2.1 public void user(name,surname, telephoneNumber, address);
This method saves information about user in the database into the Users table.

5.1.3 Purchases window

5.1.3.1 public void delete(orderId);

This method will delete selected order.

5.1.3.2 public void add(orderId);

This method will change sum of the flowers.

5.1.3.2 public void order(orderId);

This method will create the order and save all information this order to the database into tables orders, users and flowers.

6 Detailed Design

[NOT REQUIRED]

7 Design Rationale

7.1 The order
7.1.1 Description

Server side of our application must connect to the database of any type.

7.1.2 Factors affecting issue

7.1.2.1 The database type used by customer may be changed.

7.1.2.1 Each database have its own syntax.

7.1.3 Alternatives and their pros and cons

7.1.3.1 Write static connection type to each file that needs it.

7.1.3.1.1 In case of database replacement connection settings should be changed in all files one by one

7.1.3.2 Write separate Database connection class

7.1.3.2.1 In case of database replacement connection settings should be changed only once at that file.

7.1.3.2.2 Appropriate relationship with such file should be provided for each class.

7.1.4 Resolution of Issue
We decided to create separate database connection class in case to improve maintainability.

7.2 Database connection

7.2.1 Description

Our application should provide one stable connection to the database.

7.2.2 Factors affecting the issue

7.2.2.1 Too many opened connections may get server go down.

7.2.3 Alternatives and their pros and cons

7.2.3.1 For each client create separate database connection.

7.2.3.1.1 Server may go down when number of connected clients grows up.

7.2.3.2 Create single connection that can be used by any user thread.

7.2.3.2.1 There is only one single connection

7.2.3.2.2 Information update may be slow as connection becomes busy with big number of clients.

7.2.4 Resolution of the Issue
We decided to create one single connection object using Singleton Pattern.

7.3 Administration

7.3.1 Description
Only administrator should be allowed to add new users, and change their privileges

7.3.2 Problems affecting the Issue

7.3.2.1 Staff should not be allowed to add new users.

7.3.2.2 Customer has personnel department which is responsible for that issue.

7.3.3 Alternatives and their pros and cons

7.3.3.1 Create small separate application for administrators only.

7.3.3.1.1 Security level is high because only few people install this application

7.3.3.1.2 One more module and separate database table should be added.