Architecture Document for Phone Book Project

MicroFocus HomeWork Challenge

Created by Noam Nikki Yehuda

1. **Introduction:**

This document is for This document provides a high level overview and explains the architecture of the Phone Book application.

The document defines goals of the architecture and logical layout, design and choices made in the development of this application.

The Phone Book application was designed with a client, server and a graphic user interface (GUI). In this application you have the ability to add, delete and search of different contacts in your phone book.

The server allows you to keep you contacts even when the application is no longer running.

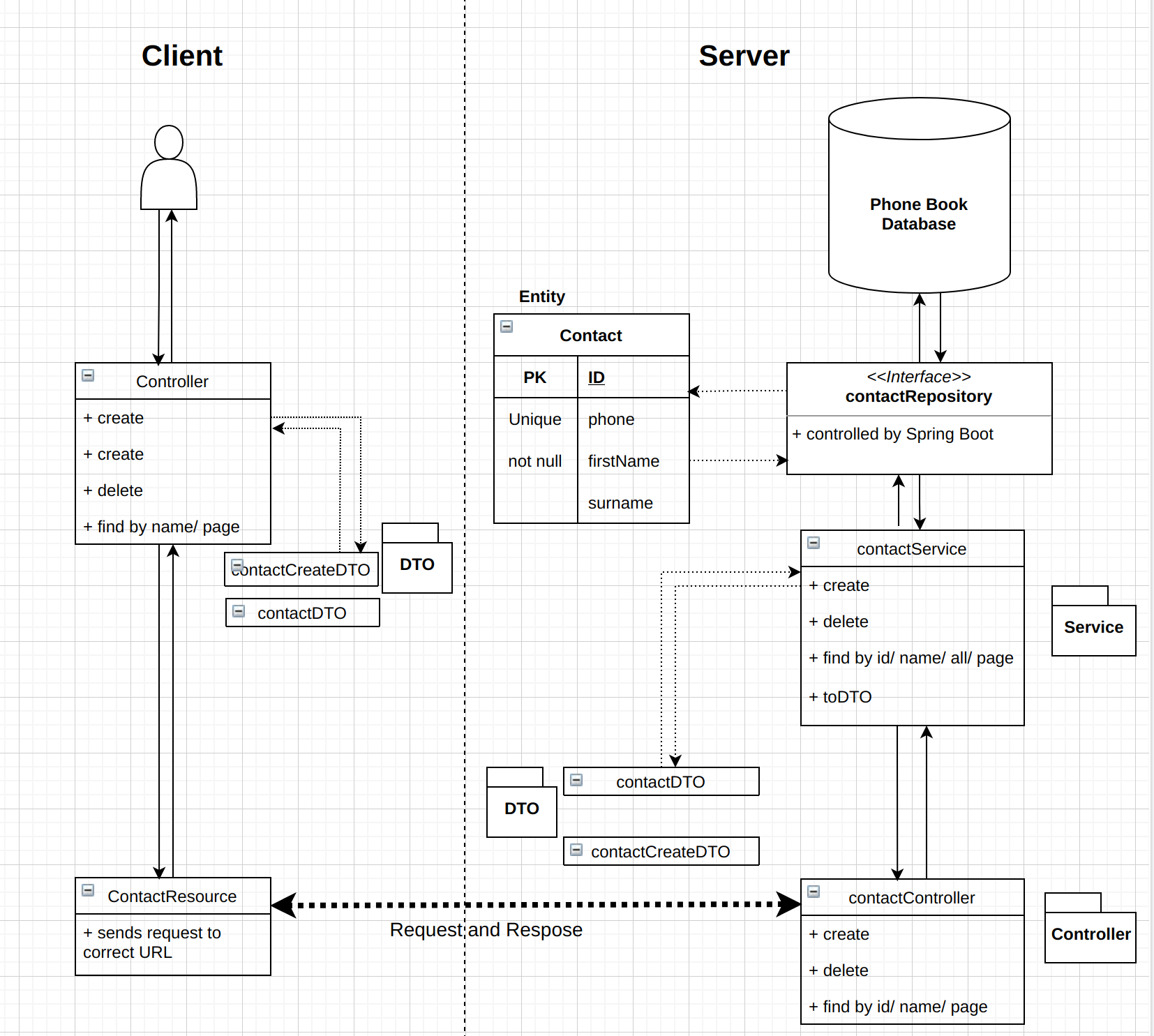
* 1. **Scope:**

The scope of this project is to build a phone book a contact entity, every entity should have at least two fields which are “name” and “phone”. The Phone Book functionalities must include the ability to ‘add an entity’, ‘remove and entity’, ‘list all entities’, ‘and ‘search for all entities by name’.

1. **Project Explanation and Plan:**

My plan is using spring boot and docker to create an application with a client and server that let a person access a database of their contacts. Using docker and prolog I can create a database and link it to my server. My Server will be linked to my database using a repository that will send the information to my services. My services will then transfer the contact to the data transfer object and send it to the client through a controller. The Client will be able to ask for specific contacts by name or see all contacts by page (5 for a page). The client will also be able to delete the contact by ID number. All requests will be done in a GUI, they will be prosseced by the fxml controller and sent to the contact recource which will use a rest template to send the requests by using the correct URL.

**2.1 Packages & Classes:**

****

**Server:**

**Src.main.**

Application: the main class for running the server

Entity.contactEntity: this class holds an entity on contact ( phone, name, and surname)

Repository.contactRepository: this class is a part of Spring Boot, it lets the database communicate with the entity and out application

Services.contactServices: this class communicates with the repository class to get the entity and convert it to the correct DTO

Controller.contactController: this class receives requests from the client and sends it to the service class and then returns the correct content

DTO.

ContactDTO: this class is the Data Transfer Object for contact

ContactCreateDTO: this class holds the contact that the client is requesting to create

**.test.**

testContactController: tests for controller class

testContactServices: tests for services class

**Client:**

**Src.main.**

App & ClientfxApp & StageListener: All basic classes to run the app from the client side and connect to server

Controller: Connects fxml GUI to code and takes in clients requests

ContectResource: takes the clients requests and sends them to server and then sends back response if needed.

**2.2 Test Plan:**

**Test Coverage:**

**Scope:** Our scope for test is all major application functions such as Creating new contact, Deleting a contact, Finding contacts by name, and Displaying all contacts. It is also important to test all inputs that should not be accepted such as contacts with no first name of phone numbers, and contacts with a phone number that is already in the database. Also the client should be tested to make sure that all functions can communicate with the server correctly

**Out of Scope:** Because the application is relatively small the only things out of scope are small helper functions should not be tested.

**Quality Objectives:**

* Ensure the Application Under Test conforms to functional and non-functional requirements
* Ensure the AUT meets the quality specifications defined in scope of project(1.1).
* Bugs/issues are identified and fixed before going live

**Test Methods:**

**Overview**: Because the project requirements are completely defined the testing methodology should be the waterfall model. The waterfall model is very simple to plan and manage and lets projects like this the ability to be easily tested.

**Report Test Results:** Through log file

Diagram

Description automatically generated

**Test Levels:**

* Implementation − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing. The unit tests will be done on all functionalities stated in the scope(create, delete, find, and show all) in every class.
* Integration and Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

**Test Completeness:**

* 100% test coverage
* All Manual & Automated Test cases executed
* All open bugs are fixed or will be fixed in next release

**Testing Tools & Environment:**

* JDK 15
* Junit
* Spring Boot
* JSON

**3.How to Run**

**Requirements:**

* Docker
* JDK 15
* Spring Boot Framework

**Setup:**

1. Download & Install requirements
2. Clone this repository
3. Start your docker application: ‘sudo systemctl start docker’
4. Run this command: ‘docker volume create –name=postgres\_data
5. Run this command inside your directory with cloned repository: ‘sudo docker-compose up’

**Running:**

1. go to micro\_focus\_project and run:

./gradlew test

./gradlew bootRun

2. go to micro\_focus\_client and run:

./gradlew bootRun