DELIVERY TRACKING SYSTEM

Alan, HoKinFai

ROADMAP

Live Demonstration

Introduction

My Approach

UML Diagrams

Test Driven Development

SOLID & Four Pillars

Design Pattern

Lesson Learned

ckage

localhost:8080/deliverytrackingsystem/index.jsp

facebook youtube Google 有道首页 根植澳門 服務澳門 |... Mail & Drive University Programming Job My Dad Health Text Mini

Package

GRAVIDA SEM LIGULA VEL NISL

Members Login: LOGIN

Home | Track | Services | Help | About | Contact | SIGNUP NOW

Business Opportunities | What Our Future Plans | Our Success | News | FAQ | Company Login

GUARANTEED 100%

Our Companies Main Features

Introduction

- *This is a delivery tracking system.*
- *Users can login to view their orders.*
- *Companies can add new orders and update, check existing orders.*

More Ideas

- *This website has been builted using Java JPA and still in development stage.*
- *More functions such as request for orders will be implemented in the second stage.*
- *For more information, please refer to about index.*

More Informations

- View the source code on GitHub

  This is an open source project, you are allowed to refer to my code for non-commercial purposes.

Contact Methods

- Email: hokinfaialan@gmail.com
- LinkedIn
- GitHub
- Facebook

PROJECTS **2007**

Future Plans

Sed semper, enim id fringilla posuere

USER FUNCTIONS

- User-friendly Interface;
- Track a single order;
- User Register;
- Login/logout;
- Password Encryption;
- Update User Detail;
- Check User's Orders;

COMPANY FUNCTIONS

- Company Login/Logout;
- View Company's processing Orders;
- Add Orders to any Registered Users;
- Update any Orders details;

ROADMAP

Live Demonstration

Introduction

My Approach

UML Diagrams

Test Driven Development

SOLID & Four Pillars

Design Pattern

Lesson Learned

INTRODUCTION

- Objective:
  - Establish a delivery tracking system which users (customers & companies) are able to see their placed orders.
  - Target on building one platform for multiple usages.
  - Companies can add and update the orders' details.

ROADMAP

Live Demonstration

Introduction

My Approach

UML Diagrams

Test Driven Development

SOLID & Four Pillars

Design Pattern

Lesson Learned

MY APPROACH

- Decided to work on delivery tracking system;
- Research on the most common functions that a system should have;
- Decided technologies that needed in this project:
    - JPA and Hibernate as the Data access framework;
    - Tomcat as the Server;
    - MySQL/Oracle as the database;
    - HTML/CSS as the user interface;

MY APPROACH

- Agile software development (backend, database working at the same time);
- The use of Kanban board to monitor the development progress;
- 1-2 days sprints (12 hours per day in the first week);
- More functions can be easily added in the future;

SOURCE CONTROL - GITHUB

Compare ▾ Sync

master

- This is the version that has completed all the basic f... — 1 day ago by hokinfai — 37
- Working on testing. — 4 days ago by hokinfai — 56
- The delivery features was replaced by a drop down list. — 9 days ago by hokinfai — 4
- All the basic functions have already been done!!! — 9 days ago by hokinfai — 31
- Functions for changing user details and adding mor... — 12 days ago by hokinfai — 40
- This is the third version. — 13 days ago by hokinfai — 39
- Webpages added! — 14 days ago by hokinfai — 25
- The delivery tracking system with website. — 15 days ago by hokinfai — 83

src/main/java/com/example/dao/OrderImp.java

```
+ import java.util.ArrayList;
+ import java.util.List;
+
  import javax.persistence.EntityManager;
  import javax.persistence.EntityManagerFactory;
  import javax.persistence.Persistence;
@@ -39,6 +42,19 @@ public class OrderImp implements OrderService {
          save(company);
      }

+     public List<String> getUserName(List<Order> order) {
+         List<String> nameList = new ArrayList<String>();
+         for (int i = 0; i < order.size(); i++)
+             nameList.add(order.get(i).getUser().getName());
+         return nameList;
+     }
+
+     public List<Order> getCompanyOrder(Company company) {
+         List<Order> orderList = new ArrayList<Order>();
+         orderList = company.getOrderList();
+         return orderList;
+     }
+
      public Company getCompanyByRegiManaName(String number, String name,
              String manager) {
          Query query = this.em
```

src/main/java/com/example/dao/OrderService.java

ROADMAP

Live Demonstration

Introduction

My Approach

UML Diagrams

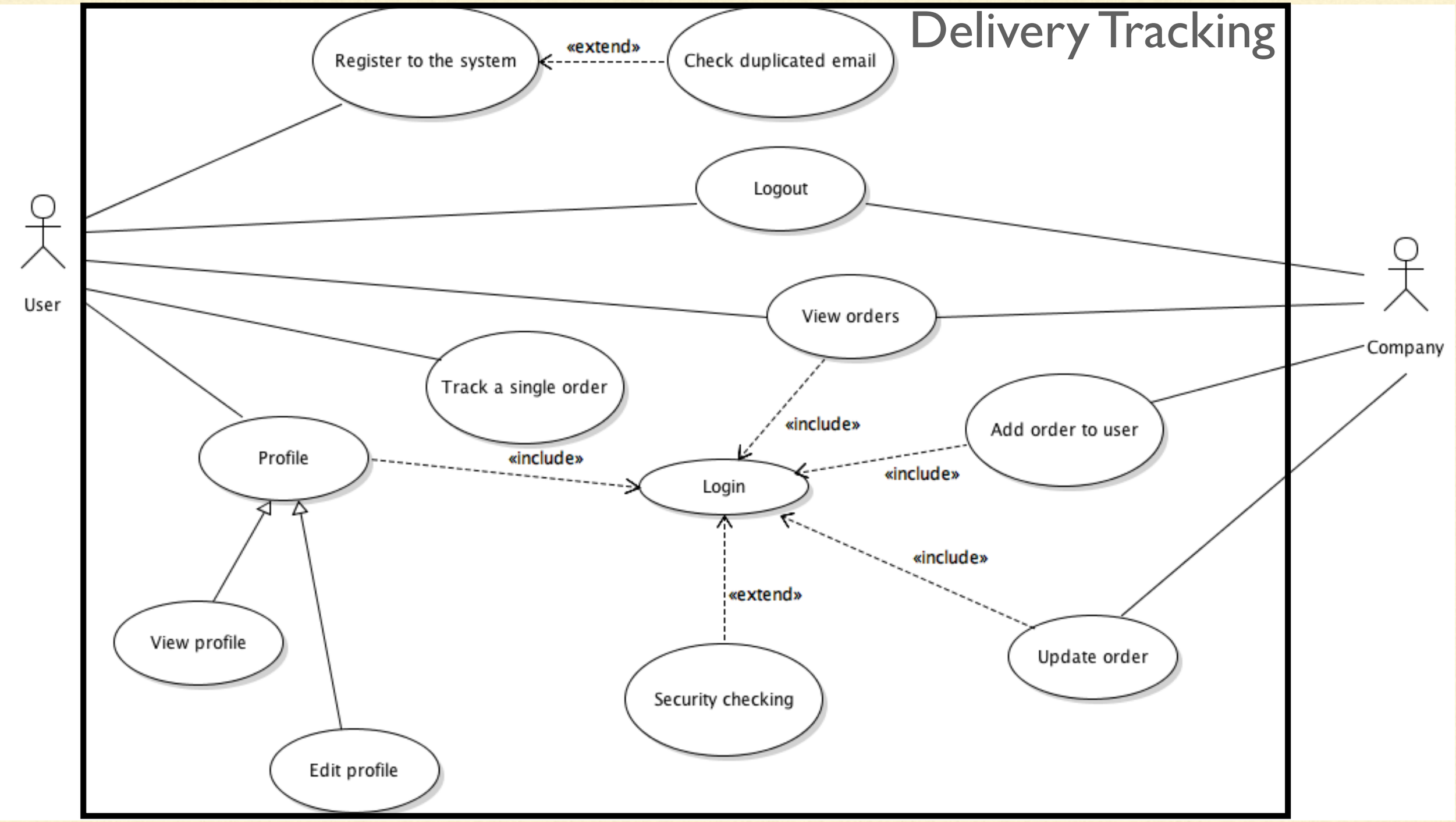
Test Driven Development

SOLID & Four Pillars

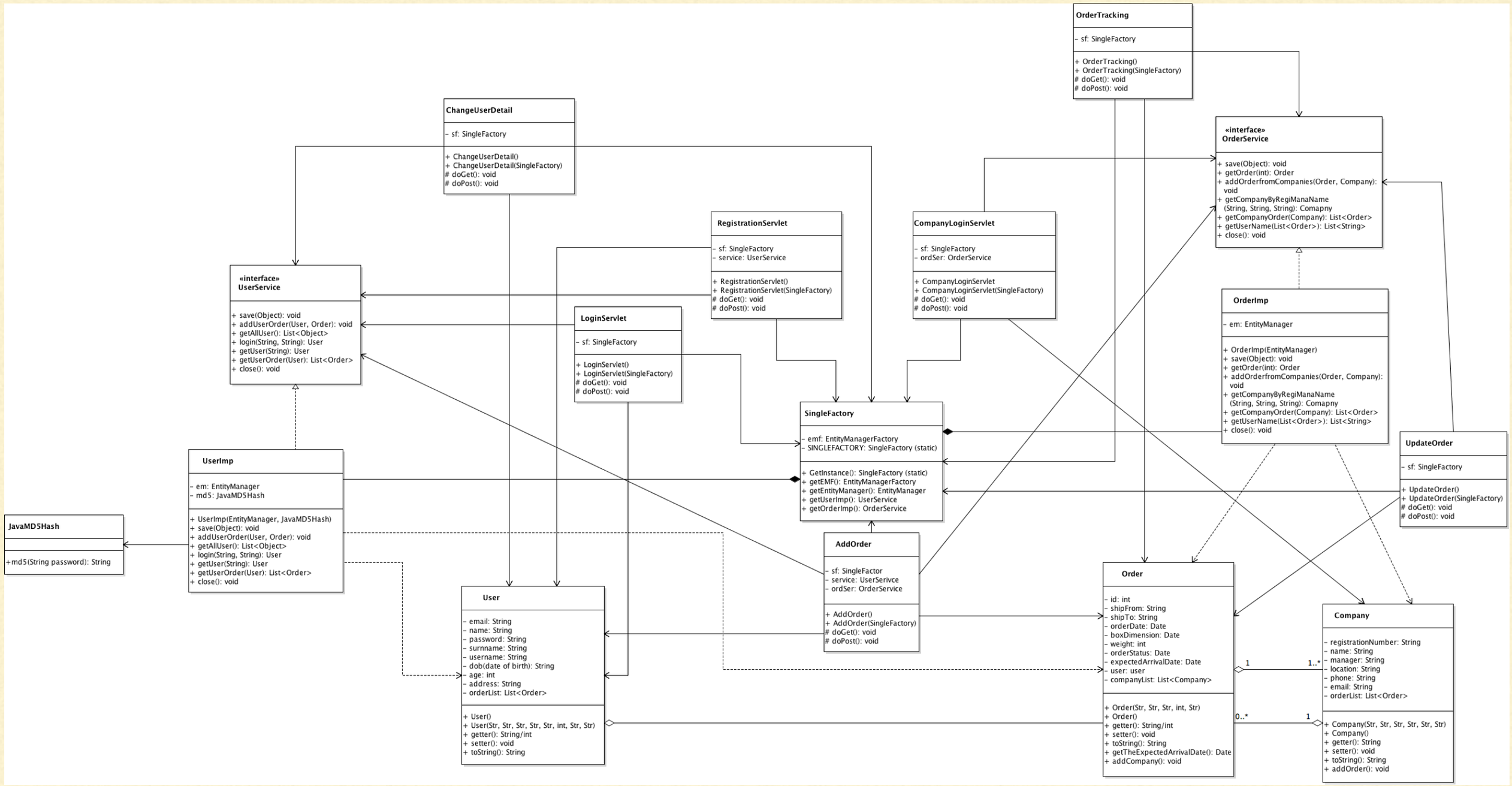
Design Pattern

Lesson Learned

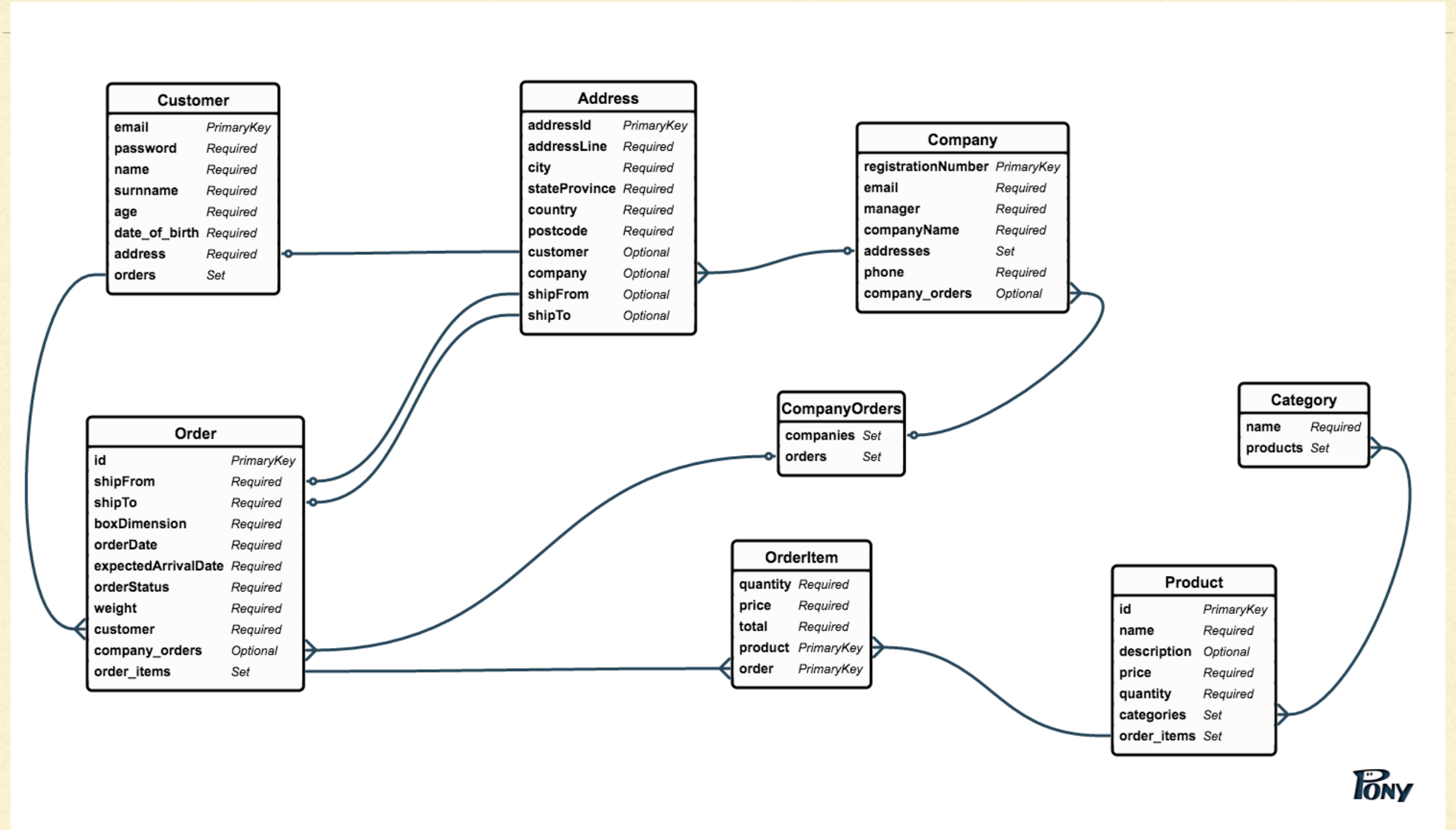
UML - USE CASE DIAGRAM



UML - CLASS DIAGRAM



UML - ERD DIAGRAM



ROADMAP

Live Demonstration

Introduction

My Approach

UML Diagrams

Test Driven Development

SOLID & Four Pillars

Design Pattern

Lesson Learned

TEST DRIVEN DEVELOPMENT

- JUnit4 and Mockito are the framework used to test the whole system;
- Has been implemented for all the models and data access objects, and servlets;
- Easy for models, but required constant rewriting for other classes;
- The coverage rate has reached 88% for 104 test cases;
- Using Mockito to mock the database and HttpRequest and Response;
- The use of Mockito is a time-consuming process since we have to mock every single object in that class;
- It is hard to mock static methods;
- Not available for GUI.

ROADMAP

Live Demonstration

Introduction

My Approach

UML Diagrams

Test Driven Development

SOLID & Four Pillars

Design Pattern

Lesson Learned

SOLID PRINCIPLE

Single responsibility

- One controller is doing one job, e.g.:
  - LoginServlet
  - RegisterServlet
  - CompanyLoginServlet
- Model classes, e.g.:
  - User
  - Order
  - Company

SOLID PRINCIPLE

```
public enum DeliveryStatus {
    ARRIVED,
    PENDING,
    DELIVERING,
    CANCELLED;

}
```

Open/Closed Principle

- Enum class stores the Delivery Status
  - Possible Delivery Status can be easily added to the system and show on the website;
  - There are only one header and one footer jsp. All pages share the same header and footer.

SOLID PRINCIPLE

Liskov substitution

Interface Segregation Principle

- There are three interfaces: UserService, OrderService, and CompanyService;
  - Each Interface only contains the methods that are specific for similar purpose so that classes/users will never implement unnecessary methods that they don't need.

SOLID PRINCIPLE

Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions.

  Interface - Concrete Class

  - UserService - UserImp
  - OrderService - OrderImp
  - CompanyService - CompanyImp
- UserService service = new UserImp();

FOUR PILLARS

Abstraction

Polymorphism

Inheritance - the use of interface;

Encapsulation - Interface inversion;