# **Harry-Technical Interview Part1**

## **\*\*\*\*\*\*Application Study & Requirement Summary**

After studying the web & application, I have made a draft version of the requirements, Which contains the main functionalities of the system

* Learning Page
* Online Restaurant eCommerce System(B2B)
  + User Management (Register, Login)
  + Marketplace (Shows foods and menu)
  + Order Management (Cart, Single & Group Ordering & Checkout)
  + Payment System (Integrate with Stripe)
  + Website + \*\*\*\*Mobile App
* Catering eCommerce System (B2C)
  + User Management (Register, Login)
  + Marketplace (Show foods and menu)
  + Order Management (Order & Checkout only)
  + Payment System (Integrate with Stripe)
  + Website only
* Internal Operation System (No information regarding this, so this is only my guess)
  + Order & Delivery Management
  + Menu & Food Management
  + Cooking & Inventory Management
  + (Optional) Reporting System
  + (Optional) Food/Menu Recommendation System.

## **Application Architecture Design (Question 2)**

Assume that we will only focus on implementing Online Restaurant eCommerce System (B2B), not the Catering one.

### Component & Requirements

The essential modules that need to be done are

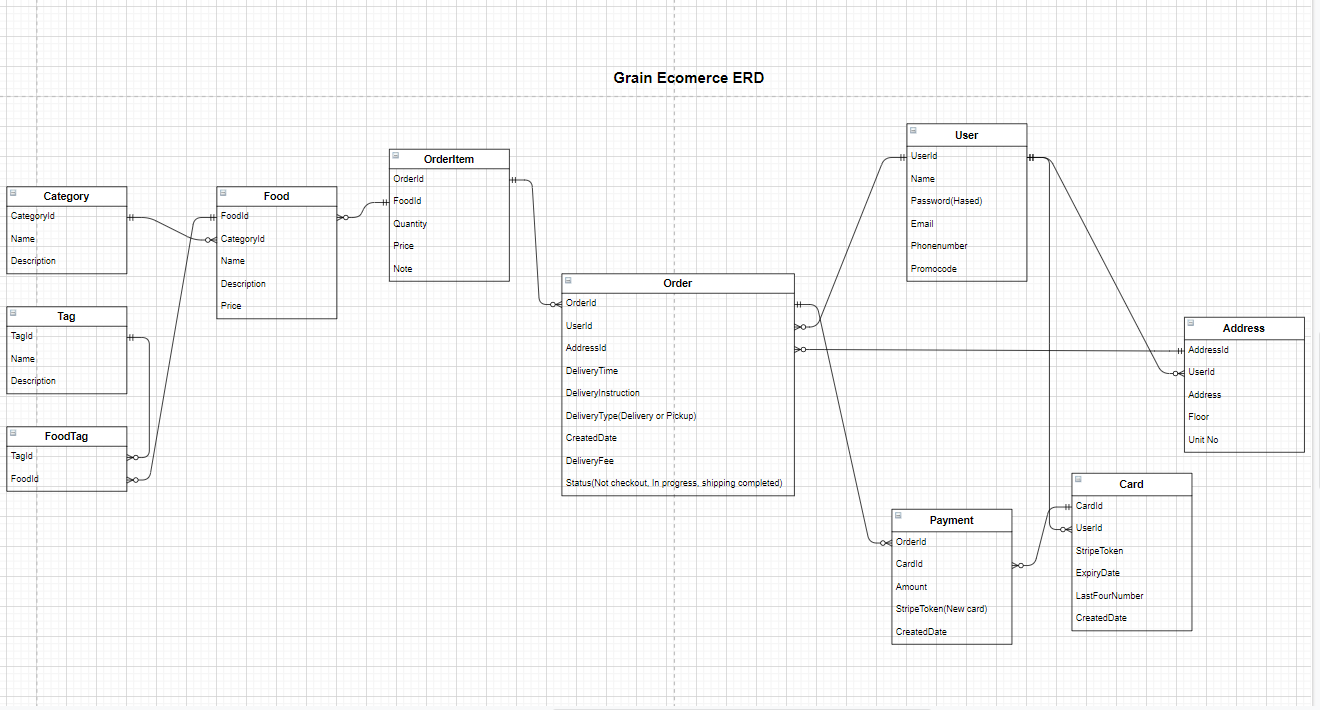
* Online Restaurant eCommerce System (B2B)
  + User Management (Register, Login)
  + Order Management (Cart, Single & Group Ordering & Checkout)
  + Payment System (Integrate with Stripe)
  + Website + \*\*\*\*\*Mobile app
* Internal Operation System
  + Order & Delivery Management
  + Menu & Food Management

### DB Design

I made simplified ERDs that contain only essential fields. To simplify the question, I ignore the tables related to our internal operation system (delivery and food management)

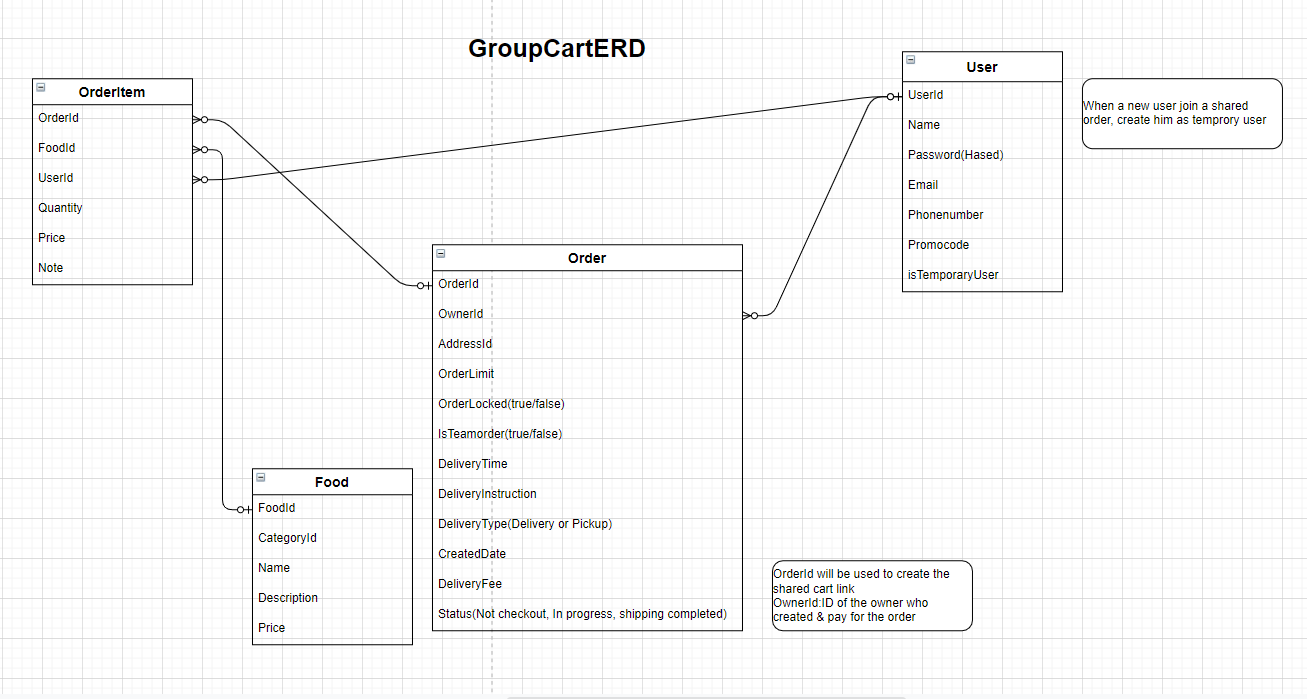
eCommerce (Big image: <https://imgur.com/sWrVB1m>)

..



Group Cart Ordering – Add additional fields to existing ERD, remove the duplicated tables in first section (Big image)

..



API Design

Public API

|  |  |  |
| --- | --- | --- |
| Endpoint | Output | Additional Note |
| POST | Register new user |  |
| POST | Login into the system, return an access token | There should be rate-limit on this API to prevent password  guessing |
| GET | List of foods grouped by categories |  |
| GET | List of available delivery time slot |  |
| GET | A single order with food information, delivery time, total | For Group Ordering |
| PUT |  | For updating item in cart in group odering |

Authenticated API

|  |  |  |
| --- | --- | --- |
| Endpoint | Output | Additional Note |
| GET/ cards  POST/ cards  GET/ cards(cardId)  PUT/ cards(cardId) |  | For manage payment and credit cards |
| GET/ addresses  POST/ addresses(cardId)  GET/ addresses(cardId) |  | For message addresses |

### Realtime Group Cart Ordering Flow

Currently, the Group Order is updated in real time. It means when a user updates an item in the order, others will be notified. This can be achieved using real-time technology (SignalR, Socket.IO, ActionCable, as mentioned in section 3.2)

Here is a simple flow of how that should work:

1. When the owner creates a group order, change the order ‘isGroupOrder’ to true and ‘isLocked’ to false. Then create a channel/group, subscribe the current user to that group
2. When another user click on the shared link, subscribe them to the same group
3. Syncing the change between them is troublesome, so when a user updates the order, we can publish a ”sync” signal to all users.
4. After receiving the “sync” signal, these users will refresh the order by calling GET/order/(orderId). The order stored in DB is the single source of truth
5. When the lock the order, change ’isLocked ’ to false. Other users can not make any edits anymore

For question 4,5,6 (Question 3 missing), I will make the following assumption about our resource & business priorities and technical priorities:

* Human Resources: small team with 2 designers and 4-5 engineers, no dedicated sysadmin/DevOps
* Business Priorities: Features should be developed incrementally, following Lean Startup Principal (thelearnstartup.com).We will try to release MVP first to get user feedback, then improve or pivot based on the result, Feature does not need to be perfect, just work well enough so we can verify the market & user’s need.
* Technical Priorities: The code should be well-documented and maintainable (low technical debt and easy to develop new features). Currently, our bottleneck is the operation (cooking, delivery, planning), not \*\*\*\* platform, so we don’t need to worry about scalability for the time being. However, we might need to scale up as we expand into other countries.

## **High level technology choices**

### Overall Architecture Design:

Despite all the hype, I think micro-service might not be a good choice for our team yet, considering that:

* We don’t have a dedicated DevOps to set up the service orchestration and infrastructure.
* We have only 1 small team of 4-5 engineers, not multiple teams
* Micro-service architecture does not bring that much value to us, as scalability is not our issue for the time being

In my last company, we used to split the big monolith into microservices architecture. It does help us to release more frequently, with more confidence, but it also makes troubleshooting, refactoring, refactoring, and local development more difficult.

Personally, I think it’s best to just use the monolith architecture for now, until we get a bigger team, and we can find the value of migrating to microservices:

…

…

However, to prepare for the future, we can separate the logic into modules. If the need arises, we can split the recommendation or order processing into a different module, or separate the online restaurant & catering into different services.

### Technology choices

Infrastructure

I recommend we use infrastructure provided by popular Cloud Provider (PaaS), based on the following reasons:

* Less up-font cost, we only need to pay for the resource we need, instead of spending money on big servers
* We don’t have dedicated SysAdmin/DevOps. Using cloud infrastructure, we don’t need to spend time maintaining them (Updating OS & software, monitoring, backup), and can spend time on developing features/writing bussiness logic instead.
* As our system grows bigger, we can easily scale the infrastructure.

|  |  |  |
| --- | --- | --- |
| Technology | AWS | Google Cloud Platform |
| Web application | Elastic Beanstalk | Google app Engine |
| Load Balancer | Elastic Load Balancing | Cloud Load Balancing |
| CDN – For deloying front-end and assets(images, files,...) | Amazon S3 | Google Cloud Storage |
| DNS & Domain Management | Amazon Route 53 or Cloudflare | Google Cloud DNS or Cloudflare |
| Database | See below | |

**Back-end technology**

I am most familiar with .NET and NodeJs technologies, so my technology recommendation is based on that. In reality, we should choose the technology based on the experience & technical skill of the current engineering team.

|  |  |  |
| --- | --- | --- |
| **Technology** | **.NET** | **Nodejs** |
| Web application | ASP.NET Core | ExpressJS (Battle-tested, very popular framwork) |
| Real-time | SignalR | Socket.io |
| ORM | EntityFramework | Sequelize or TypeORM |
| Background Task/Cron Task | Hangfire | Node-schedule or bull |
| Database (should use PaaS provided by Cloud Provider) | Microsoft SQL Server | PostgresQL or MySQL |
| Caching | Redis Cache | |
| Message Queue | Use the one provided by our Cloud Provider (Geogle PubSub or AWS SNS ) | |

**Front-end technology**

ReactJS or VueJS

|  |  |
| --- | --- |
| **ReactJS** | **VueJS** |
| Mature technology with good documentation, tutorial & tooling support | Not as popular as React, Good documentation and lower learning curve. Easy to pick up and get productive |
| Many package for routing & state management & styling => Hard to choose one for the project | Official packages: VueRouter + Vuex. Built-in single file component with styling |
| Many popular libraríes and packages | Not as popular as React |
| Big developer pool for hiring | Smaller developer hiring pool. However VueJS have how learning curve so developer can easily learn & get prodcutive |

I have used both of them. Personally I prefer ReactJs + TypeScript(the toolings and Developer Experience are really awesome)

**Mobile App**

React Native seems to be a suitable choice (I tried that before to create my own app, the experience is not too bad).

We can’t afford dedicated Android & IOS developers, so our web developers can make use of their own JS & React knowledge to build mobile apps on both IOS and Android.

Our application doesn’t not require much native code or access to the device capabilities anyway.

Altermative: NativeScript, ionic or Xamarin

## **Idel Deloyment Process(Question 7):**

I assume that we're using a hosted Git repository such as Gitlab/Github/Bitbucket, that supports Merge Request & Code Review, and able to integrate with popular Cl such as TravisCI, Gitlab Runner or CircleCl.

I propose a simplified version of git branch (This might be different based on team). Each project should have it own repository (If we have 1 front-end, 1 back-end and mobile, it means 3 repositories)

....

All merge requests/pull requests must pass the unit test in CI before it can be merged.

The deployment process should be straightforward and can be run in our Cl:

Front-end: Run 'yam build' then upload the artifact into Amazon S3 or Google Cloud Storage

Back-end: Use proper deployment tool by cloud provider 'gcloud app deploy. If we deploy to a VM, we can write a bash script to SSH to that VM and deploy: or use the correct deployment tool of that language (ex. Capistrano), that can be run in the CI

Mobile: The Cl should be able to build an artifact (.ipa file for iOS, .apk file for Android), that can be downloaded. After that, we upload the new artifact to the AppStore, along with the changelogs. This step can only be done manually. To roll back a release in emergency situations, we can use git revert in the master branch, then commit the change.

## **Implementation Plan & Estimation (Question 5,6)**

As mentioned in section 1, here are the draft requirement of the functionalities our system:

* Landing Page
* Online Restaurant eCommerce System (B2B)
  + User Management (Register, Login)
  + Marketplace (Show foods and menu)
  + Order Management (Cart, Single & Group Ordering & Checkout)
  + Payment System (Integrate with Stripe)
  + Website + \*\*\* Mobile App
* Catering eCommerce System (B2C)
  + User Management (Register, Login)
  + Marketplace (Show foods and menu)
  + Order Management (Order & Checkout only)
  + Payment System (Integrate with Stripe)
  + Website only
* Internal Operation System (No information regarding this, so this is only my guess)
  + Order & Delivery Management
  + Menu & Food Management
  + Cooking & Inventory Management (Optional)
  + Reporting System (Optional)
  + Food/Menu Recommendation System

We can separate the work into 3 phases of development.

### **Phase 1- Basic eCommerce Functionality:**

Output: In this phase, we focus on the bare-minimum functionalities that can be released. Users must be able to view food, order and checkout.

Other functionalities such as Group Cart, Mobile can wait until the next phase. Therefore, we might not have time to develop'our Internal Operation System for managing order and food. So a lot of manual work must be done here.

I assume that we want to start with online reštaurants first on this page, catering in the next phase. If the business focuses more on Catering, we can implement catering in this page,and online restaurant in the next phase.

**Ballpark estimation**

* 2-3 months for all the functionalities to be ready for testing in Staging environment.
* Another 2 weeks 1 month to do the testing & bug fix & feedback before releasing to Production
* **Totally this phase will take 3-4 months**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Resource Needed** | **Time needed** |
| Landing Page | 1 front-end engineer  1 designer | 3-5 working days |
| Setting up Project Foundation (project structure, coding convention, library, development) | Whole engineering team | 8-10 working days |
| User management(Register, Login) | 1 front-end engineer  1 back-end engineer  1 designer | 3-5 working days  5-8 working days if 1 full-stack engineer |
| Marketplace (Show foods and menu) | 1 front-end engineer  1 back-end engineer  1 designer | 5-8 working days  8-10 working days if 1 full-stack engineer |
| Order Management (Cart and checkout) | 1 front-end engineer  2 back-end engineer  1 designer | 8-10 working days |
| Payment System (Integrate with Stripe) | 1 back-end engineer  1 designer (involve very little) | 3-5 working days |

**Note**: I added 40% of buffer on my estimation, because at the beginning of the project, the engineers need to take time to setup the foundation (project structure, coding convention, library, deployment, logging), so the development speed will be slower than usual.

### **Phase 2 – Creating & Mobile App & Internal System**

-

* Totally this phase will take 2-3 months

|  |  |  |
| --- | --- | --- |
| **Feature** | **Resource Needed** | **Time needed** |
| Catering Marketplace (show foods and menu) | 1 front-end engineer  1 back-end engineer  1 designer | 2-3 working days  3-5 working days If 1 full-stack engineer |
| Catering Order Management(cart and checkout) | 1 front-end engineer  1 back-end engineer  1 designer | 3-5 working days  5-8 working days If 1 full-stack engineer |
| Catering Payment Sytem(Integrate with Stripe) | 1 back-end engineer  1 designer (involve very little) | 2-5 working days (Might be faster if we can resue the integration for online restaurant) |
| Mobile App (Basic Functionality  As web phase 1) | 1 or 2 back-end engineer  2 font-end engineers | 10-15 working days (we should be able to resuse our RestAPI for mobile) |
| Internal Order Delivery Management | 2 back-end engineer  1 designer | 8-10 working days |
| Internal Food & Menu Management | 1 front-end engineer  1 back-end engineer  1 designer | 3-5 working days  5-8 working days if 1 full stack engineer |

### **Phase 3 -Additional Improvements**

Output: In this phase, we focus on the new features that can improve our user experience and increase our efficiency.

Ballpark estimation:

* estimation 1.5-2 months for Catering functionalities to be ready for testing in Staging environment.
* Another 2 weeks - 1 month to do the testing & bug fix & feedback before releasing to Production
* We will continue this phase until we got another big project or further improvement

|  |  |  |
| --- | --- | --- |
| **Feature** | **Resource Needed** | **Time needed** |
| Group Cart Ordering | 1 front-end engineer  1 back-end engineer  1 designer | 8-10 working days |
| Mobile App (Addtional Functionalities) | 1 or 2 designers  2 font-end engineers | 10-15 working days  (We should be able to resue our RESTAPI for mobile) |
| Internal Operation System Improvêmnt | 1 front-end engineer  1 back-end engineer  1 designer | Unknown |