티끌모아

Final Report
2018. 6. 15 (version 2.0)

Kim Shin Hwi
Kim Ji Hoon
Seol Jae Wan
Yu Geun Gook

Abstract

"티끌모아" will provide a service for people to manage their coupons. Coupons may come from various places (restaurant, cafe...), but many people can not use those coupon because they often lost those. So we want to make online coupon management service.

Here is our catchphrase :

"I want to eat one free drink per 10 drinks!"

There are three main parts: giving stamp to online coupon, showing how many coupon customer has, using collected coupon.

- Giving stamp to online coupon (by store only) : Store account can give stamp to customer's coupon. Store account uses customer's phone-number to give stamp (If phone-number isn't registered in our service, stamping would be fail). When customer comes this store at first, coupon will be created in customer screen with stamp.

- Showing how many coupon customer has (by customer and store both) : Coupons are created by each store. So customer can see which store they have coupons of, and how many stamp in this coupon. Also, Store account can see only its coupon of customer.

- Using collected coupon (by store only) : Coupon must be used by store with customer's permission. When store account use customer's coupon, he must put customer's phone-number for searching.

Motivation & Problem

We want to provide solution about problem of paper coupon and membership service

vs Paper coupon

- Permanency : paper coupon can be damaged and lost easily. But our online-coupon is always with you
- Portability : we must prepare paper coupon in advance. But we cannot prepare paper coupon always. Our online-coupon is always with you if internet is connected.
- Transfer : In case where 5 people have only 3 stamps each but minimal stamps that can be used at is 10, each stamp has no value. But if each customer can give their stamps to another customer, stamps can get value. Furthermore, customers may exchange their coupons!

vs Membership service(mileage)

Membership service is only provided for frequently visiting customers. Customers who do not visit store frequently can not get benefit with membership service.
For example, a person uses certain cafe once a month. He will not always prepare coupon for this cafe. So he can't get benefit of coupon.
But we can provide that benefit (even after 10 months) !

Related Work

-Kakao-pay membership
Kakao-pay provides service that all membership point can be managed by one membership. As it was before, we have to bring each membership card or find each barcode to use membership point(Happy-point, CJ-one, L-point, SINSEGAE-point). But in this service, only one barcode from kakao-pay can manage all those point.

Functionality

1. Customer can see all of his/her coupons from various stores on one window.
    - Coupon list can be filtered by name of store.

2. Store can manage all it's coupons on one window.
    - The administrator can see all coupons on one window, and the coupon list should be filtered by name of customer.
    - Store's administrator can give coupons to each customers.
    - The administrator can take coupons up from each customers.

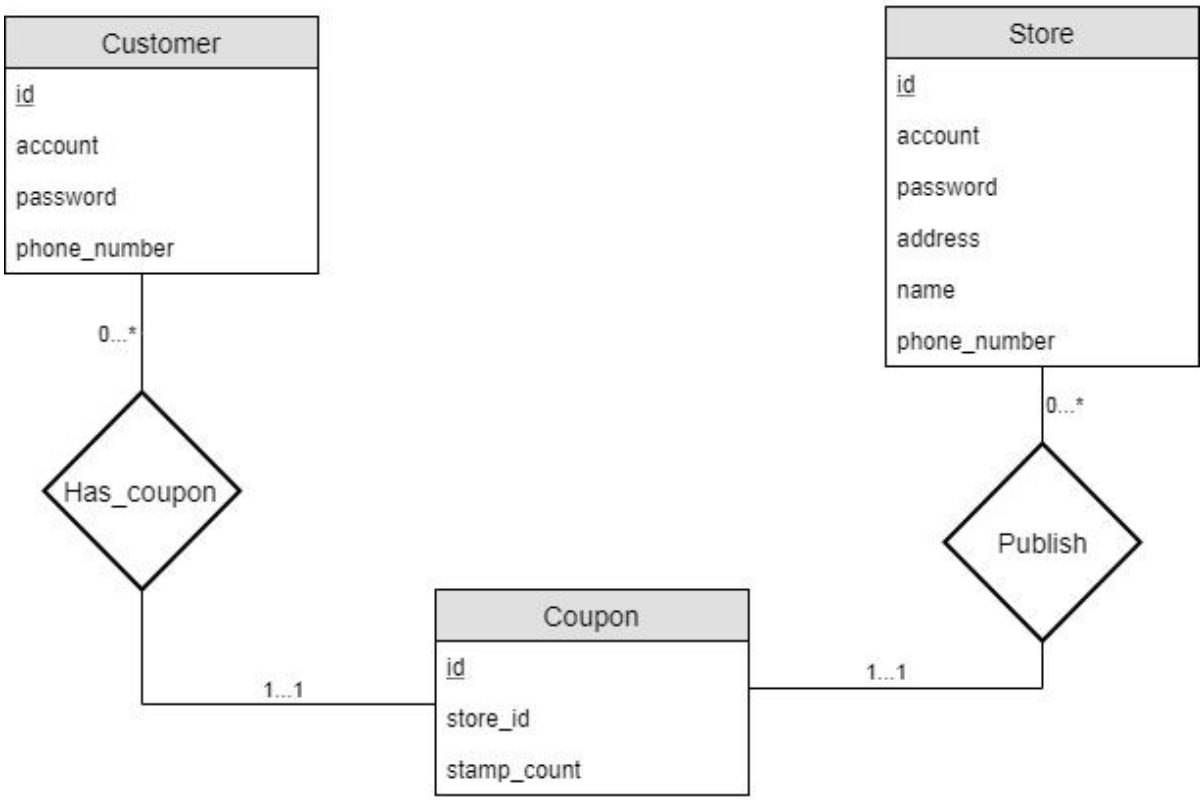
3. Customer can give his/her coupons to other customers.

Design & Implementation

1. Backend

1.1. Model

- E-R Diagram

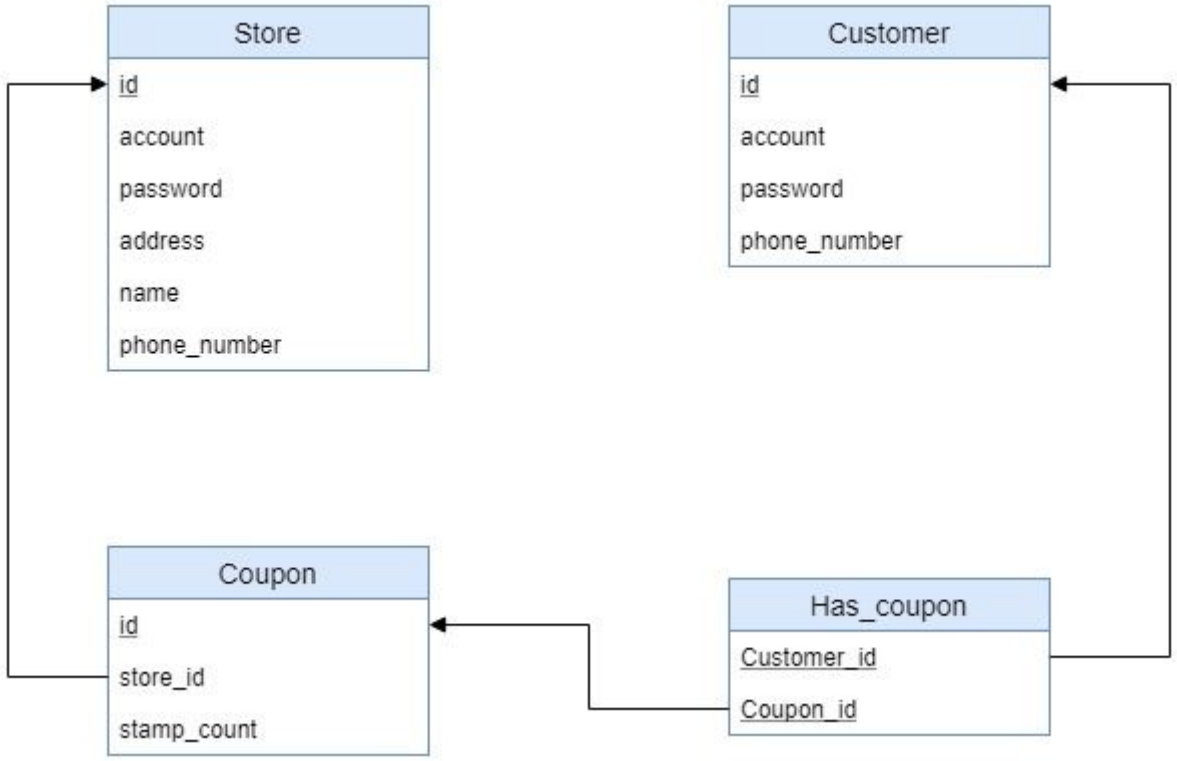


In E-R diagram, rectangle means entity set, diamond means relationship set and numbers above line represent cardinality constraints. All attributes are listed inside each rectangle. Underlined attributes are primary key of each entity set.

There are three entity sets and two relationship sets. Each ‘customer’ and ‘store’ entity stands for customer account and store account. Users sign in with ‘account’ and ‘password’ attributes. ‘Coupon’ entity has relationship with ‘customer’ entity and ‘store’ entity. ‘Has_coupon’ relationship has information about which customers have which coupons. 'Publish' relationship indicates which coupons were issued in which stores.

A customer can have no coupon or can have many coupons. Likewise, a store can publish no coupon or can publish many coupons. Therefore, cardinality constraints should be 0…*. On the other hand, a coupon has to have only one owner(customer) and is published by certain store. Therefore, cardinality constraints should be 1...1.

- Relation schema diagram



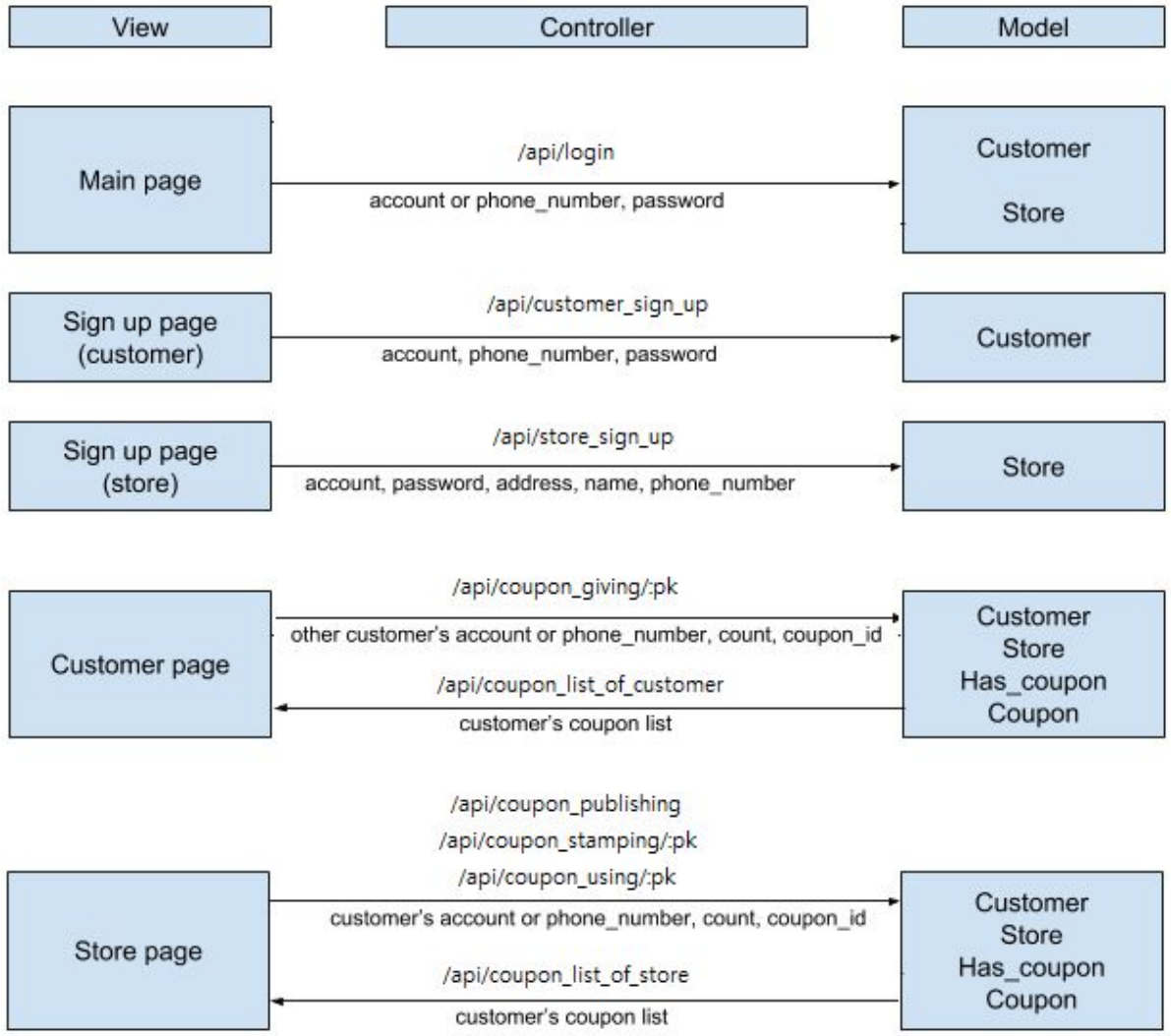
Based on E-R diagram above, relation schema diagram can be made like this. Rectangle means relation schema, and all schema attributes are listed inside each rectangle. Underlined attributes are primary key of each relation, and arrow stands for foreign key constraints.

Converting E-R diagram to relation schema diagram, ‘publish’ relationship set is simply represented by foreign key constraint. ‘Coupon’ relation has ‘store_id’ that is foreign key attribute pointing ‘id’ of ‘store’ relation.

On the other hand, the ‘Has_coupon’ relationship set is converted to a relation. It represent relationship between ‘customer’ and ‘coupon’ by having ‘customer_id’ and ‘coupon_id’ attributes.

1.2. Controller

Here is controller design.



Left side is view part (frontend) and right side is model part (backend). Left-to-right arrow represents http-request with user inputs from view, and right-to-left arrow represents http-response with data from model. Above the arrow, there is an API that controller uses to transfer JSON data below the arrow

1.3. API

- Specifications of RESTful APIs

Model	API	GET	POST	PUT	DELETE
Customer, Store	/api/login	X	validate log in	X	X
-	/api/logout	logout	X	X	X
Customer	/api/customer_sign_up	X	create new customer	X	X
Store	/api/store_sign_up	X	create new store	X	X
Coupon, Has_coupon	/api/coupon_publishing	X	create new coupon	X	X
	/api/coupon_list_of_customer	get coupon list	X	X	X
	/api/coupon_list_of_store	get coupon list	X	X	X
	/api/coupon_stamping/:pk	X	X	add 1 to count of stamp	X
	/api/coupon_using/:pk	X	X	subtract given number from count of stamp	X
	/api/coupon_giving/:pk	X	X	sub given number from count of stamp of one's coupon, and add same number to count of stamp of another's coupon	X

- Permissions

Coupon Publishing : logged-in Store account
Coupon List of Customer : logged-in Customer account
Coupon List of Store : logged-in Store account
Coupon Stamping : logged-in Store account
Coupon Using : logged-in Store account who is coupon owner
Coupon Giving : logged-in Customer account who is coupon owner

1.4. Implementation

- Develop Environment

django 2.0.5
django-rest-framework 3.8.2
python 3.6.4

- Django static file serving

We use static file serving from django. When user come our web-site first, django give static file(html file) to user. - static file was made from npm build in frontend develop. and page-switching occur by react-router

- Generics-View in Django rest framework

We develop api-view with generics-view from django rest framework. When we have to join tables from database, use serializers to join tables.

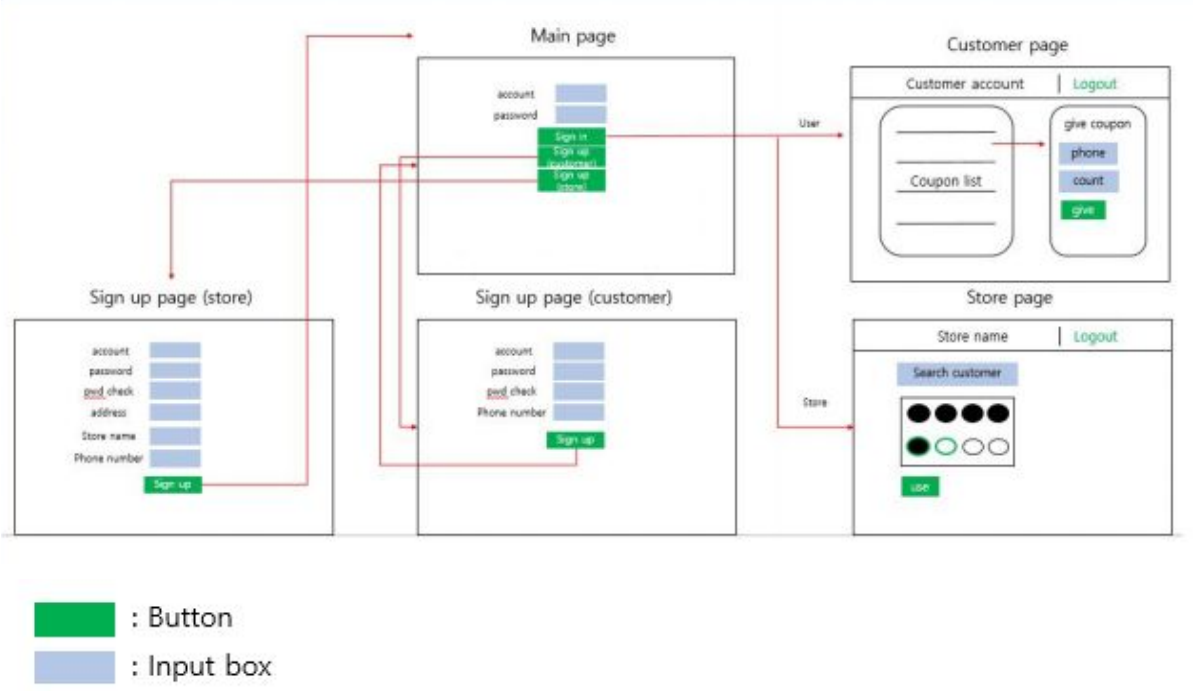
- Django Login

We develop login-api with overriding django login view. After login success, server set session-id. logout-api is also developed with django logout view.

2. Frontend

2.1. Design

- Pages



Main page (“/”)

- Sign in
- Move to sign up page

Sign up page - customer (“/sign_up_customer”)

- Get account, password, password check, phone number as input
- Sign up

Sign up page - store(“/sign_up_store”)

- Get account, password, password check, address, name, phone number as input
- Sign up

customer page (“/customer”)

- List my coupons
- When click a coupon, show ‘give’ window
- In ‘give’ window, get target’s phone number and count and transfer my coupon

store page (“/store”)

- Search customer by phone number or account
- Show the customer’s coupon status
- Give or delete coupons
- Use coupons

- Components

We have 5 pages and each page has components and methods.

main
account: input password: input signIn: button signUpCustomer: button signUpStore: button memberStoreList: listComponent
onClickSignIn onClickSignUpCustomer onClickSignUpStore

sign up store	sign up customer
account: input password: input passwordCheck: input address: input name: input phoneNumber: input signUp: button	account: input password: input passwordCheck: input phoneNumber: input signUp: button
onClickSignUp	onClickSignUp

customer page	store page
customerAccount: text logout: button couponList: couponListComponent .phoneNumber: input .count: input .give: button .components are shown only when onClickToggleUp	storeAccount: text logout: button customerAccount: input .couponPanel: couponPanelComponent .components are shown only when onClickSearch
onClickToggleUp onClickToggleDown onClickLogout onClickGive	onClickSearch onClickLogout onClickUse

2.2. Implementation

We used node.js 9.8.0 version for implementing our service frontend. Basically, we used react concept. In addition, concepts like react-router, redux, static storage are used for Single-Page-Application(SPA).

- Development Environment

node.js - 9.8.0
npm - 5.6.0
react - 16.4.0

- React-Router - 16.4.0

We do not serve page URL by web server application, but serve by Django static file serving. We implemented one html file as SPA, which can renders multiple pages, so that Django serve just one file to user's browser. To make just one file render multiple pages, we used React-Router.
Specifically, we set <BrowserRouter> tag inside 'index.js', which can be called root page of page tree, to make this file render multiple pages. And inside 'App.js', we assigned proper URL to each page components to be routed.

- Redux - 4.0.0

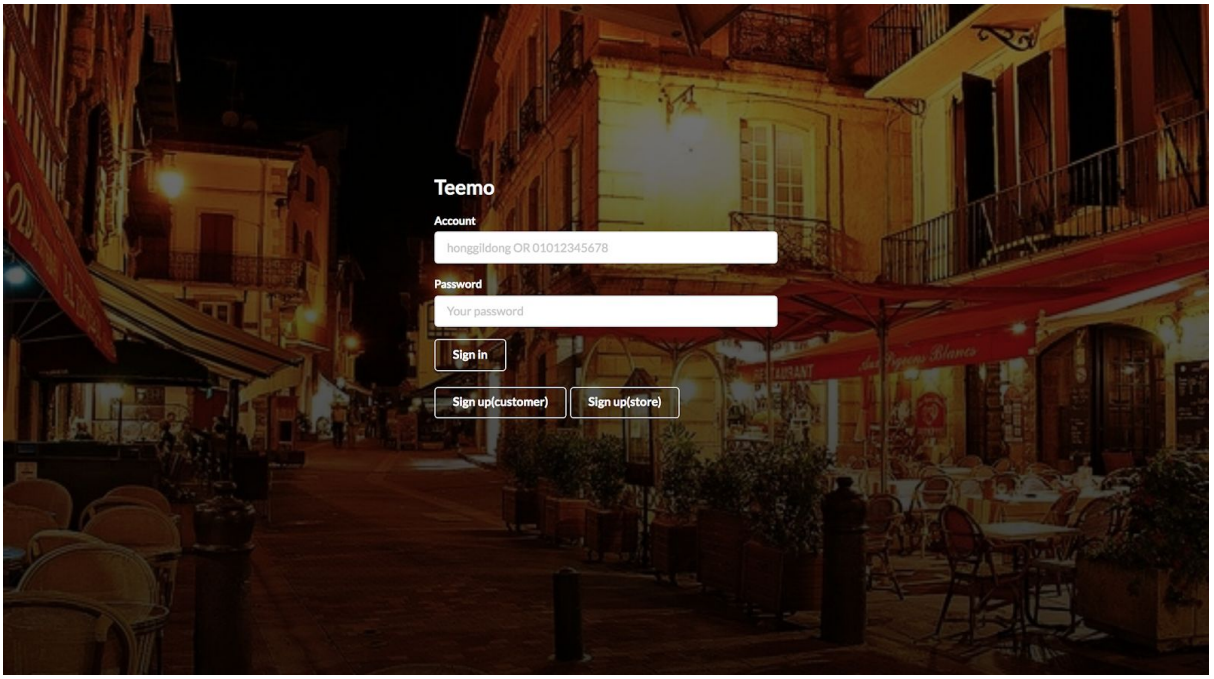
We implemented SPA using react. It was not easy to use only native react. So we introduced redux to make our work easy.
Using redux, we set a single integrated store, allowing each page component to manipulate and share its store via action dispatching.
We make container and store folder in src folder, and we connected components and redux concepts in container.
We surrounded root component with <Provider> tag in index.js as we did the case of react-router. To avoid collision with react-router, we followed the ordering of tag <Provider> and <BrowserRouter>, and we used withRouter method.
Specifically, we stored login information, user type, result of searching, query information as global state. So every components can access global state easily and use these information to compute or render something.

- Static storage

We used redux concept. But redux concept had a problem that it flushes all states when the user refresh web browser. We maintained many important states that must not be deleted (for example, flushing states make redirection to random page for lossing login information.), So it needed to be persistent.
To make our states persistent, we used local storage of web browser. We stored states in local storage not only in temporary memory.
We defined StateLoader class. Everytime states were changed, stateloader copies states to local storage.

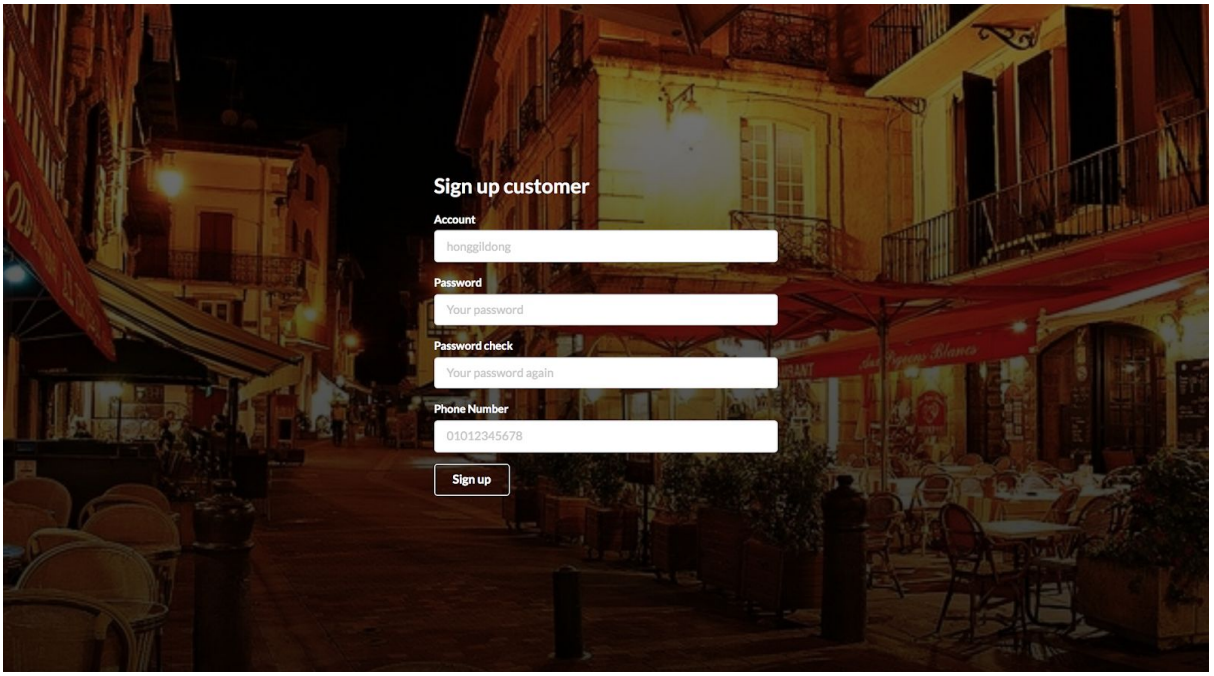
UI/UX

1. Main page



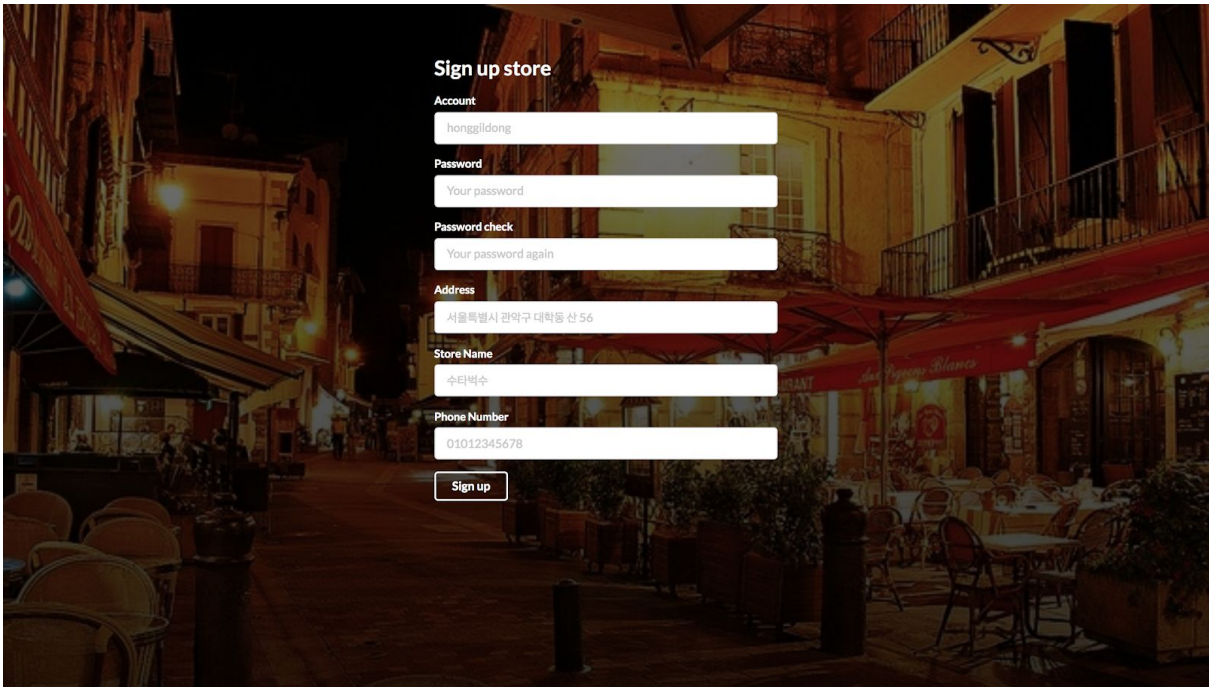
login with account, password.

2. Sign up page (customer)



If Account or Phone number already exist, you can’t sign up

3. Sign up page (store)



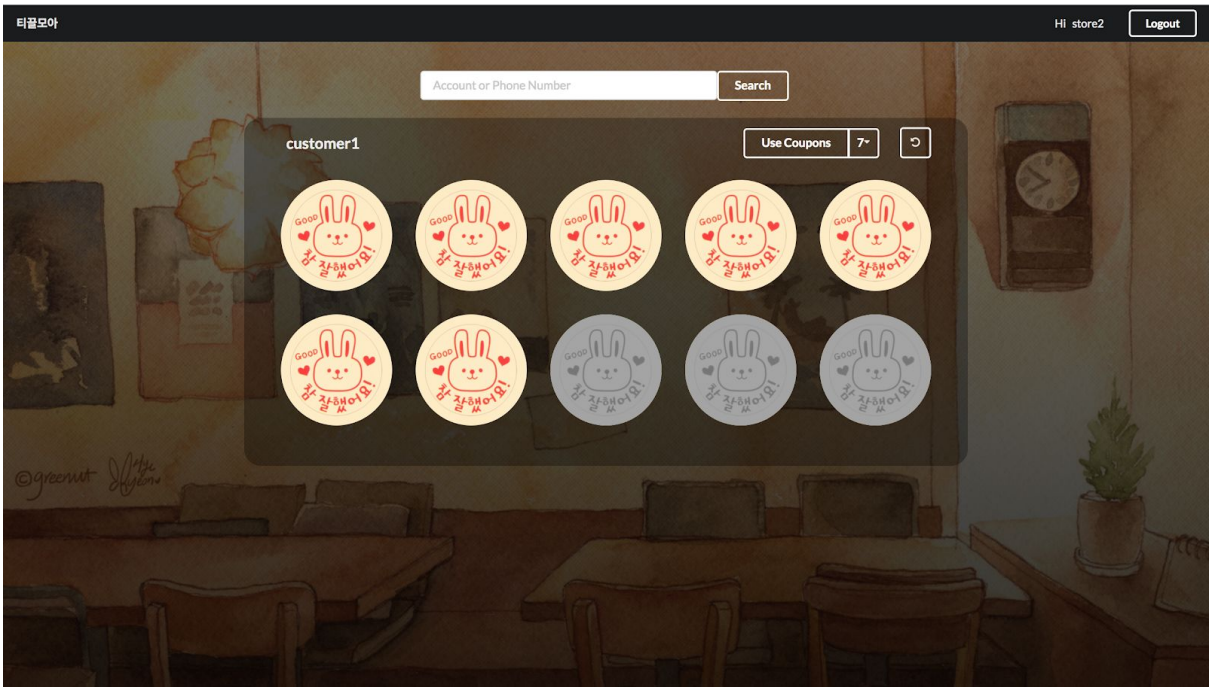
If Account or Phone number already exist, you can’t sign up,

4. Customer Page

List of coupons whose owner is the logged-in customer.

Coupon giving panel. You can give your coupons to other customers.

5. Store Page



You can search a customer there.

Coupons whose owner is the current-searched customer. You can give a stamp to the customer by click an empty circle.

Store can use for taking coupons up from current-searched customer. You can select how much stamps taking up.

Conclusion

We developed a web application which manages and integrates customer's coupon. Actually, we don't have any experience about developing web application. So, many problems are unfamiliar, and need a lot of time to know how to fix.
First, login is our big issue in project. All of our team member didn't know how to implement login and how login concept works. After understanding login concept, implementation was different problem. CORS, CSRF-Token, SESSION-id.. all of them were not easy.
Next, It is difficult to design frontend user interface because we were computer engineers, not art designers. Because we had no experience to design something, we have to work hard to make interface look pretty.
Finally, we completed all we want. Experience in developing end to end gives us many distress. But also give us many pleasure. And It help us to understanding software development. We had many valuable time in this project !