원격 입장 관리 시스템

QMS(QUEUE MANAGEMENT SYSTEM)

133071 조민우

OBJECTIVE

- 코로나로 인해 사회가 비대면을 선호하는 분위기로 급격하게 바뀌는 상황이다.
- 비대면으로 해결할 수 없는 각종 상황들에서 간단하게 사용할 수 있는 원격 대기관리 어플리케이션을 만들어본다.
  EX) 음식점 줄 서기, 은행 창구, 상담 등.

TECHNOLOGIES USED

- Ruby On rails - API server / WebSocket Server
  - DB: PostgreSQL, cache/websocket: Redis
  - TEST: Rspec
- React
  - UI: Material-UI
- Deploy: Elastic Beanstalk, S3

DEVELOPMENT PROCESS

- 1. 개발 환경 SETUP
  - AWS 계정 생성
  - Elastic Beanstalk를 이용한 EC2 인스턴스 생성 및 배포 준비
  - S3 버킷 생성 및 배포 준비.
- 2. TDD(Test-Driven-Development)를 기반으로 한 API 개발
- 3. Material-UI를 이용한 빠른 프로토타입의 WEB 개발.
- 4. API 서버 / React App 배포 및 연동.
  - 서버 배포 및 AWS 설정
  - React App과 서버의 연동 및 동작 테스트

TEST DRIVEN DEVELOPMENT

- 테스트 주도 개발 = 테스트가 개발을 이끌어 나간다
- Test, before Development
  - 코드를 만들기 전에 테스트 케이스를 먼저 작성하는 것
- 장점
  - 코드가 깔끔해진다.
    - 기능은 테스트 케이스로 정리, 개발자는 구현에 집중할 수 있다.
  - 리팩토링이 쉬워진다.
    - 리팩토링이 어려운 이유 -> 기존 코드의 동작을 검증하기 어렵기 때문
    - 테스트 코드가 있다면 검증이 가능하기 때문에 리팩토링을 쉽게 할 수 있다.
    - 즉, 유지보수가 쉬운 코드가 된다.
  - 작은 테스트 단위 -> 코드의 모듈화를 쉽게 한다.

TDD란?

```ruby
describe QueueItem, 'relations' do
  it { is_expected.to belong_to(:user) }
  it { is_expected.to belong_to(:queue_container) }
end
```

```ruby
describe QueueContainerSetting, 'relations' do
  it { is_expected.to have_many(:queue_managers) }
end
```

```ruby
describe QueueContainer, 'relations' do
  it { is_expected.to belong_to(:queue_container_setting) }
end
```

```ruby
describe QueueContainer, '#ranked_items' do
  let(:queue_container) { FactoryBot.create :queue_container }

  subject { queue_container.ranked_items }

  before { 5.times { FactoryBot.create :queue_item, queue_container: queue_container } }

  it { expect(subject.length).to be 5 }
end
```

```ruby
context '3번째 아이템을 6번째로 move' do
  let(:target_queue_item) { queue_item_02 }
  subject { queue_container.move(target_queue_item, to: 5) }

  it '1, 2, 7번째 rank는 그대로. 4, 5, 6번째는 rank가 하나씩 줄어든다.' do
    subject

    expect(queue_item_00.item_order_rank).to eq 0
    expect(queue_item_01.item_order_rank).to eq 1
    expect(queue_item_02.item_order_rank).to eq 5
    expect(queue_item_03.item_order_rank).to eq 2
    expect(queue_item_04.item_order_rank).to eq 3
    expect(queue_item_05.item_order_rank).to eq 4
    expect(queue_item_06.item_order_rank).to eq 6
    expect(queue_item_07.item_order_rank).to eq 7
  end
end
```

TDD란?

```ruby
describe QueueItemsController, 'POST #create' do
  let(:user) { FactoryBot.create(:user, name: 'test_user', email: 'testing@gmail.com', password: 'test_password') }
  let(:queue_container_setting) { FactoryBot.create :queue_container_setting }
  let(:queue_manager) { FactoryBot.create :queue_manager, user: user, queue_container_setting: queue_container_setting }
  let(:queue_container) { FactoryBot.create :queue_container }
  subject { post :create, params: { queue_container_id: queue_container.id, queue_item: { status: :queued }, format: :json } }

  context '로그인 되지 않은 유저일 경우' do
    it 'Unauthorized Error ( 401 ) 을 내려준다.' do
      subject
      expect(response.code).to eq('401')
    end
  end

  context '로그인 된 유저일 경우' do
    before { authenticate_user(user) }

    it '정상적인 응답을 내려준다' do
      subject
      expect(response.code).to eq('200')
    end

    it 'QueueItem 가 하나 생성된다.' do
      expect { subject }.to change { QueueItem.count }.by 1 # 1 만큼 변하는 것을 확인하는 것
    end

    it '응답은 queue_item_serializer 를 사용한다.' do
      subject
      expect(JSON.parse(response.body, symbolize_names: true)).to eq QueueItemSerializer.new(QueueItem.last).as_json.deep_symbolize_keys
    end
  end
end
```

TDD란?

```ruby
class QueueItemsController < ApplicationController
  before_action :ensure_authenticate_user!

  load_resource :queue_container, except: %i[index]
  load_resource :queue_item, through: :queue_container, except: %i[index]

  load_and_authorize_resource except: %i[index create]

  def index
    @queue_items = current_user.queue_items
    render json: @queue_items.queued, each_serializer: QueueItemSerializer
  end

  def create
    queue_item = @queue_container.enqueue(current_user, queue_item_params)

    render json: queue_item, serializer: QueueItemSerializer
  end

  def show
    render json: @queue_item, serializer: QueueItemSerializer
  end

  def destroy
    @queue_container.dequeue(@queue_item, false)

    payload = QueueItemSerializer.new(queue_item).as_json
    QueueManagersChannel.broadcast_to @queue_container, { type: 'dequeued', payload: payload }

    render json: @queue_item, serializer: QueueItemSerializer
  end

  private

  def queue_item_params
    params.require(:queue_item).permit(:status, :note).as_json.deep_symbolize_keys
  end
end
```

TDD란?

```ruby
class Ability
  include CanCan::Ability

  def initialize(user)
    user |= User.new
    #
    # QueueManager
    #
    can :manage, QueueManager, user_id: user.id
    can :read, QueueManager, queue_container_setting: { users: { id: user.id } }
    can :manage, QueueManager, queue_container_setting: { owner_manager: { user_id: user.id } }
    #
    # QueueContainerSetting
    #
    can :read, QueueContainerSetting, queue_managers: { user_id: user.id }
    can :manage, QueueContainerSetting, owner_manager: { user_id: user.id }
    #
    # QueueContainer
    #
    can :read, QueueContainer
    can :manage, QueueContainer, queue_container_setting: { users: { id: user.id } }
    #
    # QueueItem
    #
    can :manage, QueueItem, user_id: user.id
    can :manage, QueueItem, queue_container: { queue_container_setting: { users: { id: user.id } } }
  end
end
```

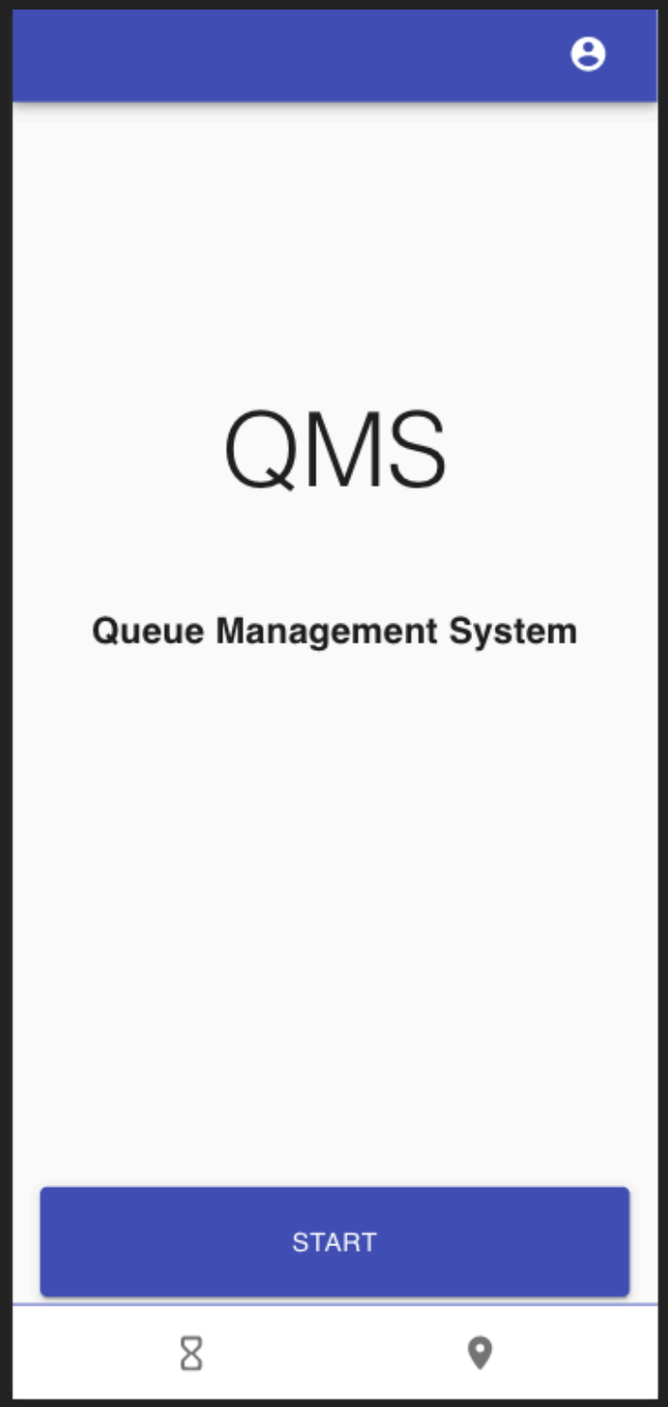
```ruby
describe User, 'normal' do
  subject(:ability) { Ability.new(user) }
  let(:user) { FactoryBot.create :user }

  let(:queue_container) { FactoryBot.create :queue_container }
  let(:queue_item) { FactoryBot.create :queue_item, user: user }
  let(:another_queue_item) { FactoryBot.create :queue_item }

  #
  # QueueItem
  #
  it { is_expected.to be_able_to(:manage, queue_item) }
  it { is_expected.not_to be_able_to(:manage, another_queue_item) }

  #
  # QueueContainer
  #
  it { is_expected.to be_able_to(:read, queue_container) }
  it { is_expected.not_to be_able_to(:manage, queue_container) }
end
```

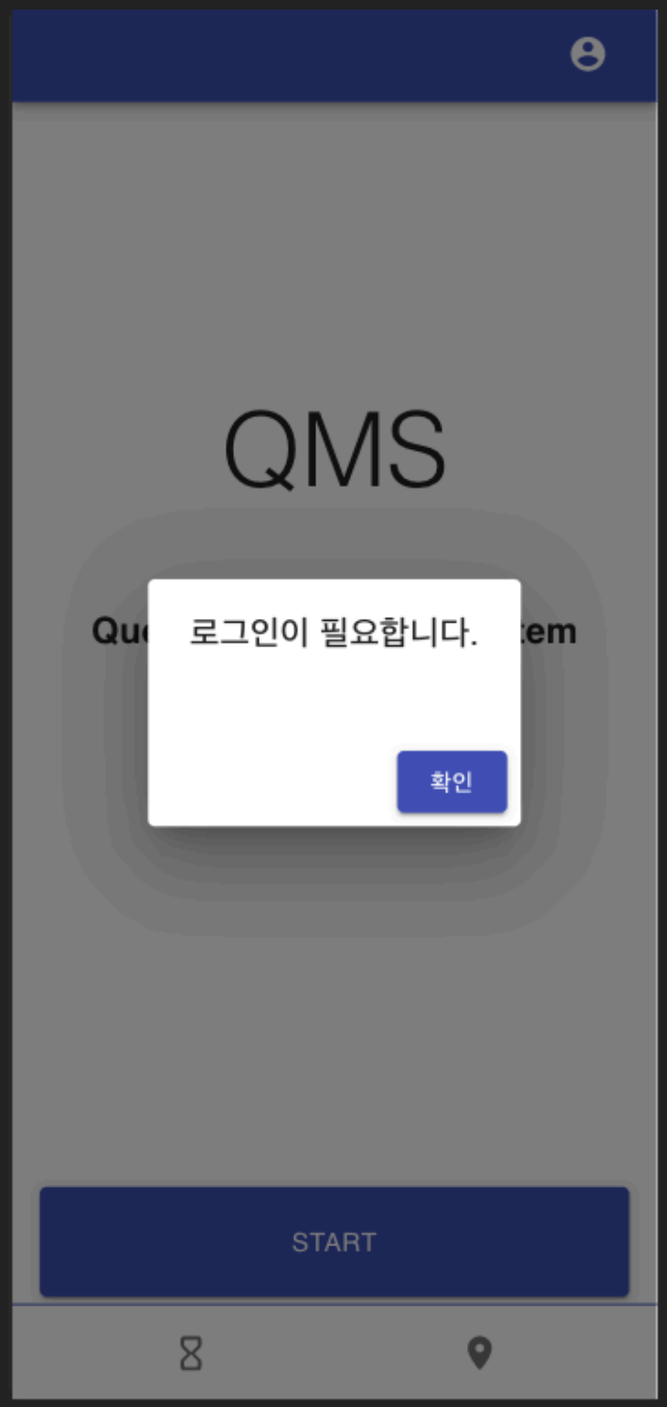
TDD란?

```
sorr ~/Study/qms-server (master*) $
sorr ~/Study/qms-server (master*) $ rspec spec
/Users/sorr/.rbenv/versions/2.6.6/lib/ruby/gems/2.6.0/gems/shoulda-matchers-2.8.0/lib/shoulda/matchers/active_model/validate_inclusion_of_mat
cher.rb:251: warning: BigDecimal.new is deprecated; use BigDecimal() method instead.
/Users/sorr/.rbenv/versions/2.6.6/lib/ruby/gems/2.6.0/gems/shoulda-matchers-2.8.0/lib/shoulda/matchers/active_model/validate_inclusion_of_mat
cher.rb:251: warning: BigDecimal.new is deprecated; use BigDecimal() method instead.
....................F.......................................................................*..........*....

Pending: (Failures listed here are expected and do not affect your suite's status)

  1) QueueContainerLog add some examples to (or delete) /Users/sorr/Study/qms-server/spec/models/queue_container_log_spec.rb
     # Not yet implemented
     # ./spec/models/queue_container_log_spec.rb:4

  2) Todo add some examples to (or delete) /Users/sorr/Study/qms-server/spec/models/todo_spec.rb
     # Not yet implemented
     # ./spec/models/todo_spec.rb:4


Failures:

  1) Admin::QueueContainersController GET #index 로그인 된 유저일 경우 이용가능한 QueueContainers 를 모두 내려준다
     Failure/Error: expect(JSON.parse(response.body).length).to eq(5)

       expected: 5
            got: 1

       (compared using ==)
     # ./spec/controllers/admin/queue_containers_controller_spec.rb:31:in `block (3 levels) in <top (required)>'

Finished in 3.73 seconds (files took 8.17 seconds to load)
107 examples, 1 failure, 2 pending

Failed examples:

rspec ./spec/controllers/admin/queue_containers_controller_spec.rb:29 # Admin::QueueContainersController GET #index 로그인 된 유저일 경우 이
용가능한 QueueContainers 를 모두 내려준다
```

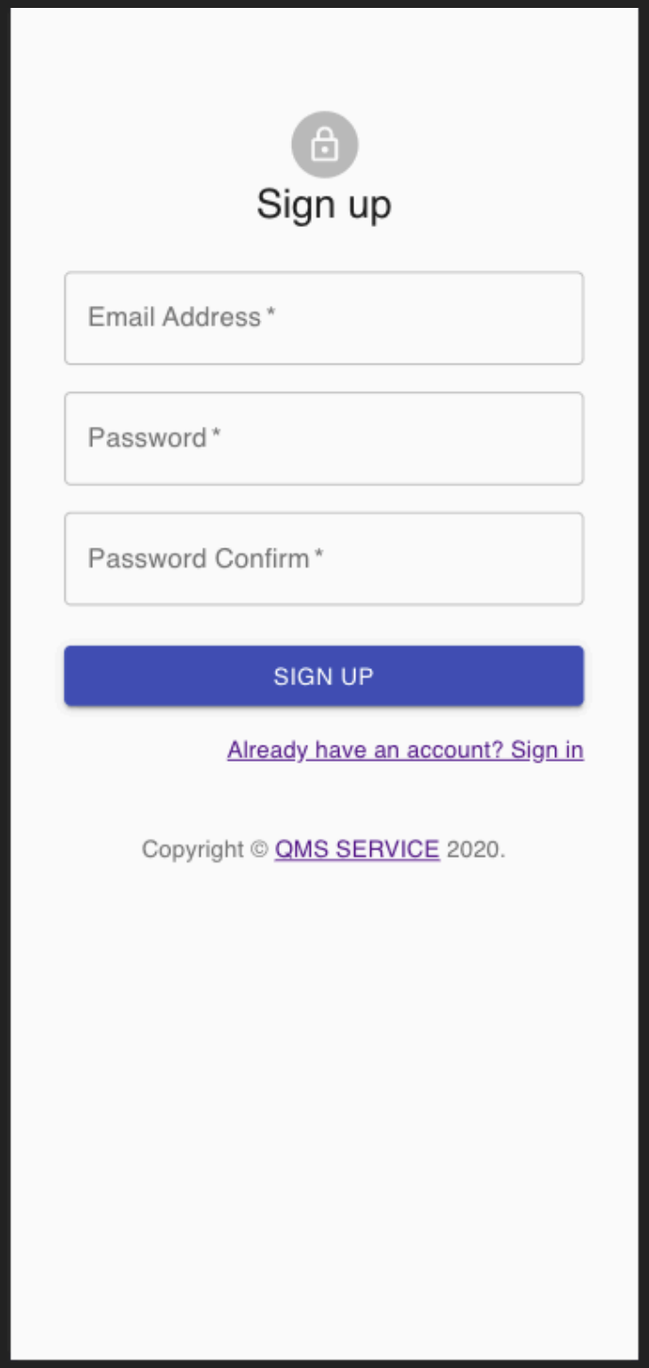
```
sorr ~/Study/qms-server (master*) $ rspec spec
/Users/sorr/.rbenv/versions/2.6.6/lib/ruby/gems/2.6.0/gems/shoulda-matchers-2.8.0/lib/shoulda/matchers/active_model/validate_inclusion_of_mat
cher.rb:251: warning: BigDecimal.new is deprecated; use BigDecimal() method instead.
.....................................................................................................

Finished in 3.51 seconds (files took 2.73 seconds to load)
105 examples, 0 failures
```

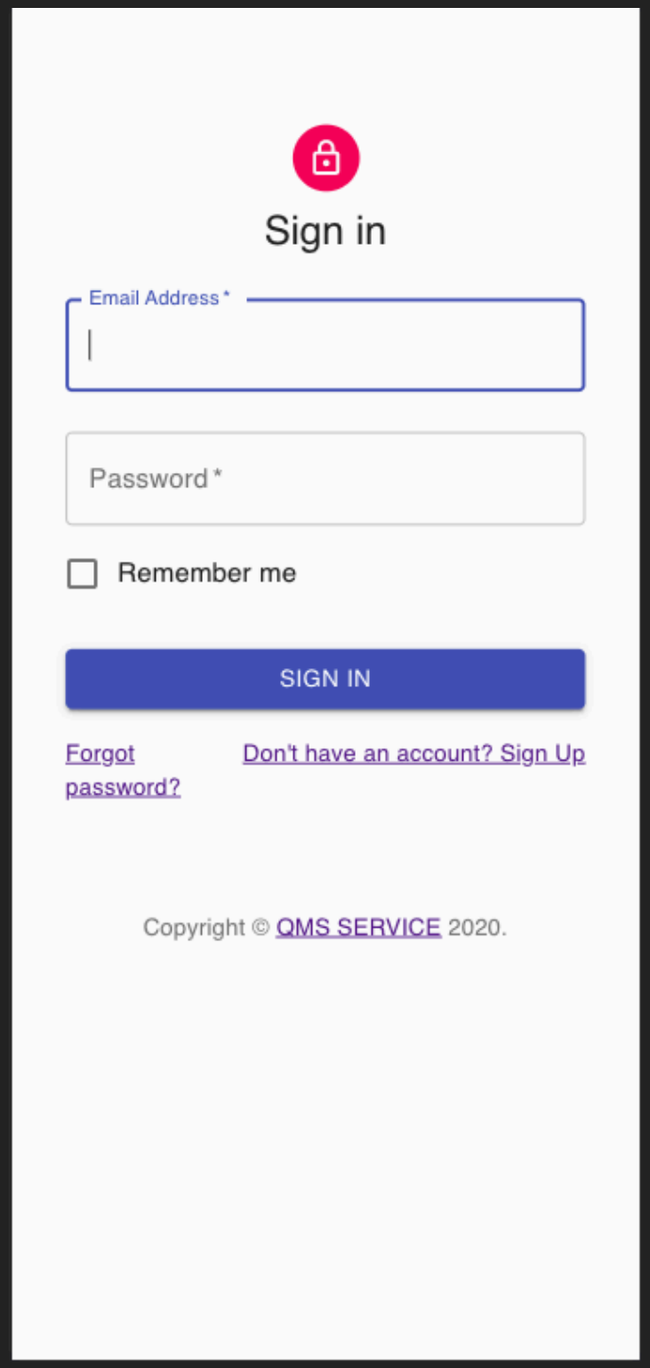
MAIN



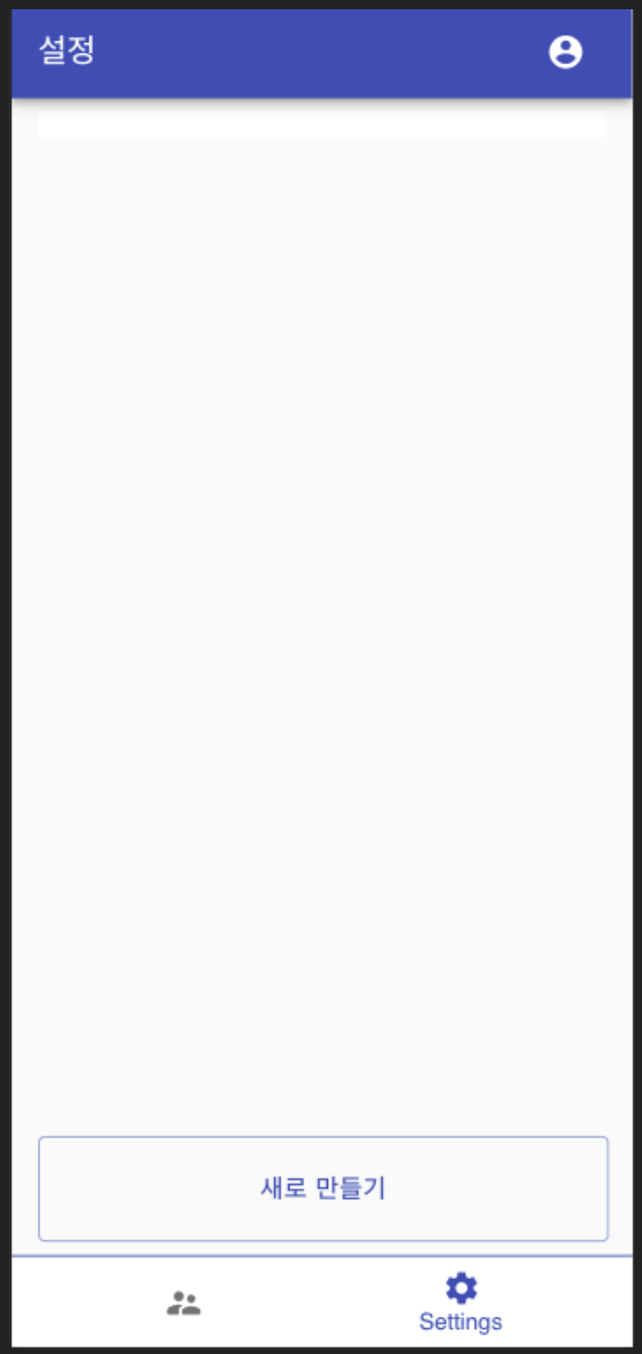


SIGN UP PAGE / SIGN IN PAGE



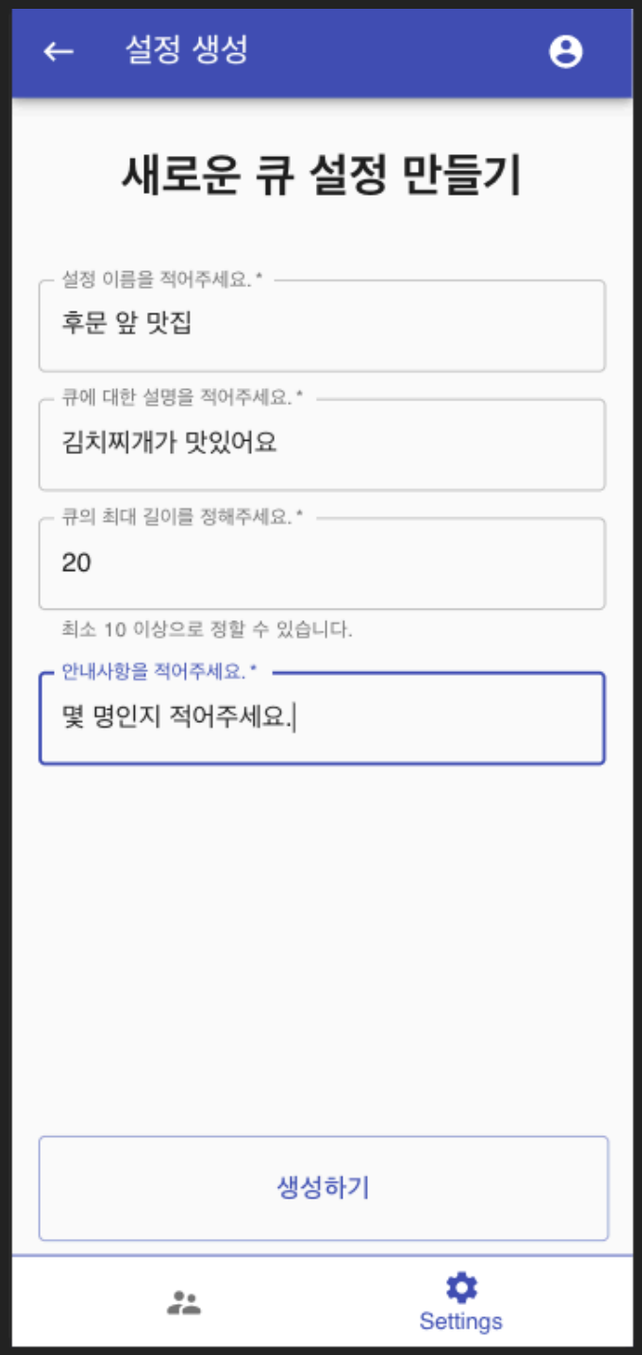


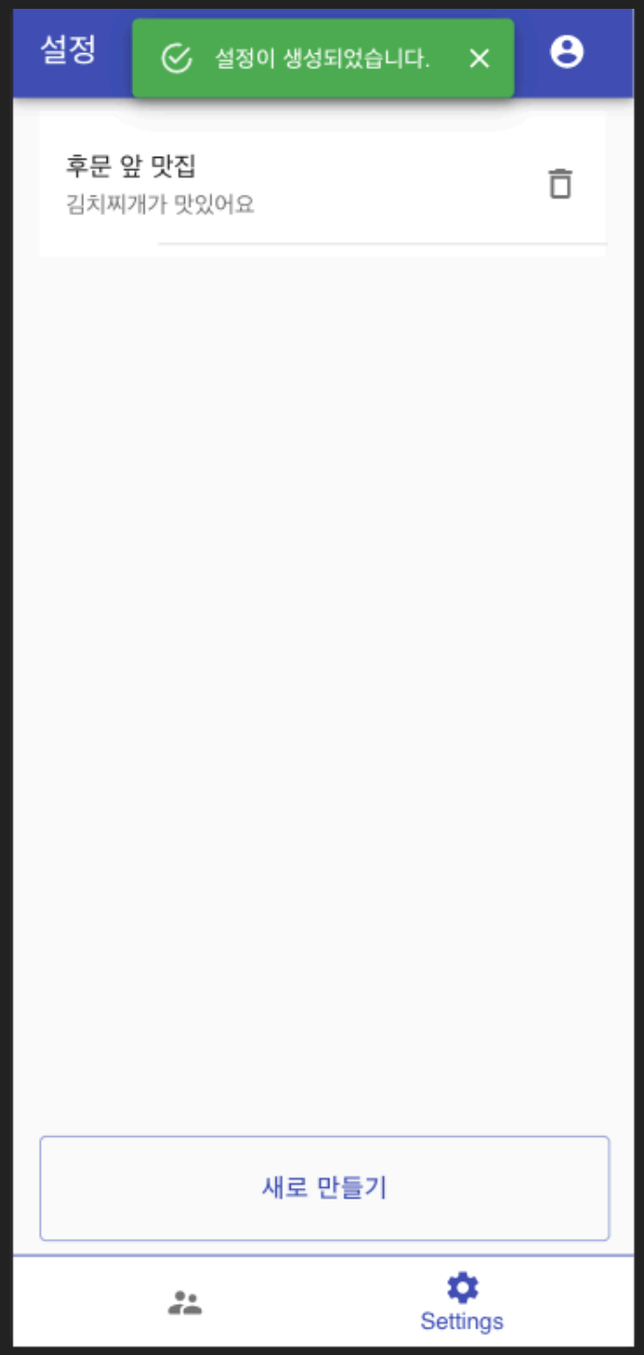
설정 관리


설정
새로 만들기
Settings


설정 생성
새로운 큐 설정 만들기
설정 이름을 적어주세요. *
큐에 대한 설명을 적어주세요. *
큐의 최대 길이를 정해주세요. *
최소 10 이상으로 정할 수 있습니다.
안내사항을 적어주세요. *
생성하기
Settings

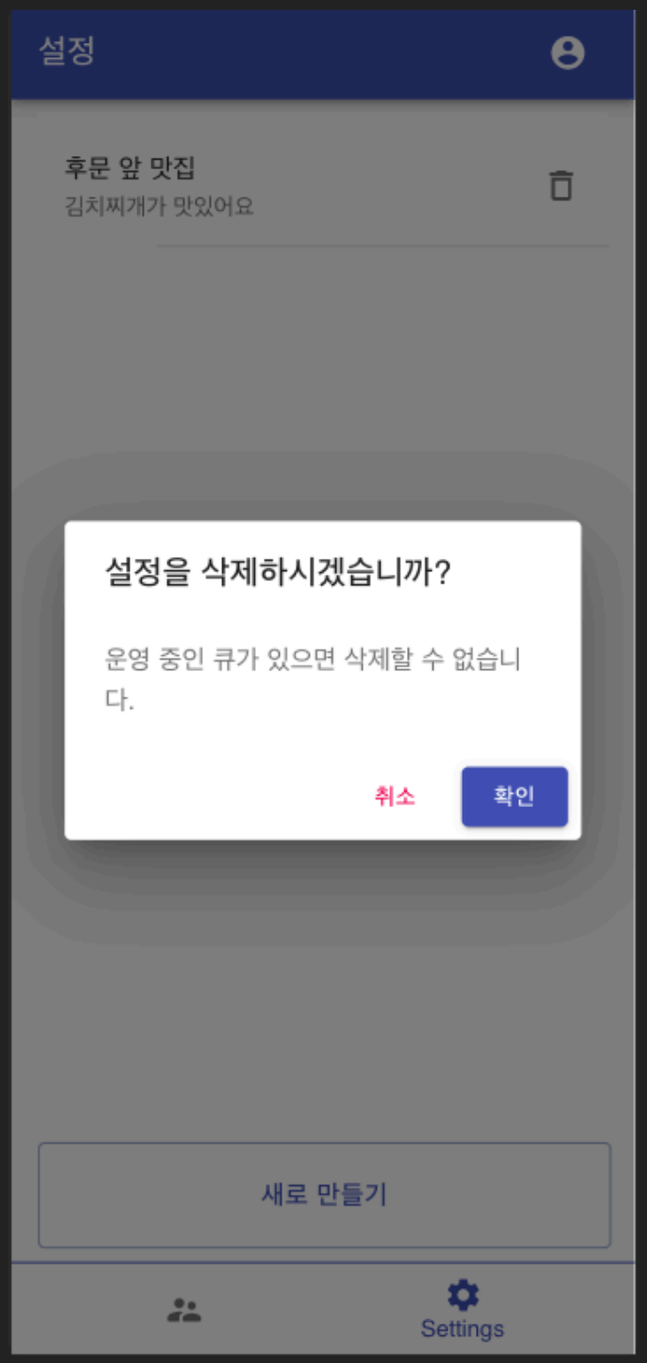
설정 관리

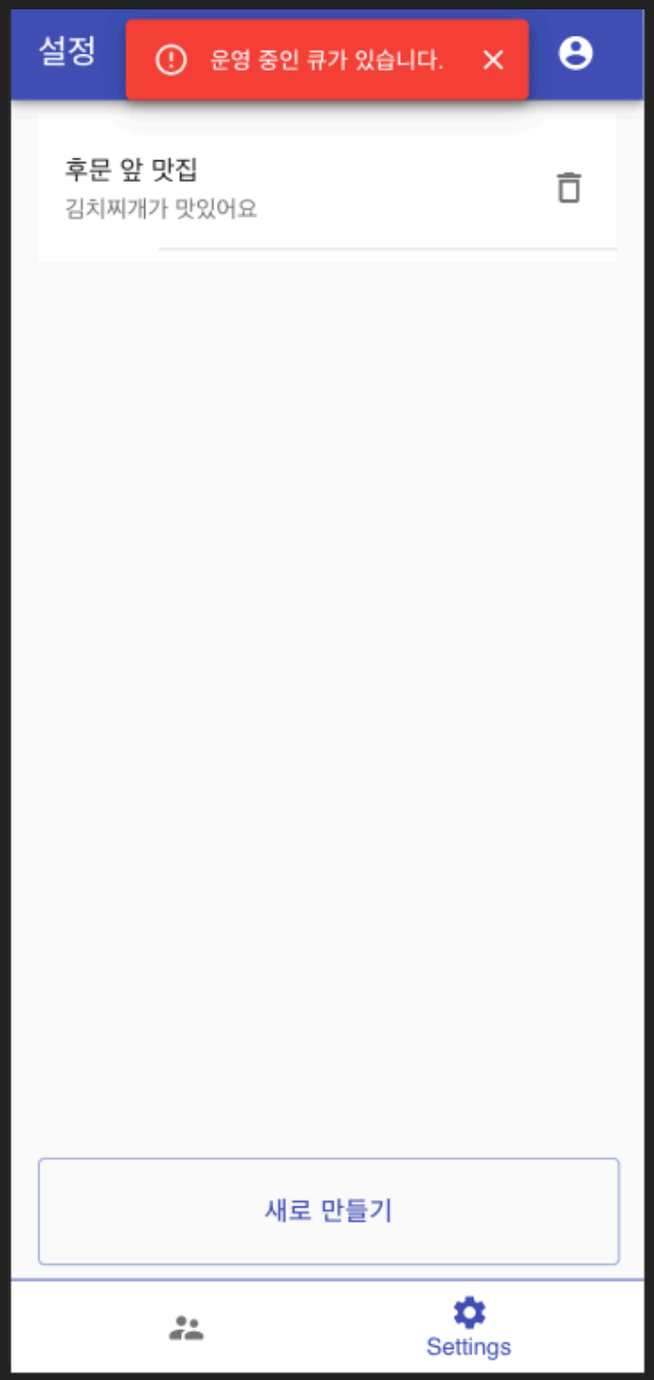

설정 생성
새로운 큐 설정 만들기
설정 이름을 적어주세요. *
후문 앞 맛집
큐에 대한 설명을 적어주세요. *
김치찌개가 맛있어요
큐의 최대 길이를 정해주세요. *
20
최소 10 이상으로 정할 수 있습니다.
안내사항을 적어주세요. *
몇 명인지 적어주세요.
생성하기
Settings


설정
설정이 생성되었습니다.
후문 앞 맛집
김치찌개가 맛있어요
새로 만들기
Settings

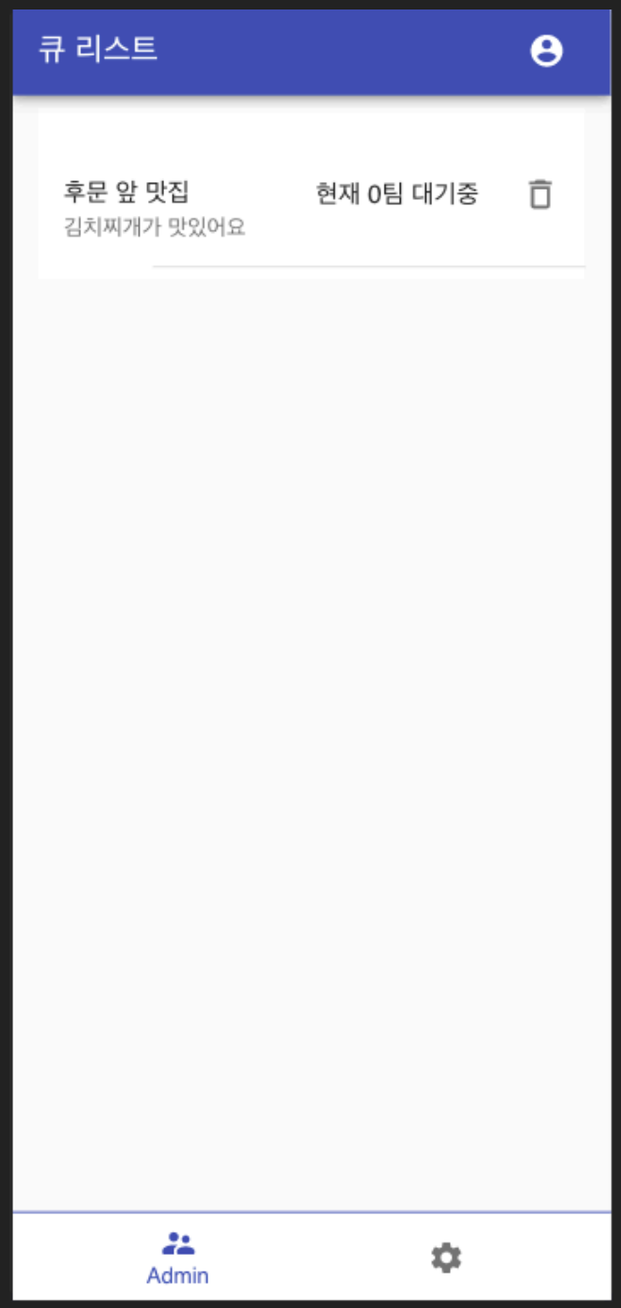

설정 편집
큐 설정
설정 이름을 적어주세요. *
후문 앞 맛집
큐에 대한 설명을 적어주세요. *
김치찌개가 맛있어요
큐의 최대 길이를 정해주세요. *
20
최소 10 이상으로 정할 수 있습니다.
안내 사항을 적어주세요. *
몇 명인지 적어주세요.
큐 시작하기
저장하기
Settings

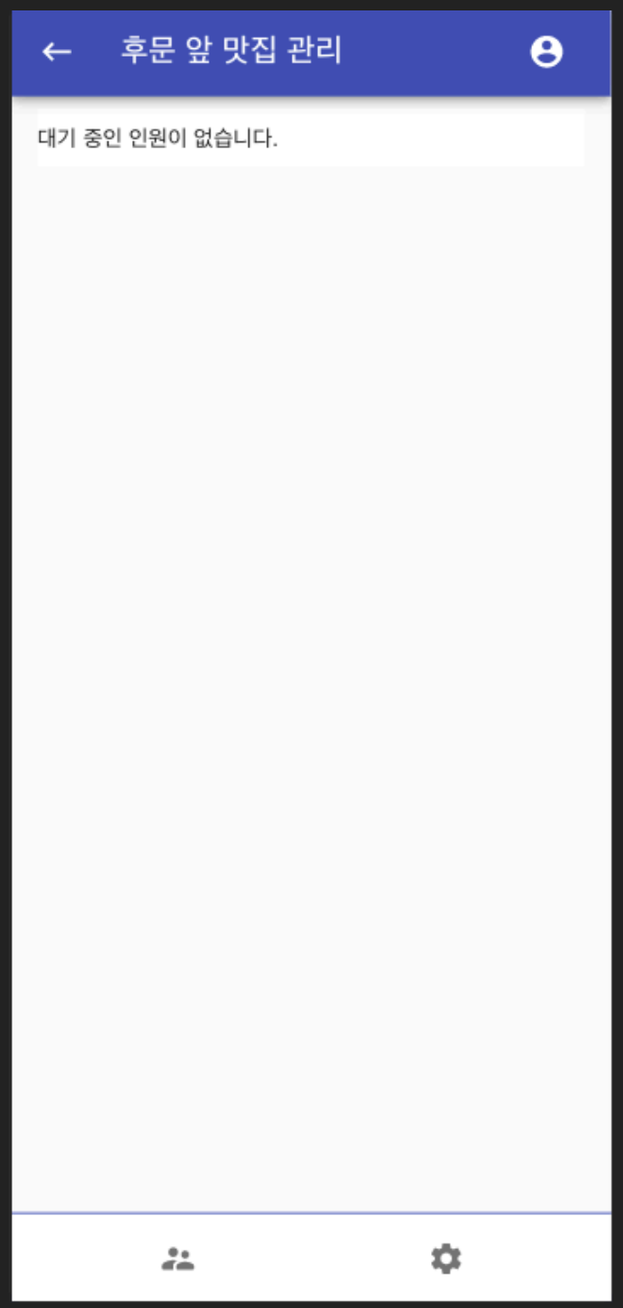
설정 관리

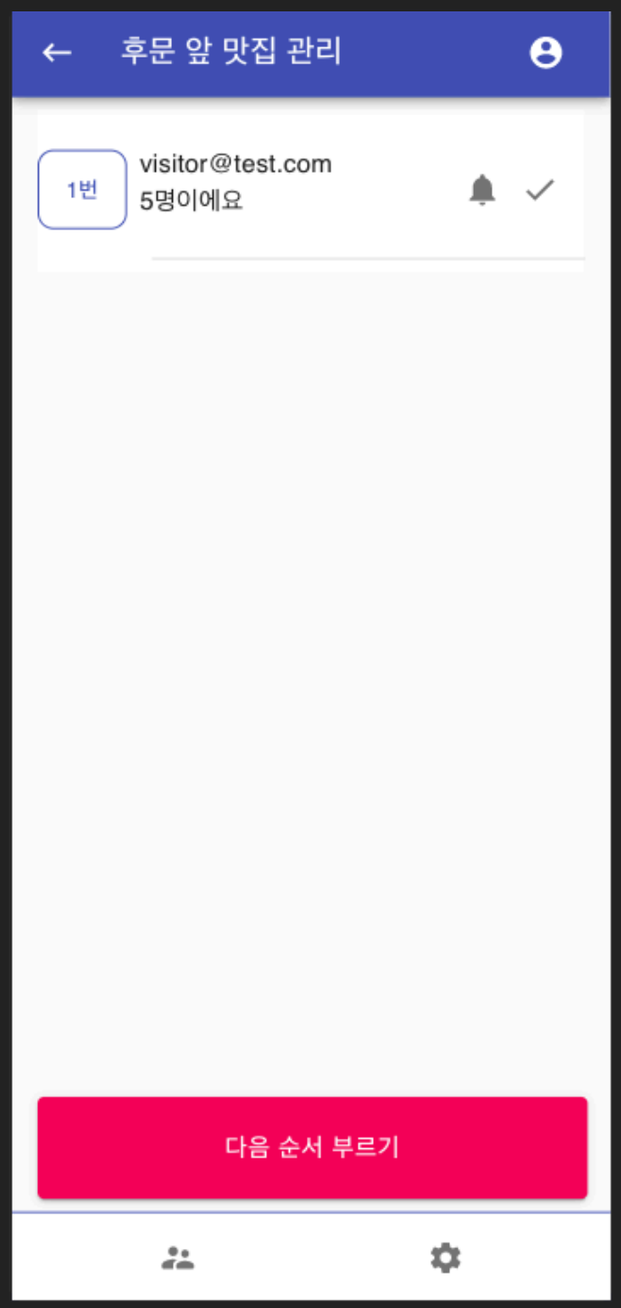

설정
후문 앞 맛집
김치찌개가 맛있어요
설정을 삭제하시겠습니까?
운영 중인 큐가 있으면 삭제할 수 없습니다.
취소
확인
새로 만들기
Settings


설정
운영 중인 큐가 있습니다.
후문 앞 맛집
김치찌개가 맛있어요
새로 만들기
Settings

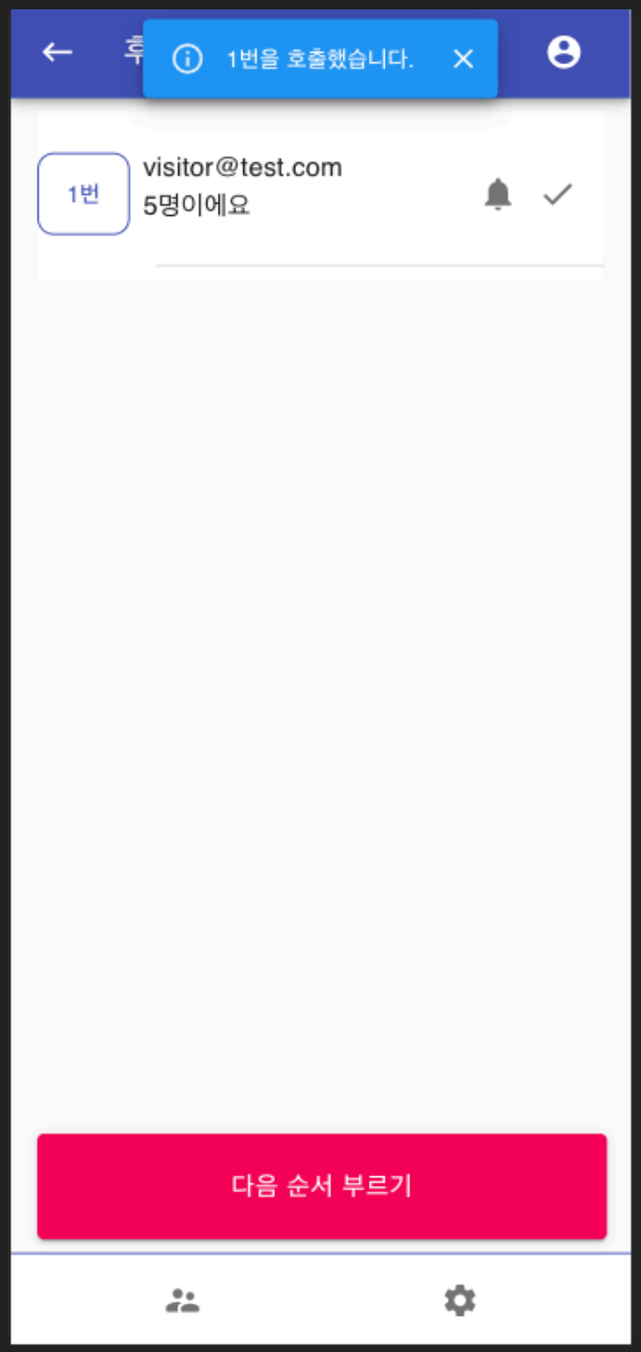
대기 관리 시작 / 대기 관리 화면

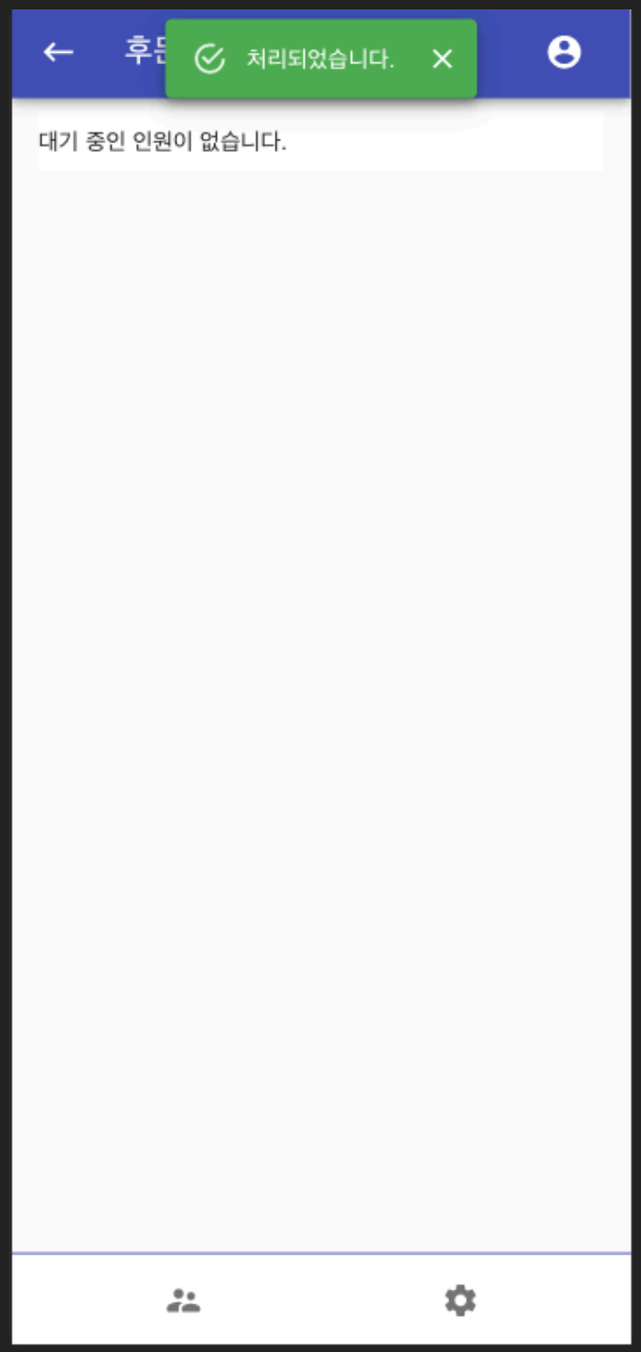




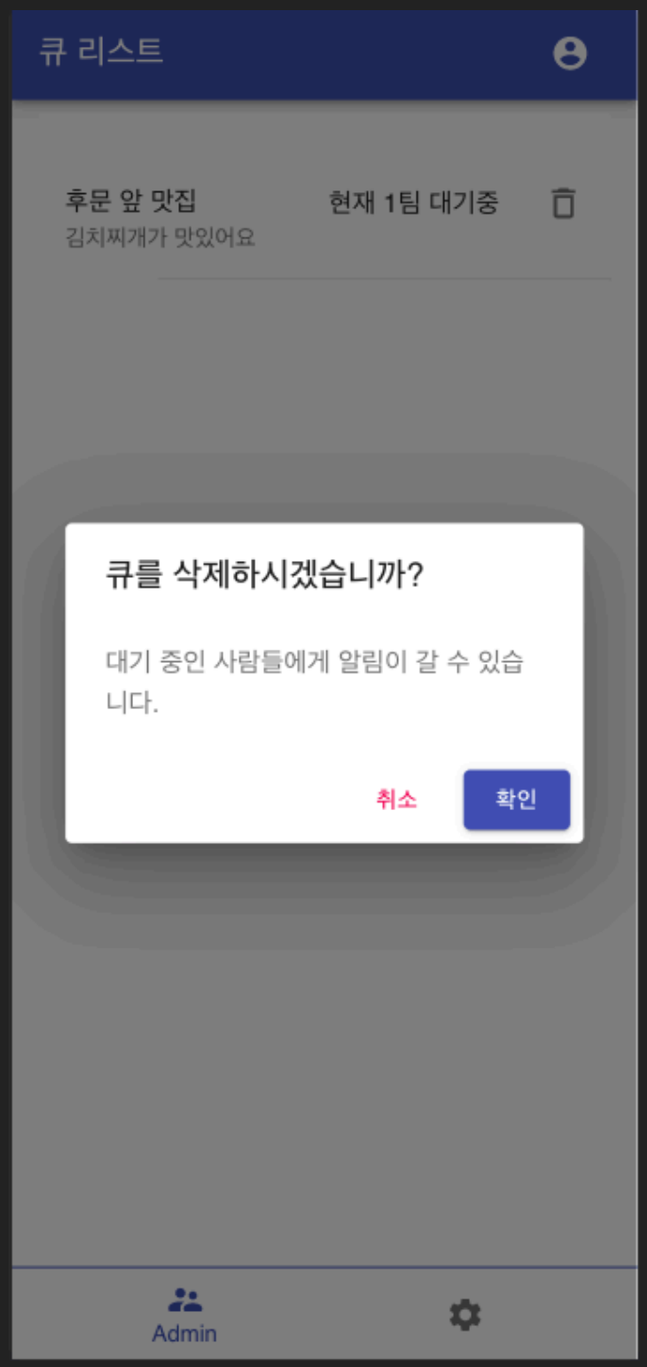


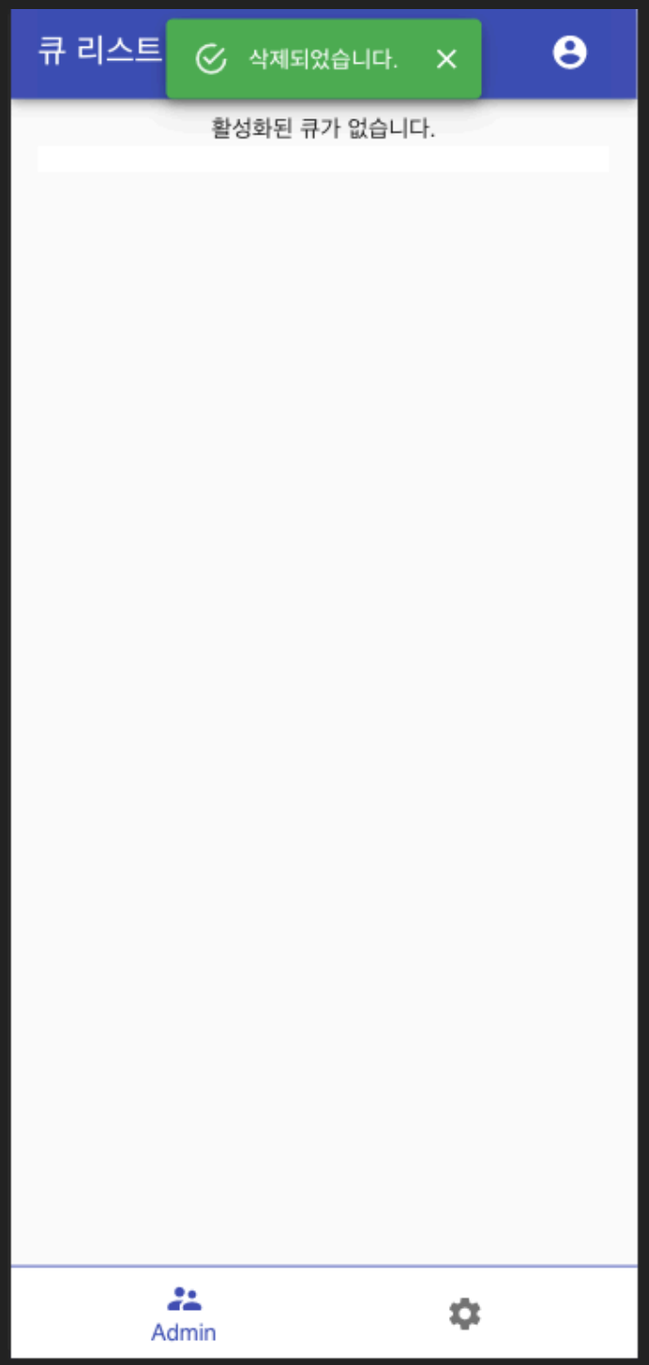
대기 호출 / 입장 처리

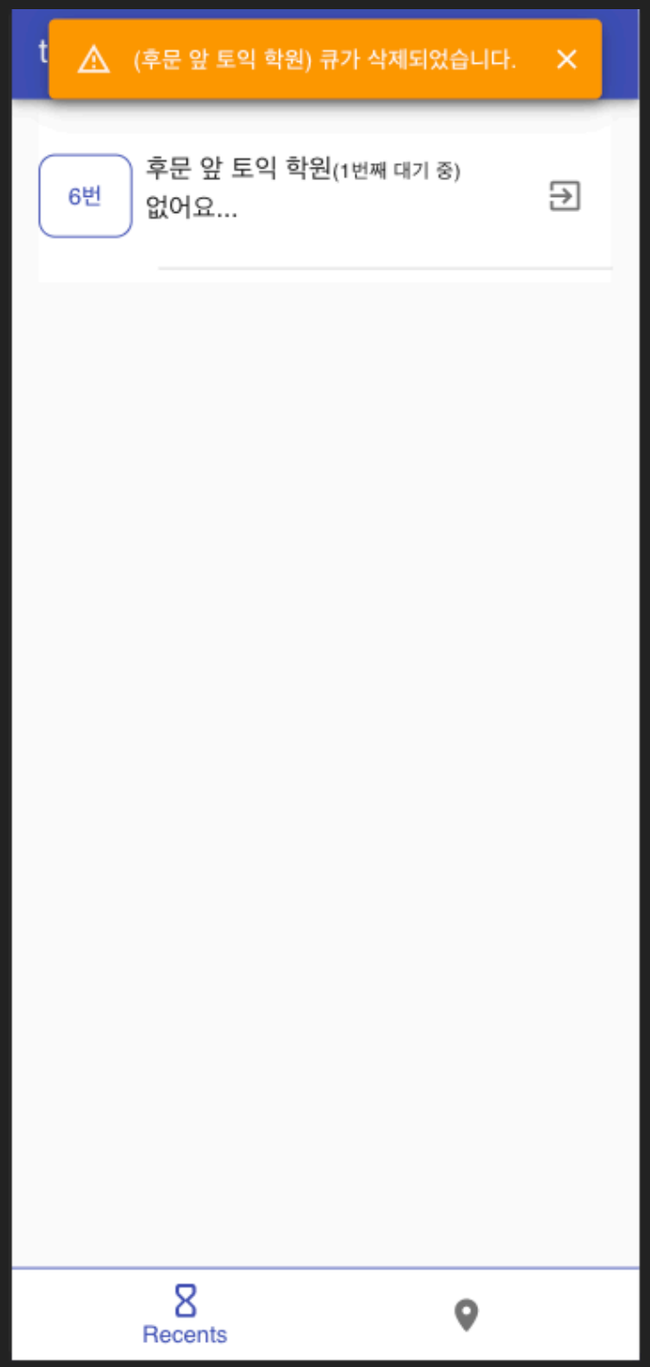

1번을 호출했습니다.
1번
visitor@test.com
5명이에요
다음 순서 부르기


처리되었습니다.
대기 중인 인원이 없습니다.

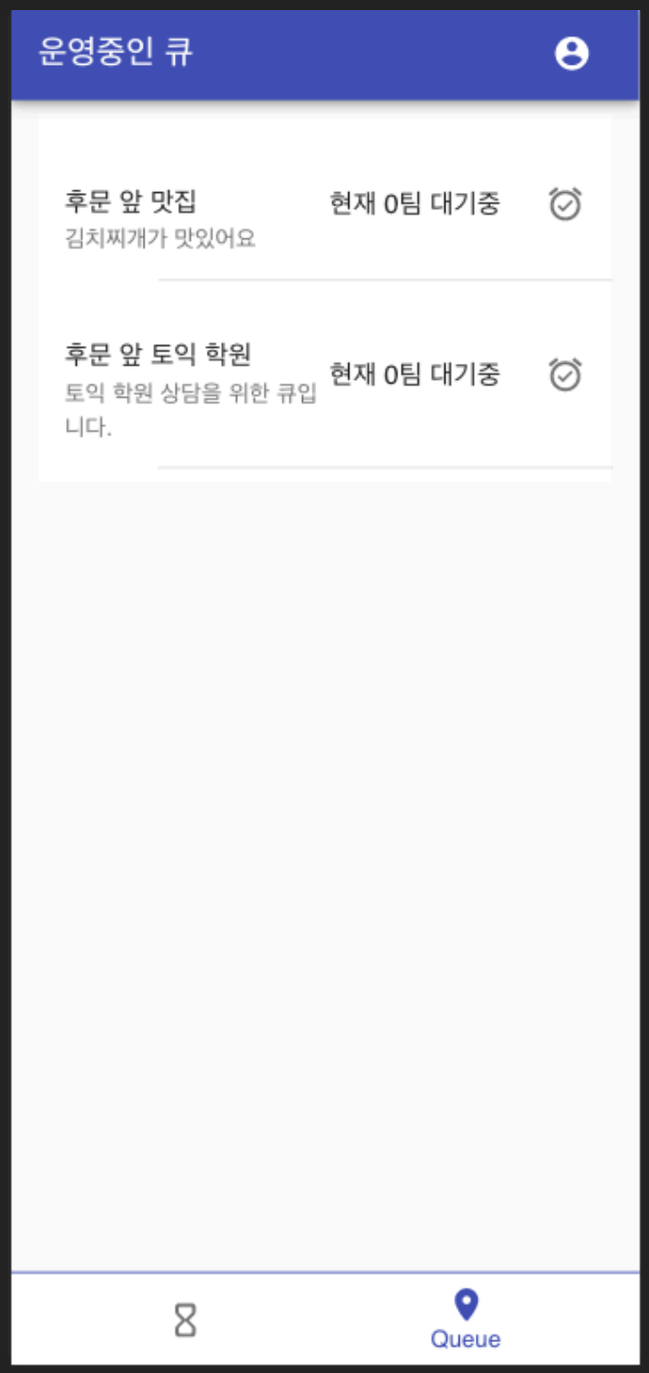
큐 관리

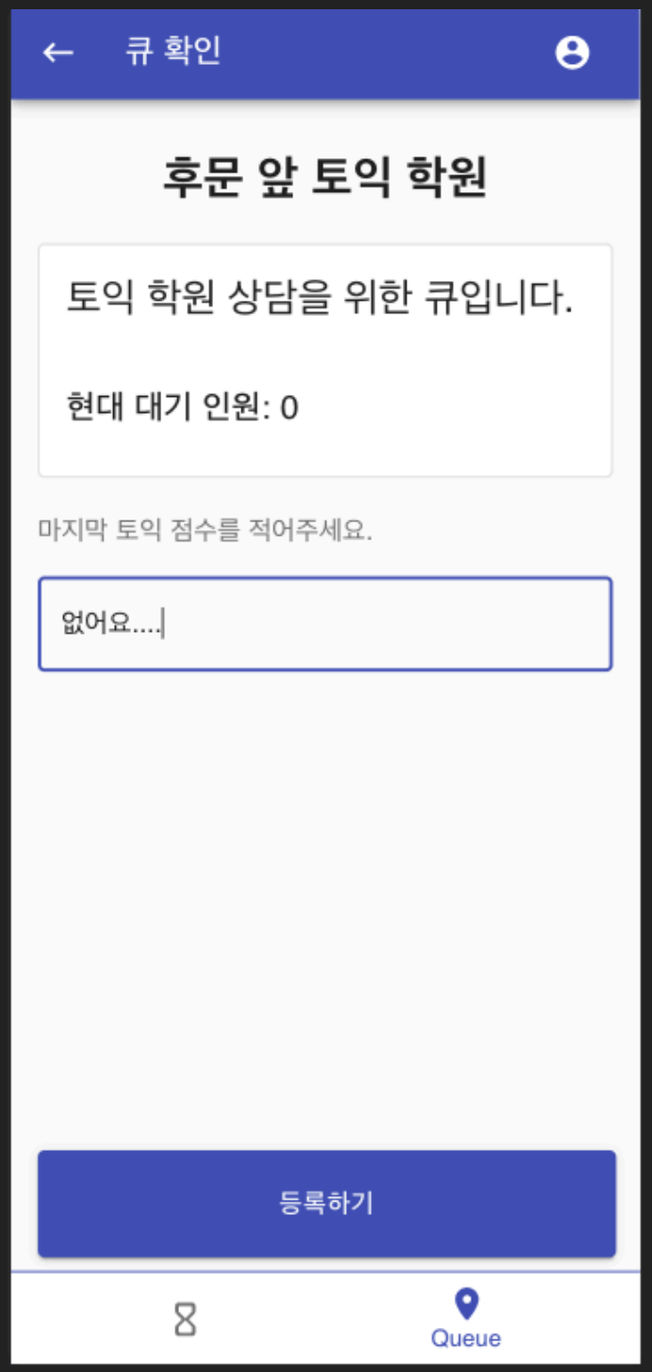

큐 리스트
후문 앞 맛집
김치찌개가 맛있어요
현재 1팀 대기중
큐를 삭제하시겠습니까?
대기 중인 사람들에게 알림이 갈 수 있습니다.
취소
확인
Admin

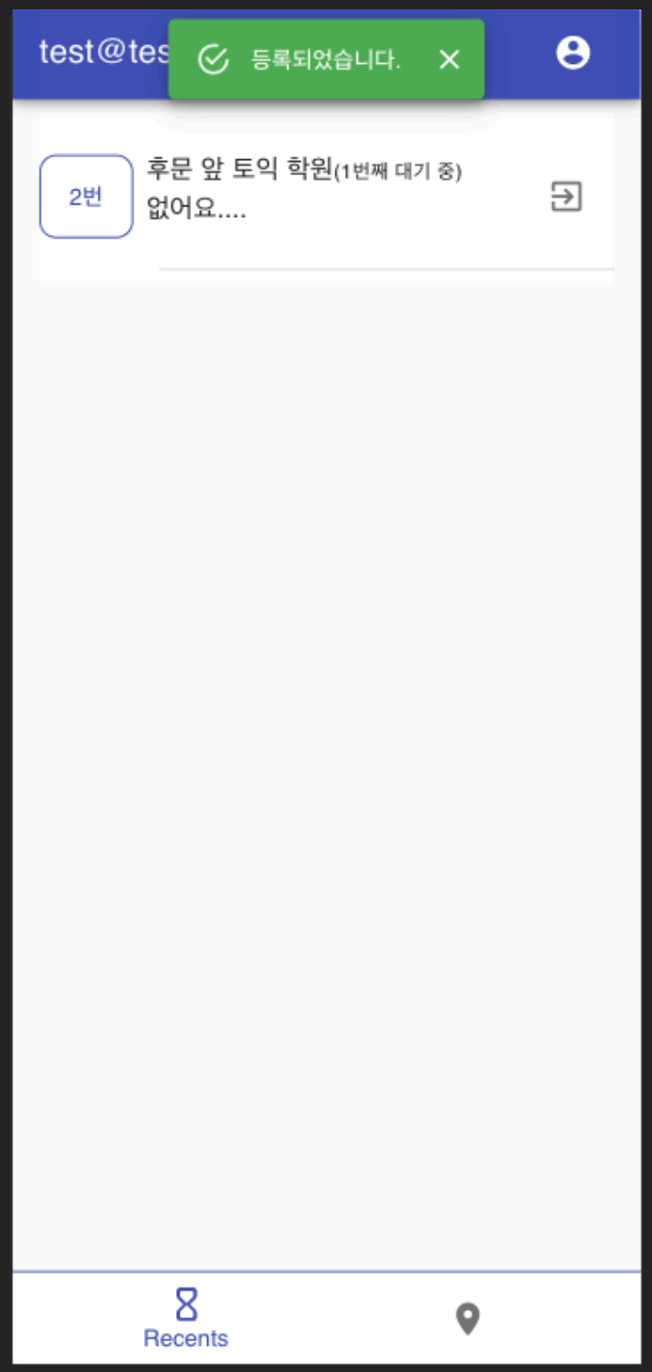

큐 리스트
삭제되었습니다.
활성화된 큐가 없습니다.
Admin


(후문 앞 토익 학원) 큐가 삭제되었습니다.
6번
후문 앞 토익 학원(1번째 대기 중)
없어요...
Recents

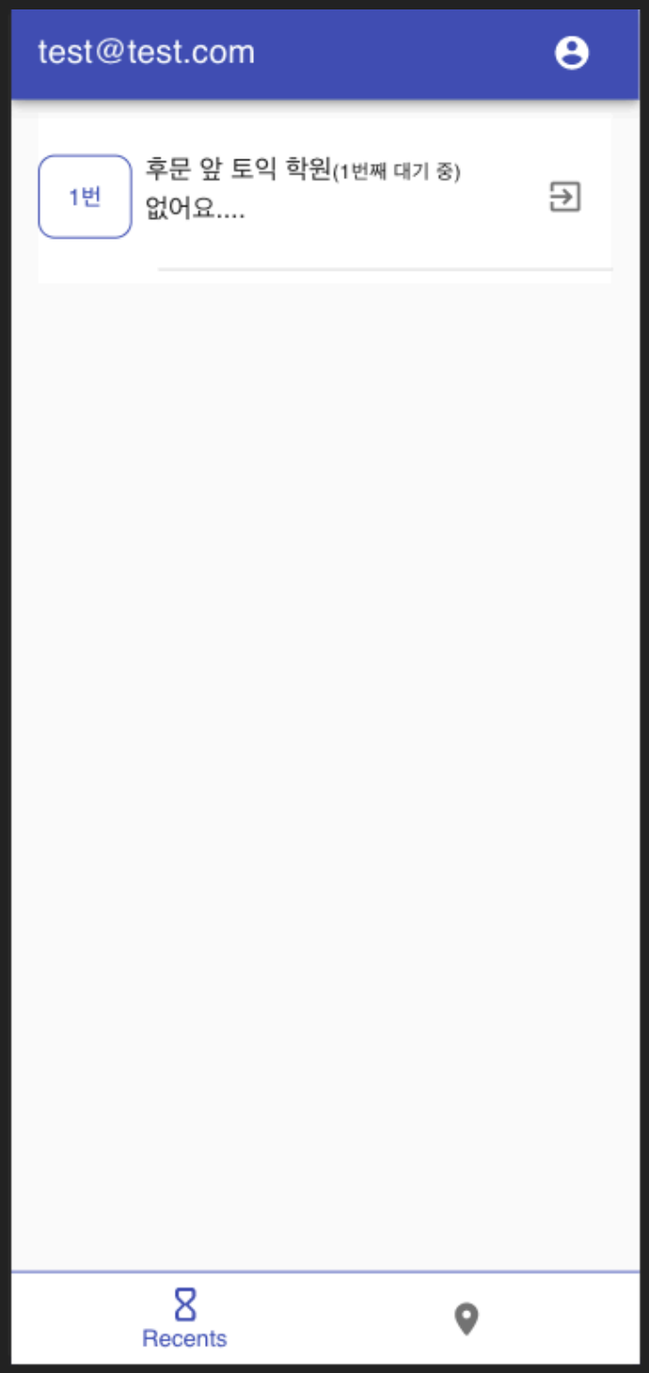
방문자 화면

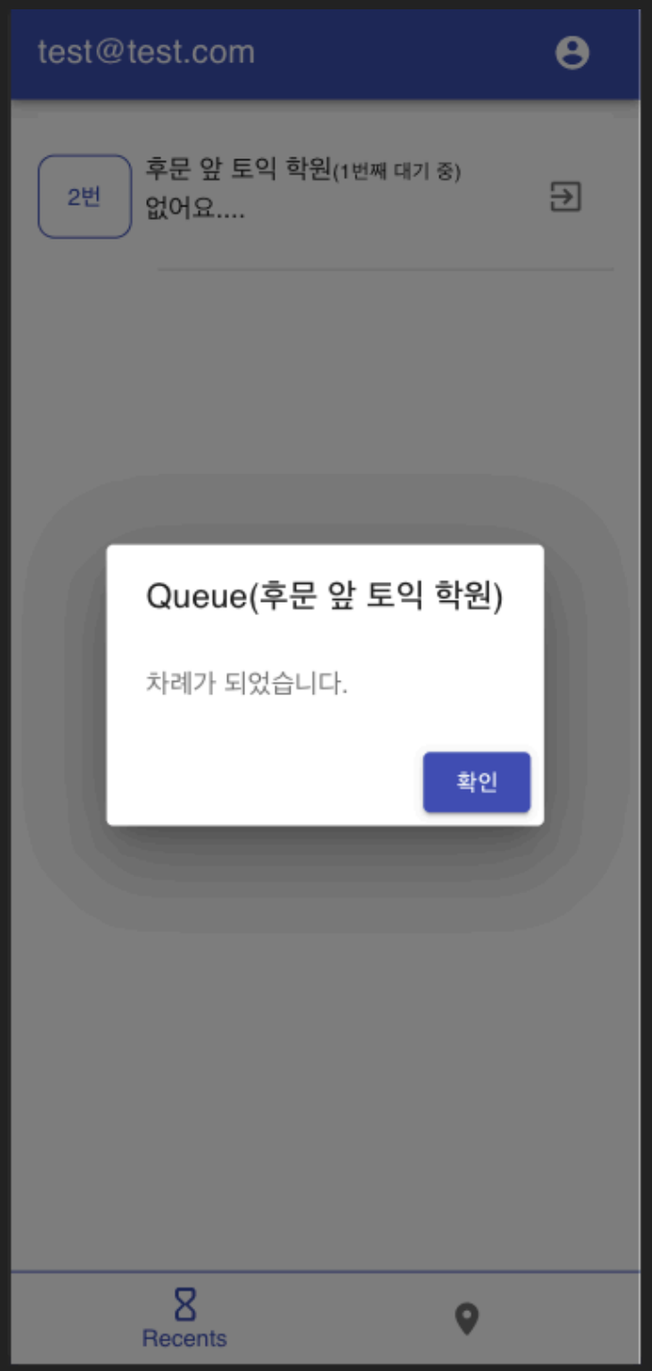

운영중인 큐
후문 앞 맛집
김치찌개가 맛있어요
현재 0팀 대기중
후문 앞 토익 학원
토익 학원 상담을 위한 큐입니다.
현재 0팀 대기중
Queue

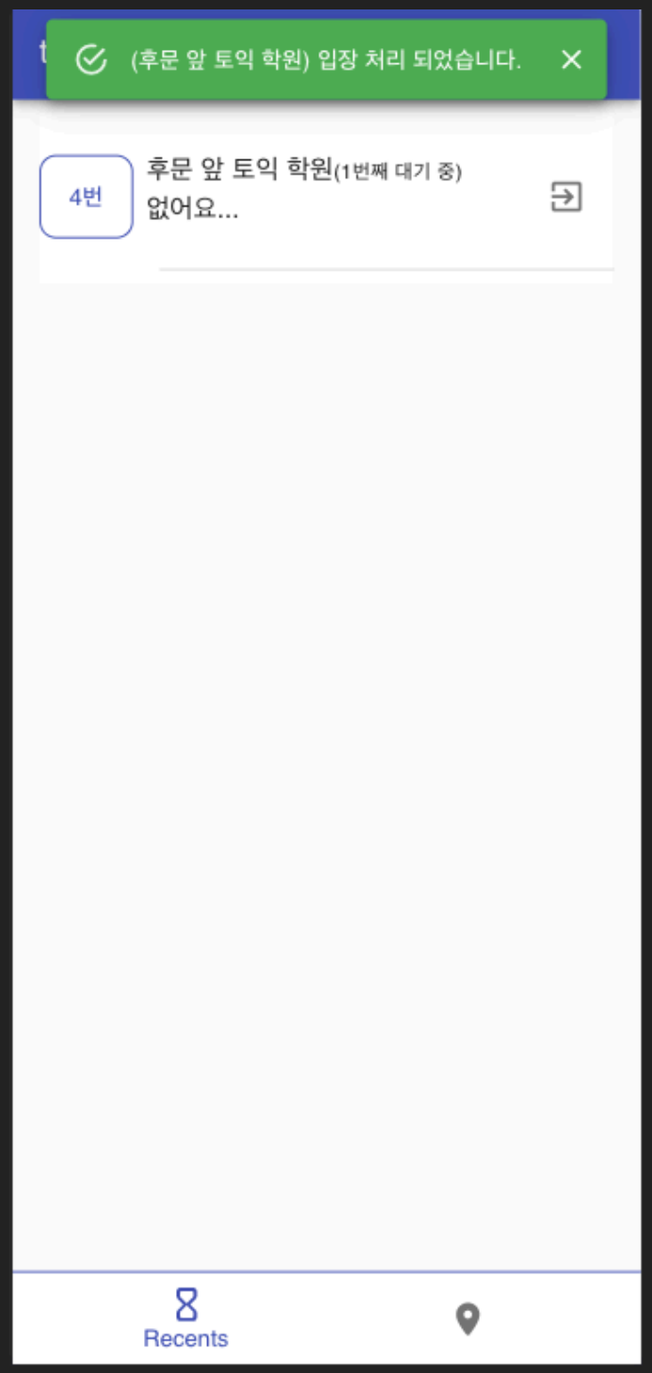

큐 확인
후문 앞 토익 학원
토익 학원 상담을 위한 큐입니다.
현대 대기 인원: 0
마지막 토익 점수를 적어주세요.
없어요....
등록하기
Queue


test@tes
등록되었습니다.
2번
후문 앞 토익 학원(1번째 대기 중)
없어요....
Recents

방문자 화면

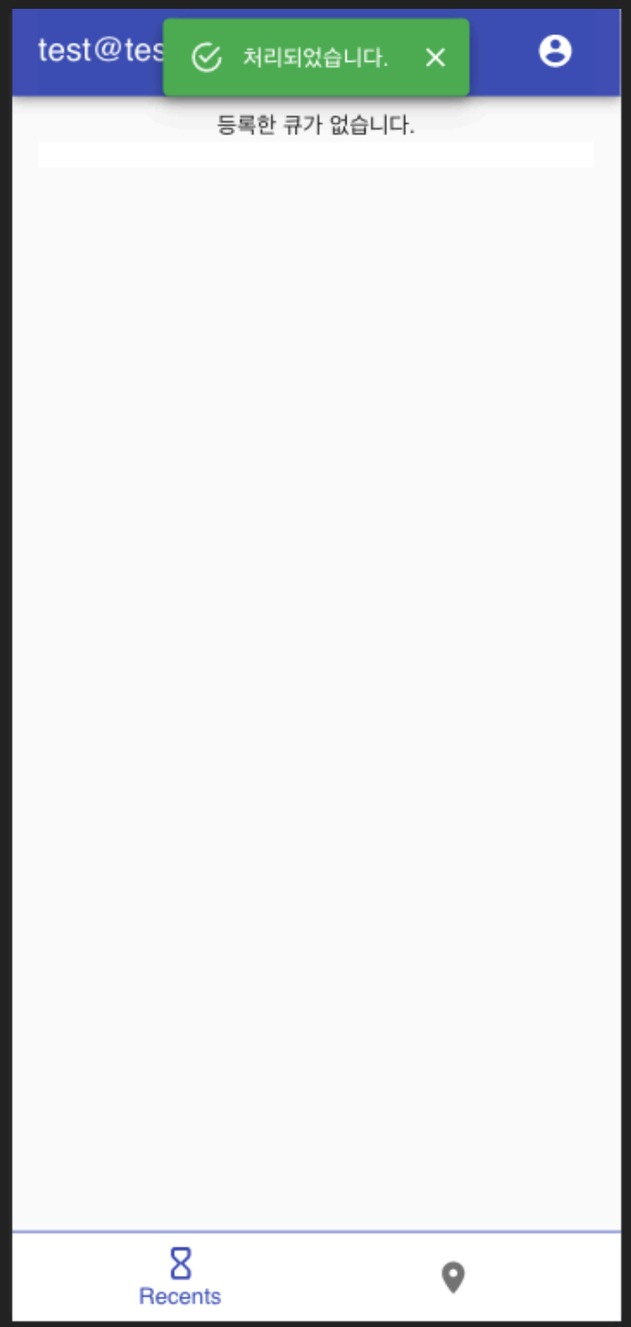

test@test.com
1번
후문 앞 토익 학원(1번째 대기 중)
없어요....
Recents


test@test.com
2번
후문 앞 토익 학원(1번째 대기 중)
없어요....
Queue(후문 앞 토익 학원)
차례가 되었습니다.
확인
Recents


(후문 앞 토익 학원) 입장 처리 되었습니다.
4번
후문 앞 토익 학원(1번째 대기 중)
없어요...
Recents

방문자 화면


test@test.com
1번
후문 앞 토익 학원(1번째 대기 중)
없어요...
대기를 중단하시겠습니까?
취소
확인
Recents


test@tes
처리되었습니다.
등록한 큐가 없습니다.
Recents

Q&A

React : https://github.com/tranquilthink/qms-front

Rails : https://github.com/tranquilthink/qms-server

감사합니다.