README

Versions used

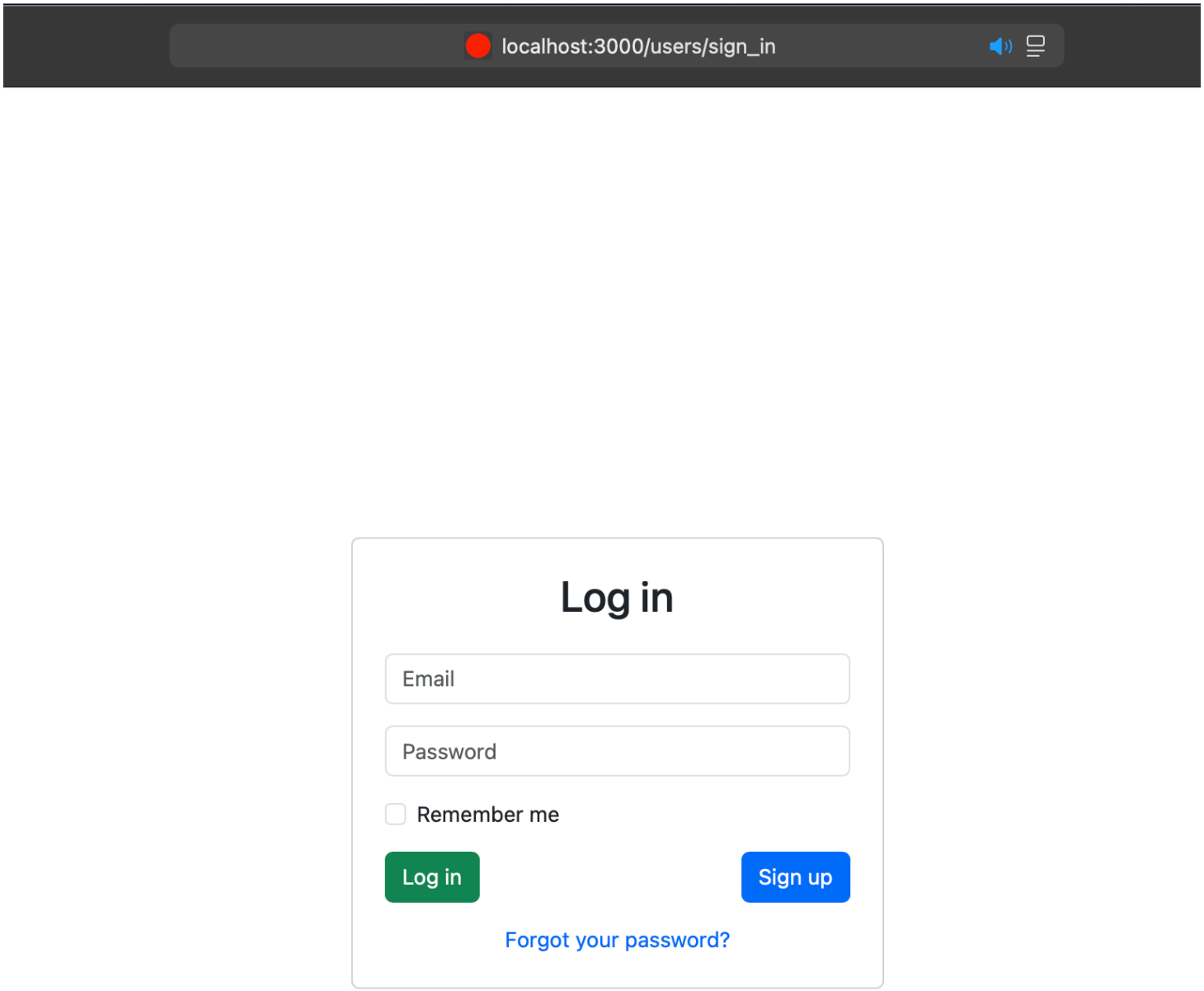
- Ruby -> 3.4.1 (2024-12-25 revision 48d4efcb85) +PRISM [arm64-darwin24]
- Rails -> 8.0.1
- Gem -> 3.6.2
- Bundler -> version 2.6.2

Table of contents

Application documentation

Login page

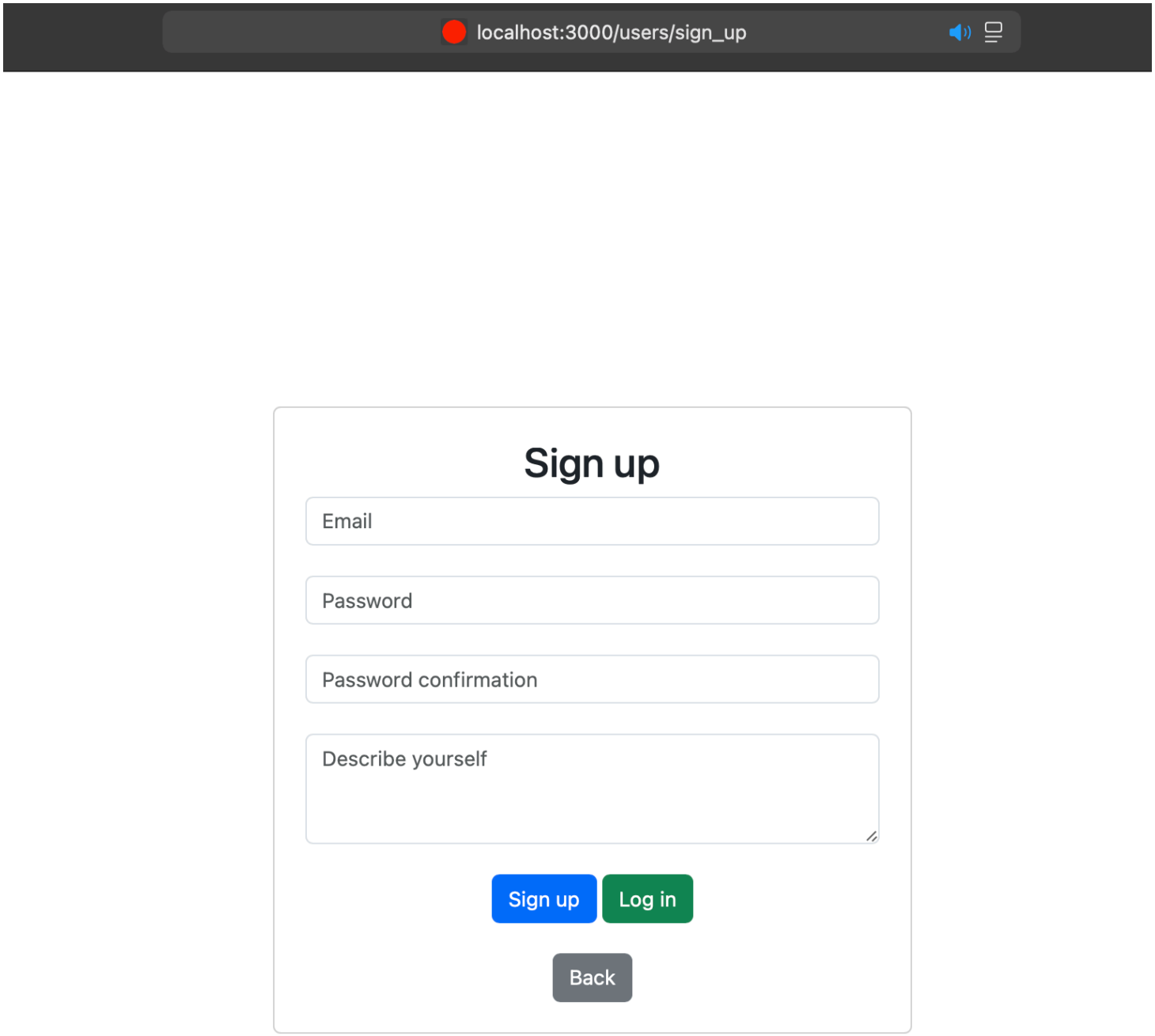
As seen in the screenshot bellow, the login page allows the user to connect to our application. Each user can either create a new account - by clicking the sign-up button, or reset his/her password. The password reset is not implemented because we chose that to implement a mailer with an actual mail is out of scope.



Login page

Sign-up page

To create a new user in the sign-up page, you must provide your email and password. Optionally, you can provide a short description to let others know what you are up to, but you can leave the field empty if you don't want to!



Sign-up page

The main page

When you login to our application, you first meet the main page - aka the INDEX page. The index page has a short overview of the functions our application offers. These include

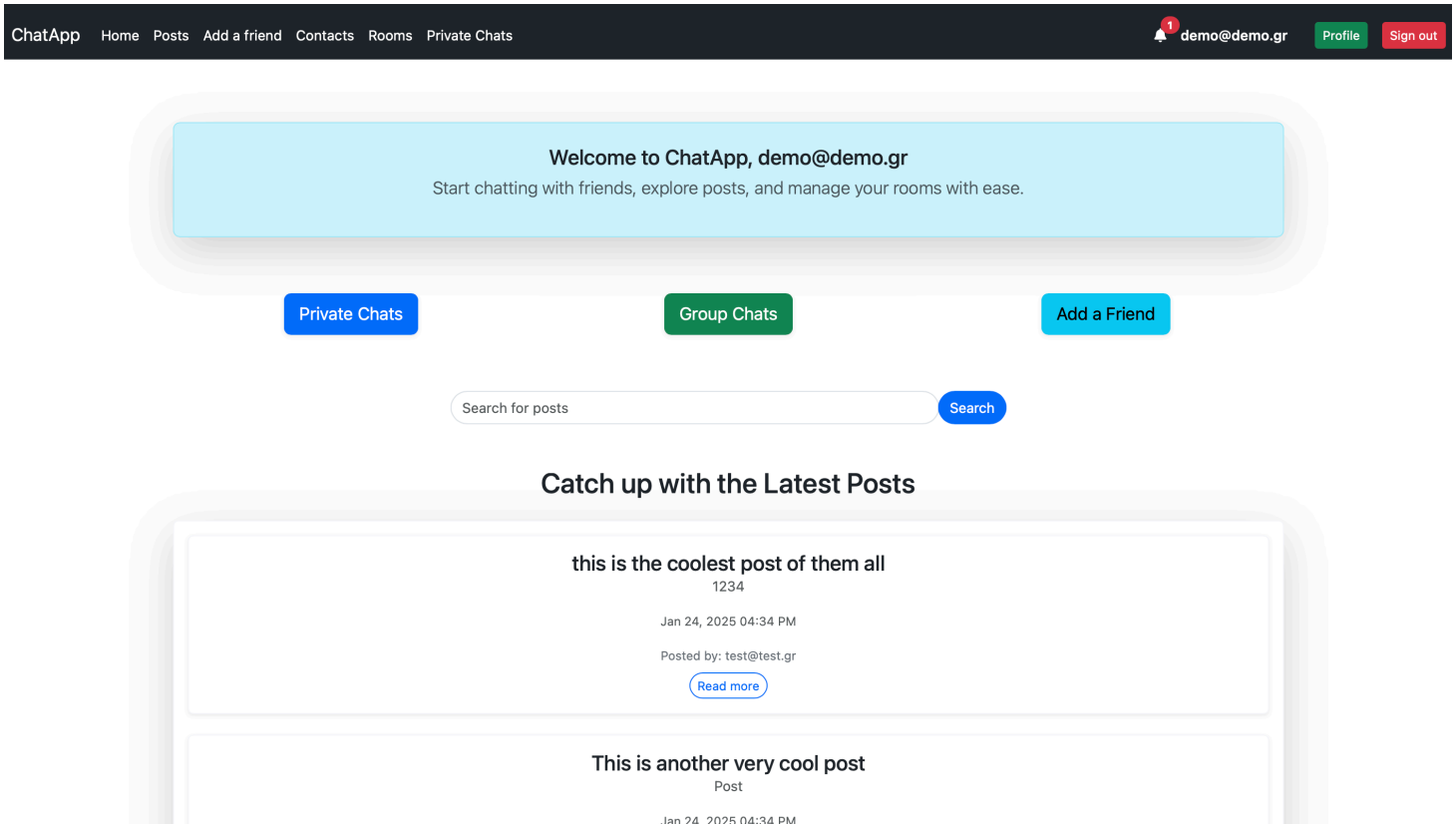
- Posting
- Friends system
- Contacts

- Notifications
- Profiles
- Public or private rooms
    - Public rooms are available for everyone to join and can be created by anyone
    - Private rooms are available only to those who are participants, aka private messages

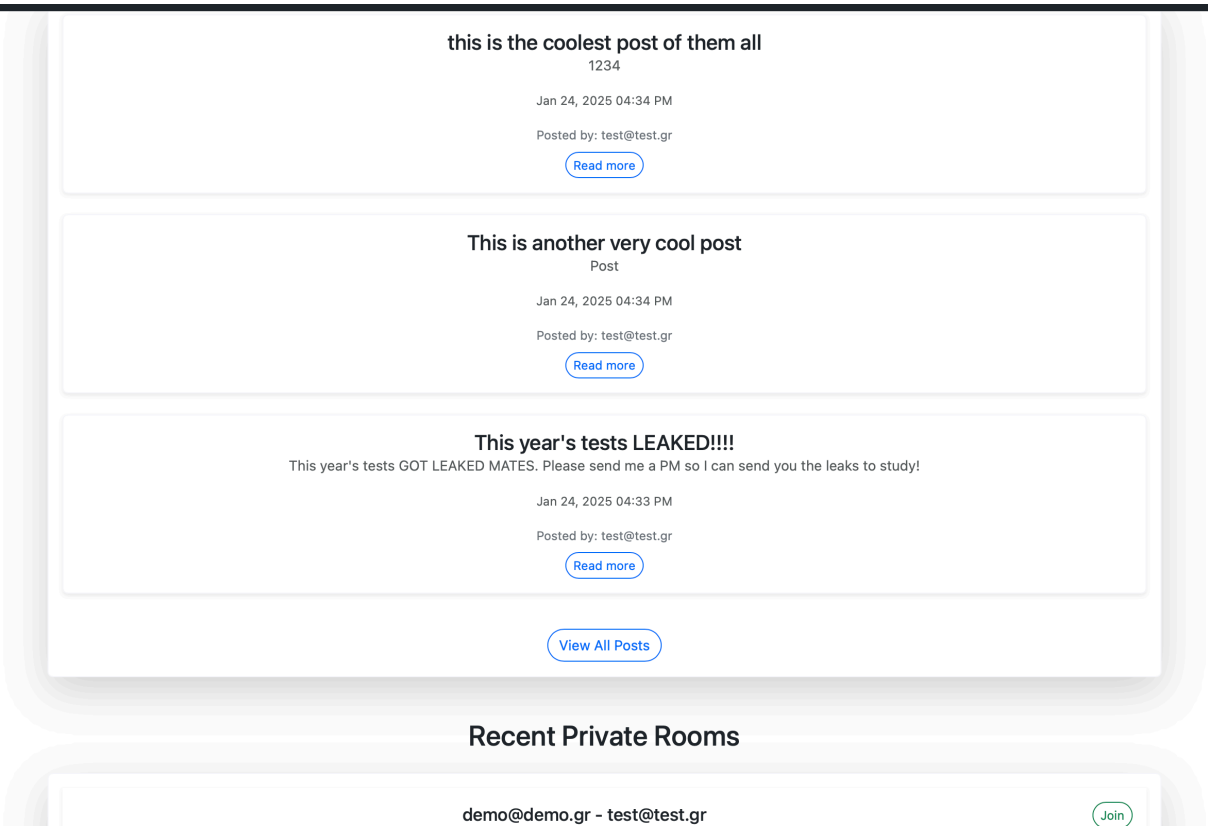
When it comes to catching up, the login page is a great place! You can access every single function immediately!

- You can access your private chats, group chats and even add a friend!
- You can search for a post or catch up with the latest 3!
- You can see your three (3) latest private chats

And more!



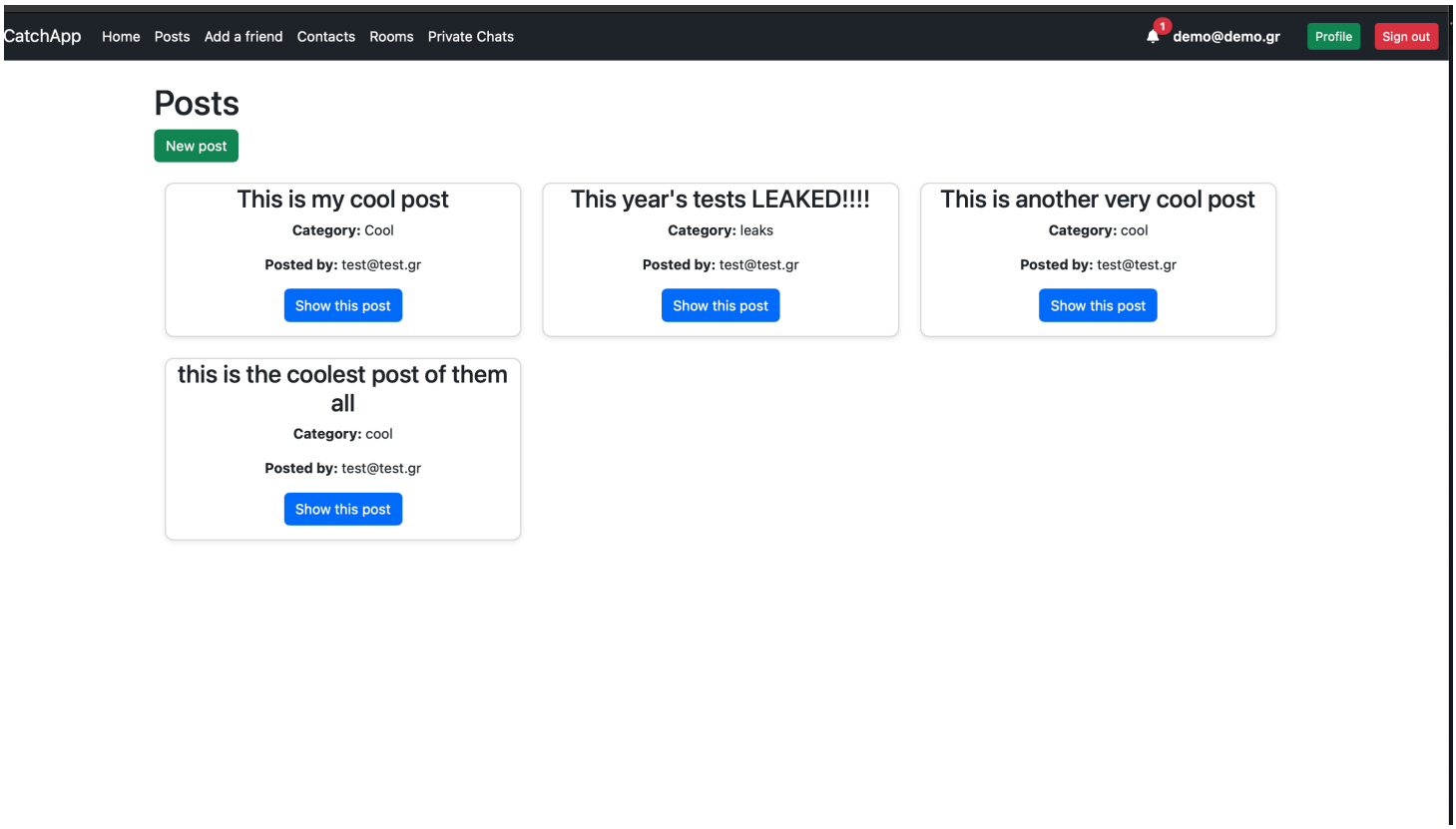
Main page



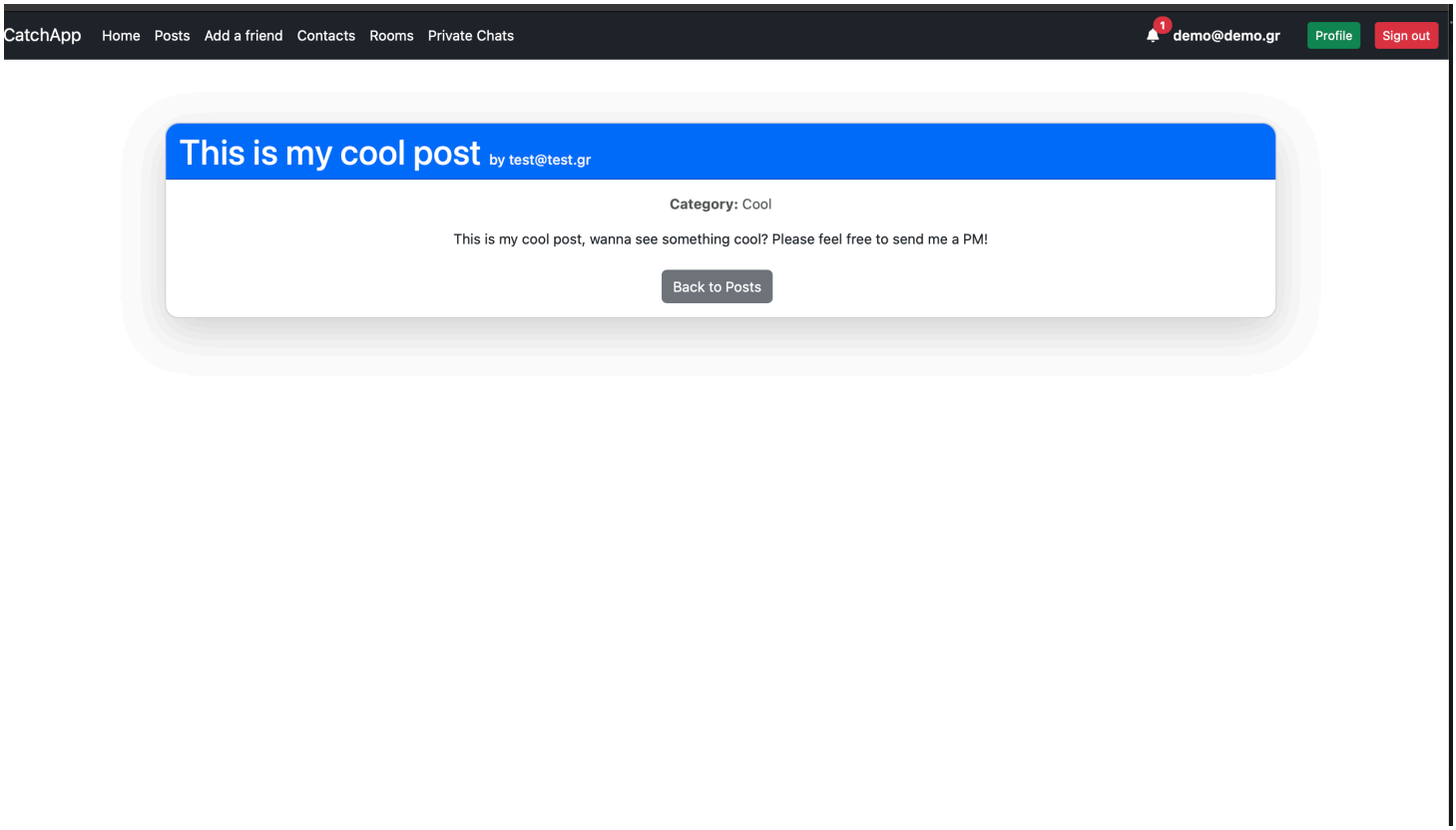
Main page

Posts overview

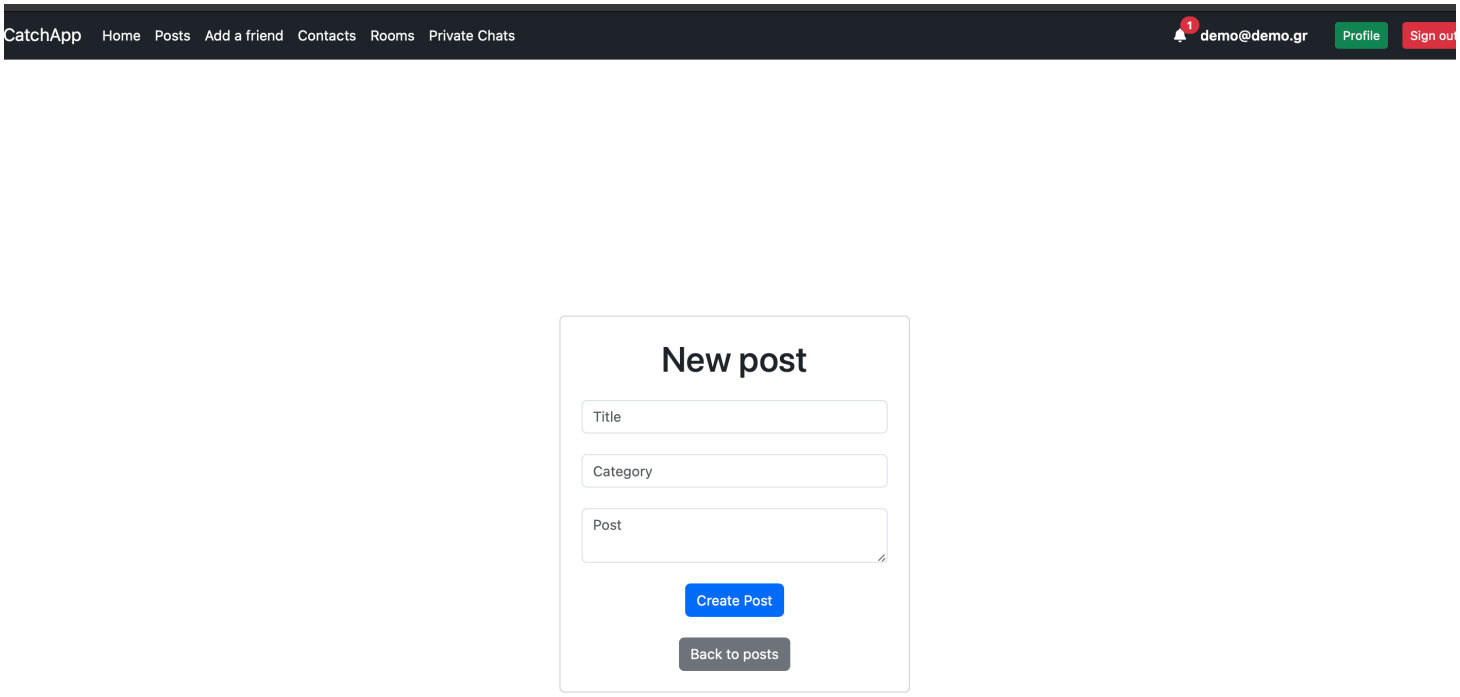
With the posts function, anyone can easily create, modify and delete their posts. To see all the posts available right now, you simply need to click the `posts` link in your hotbar or access them via the index page using the `view all posts` button.
Please note that you can only modify a post or even delete it *ONLY* if you are the owner of that post. The rest of the users, as seen in the images below, have no rights to modify or delete the posts made by another user as they can only view them.
You can also easily create a post if you want to by clicking on the `New posts` button.



Viewing all the posts



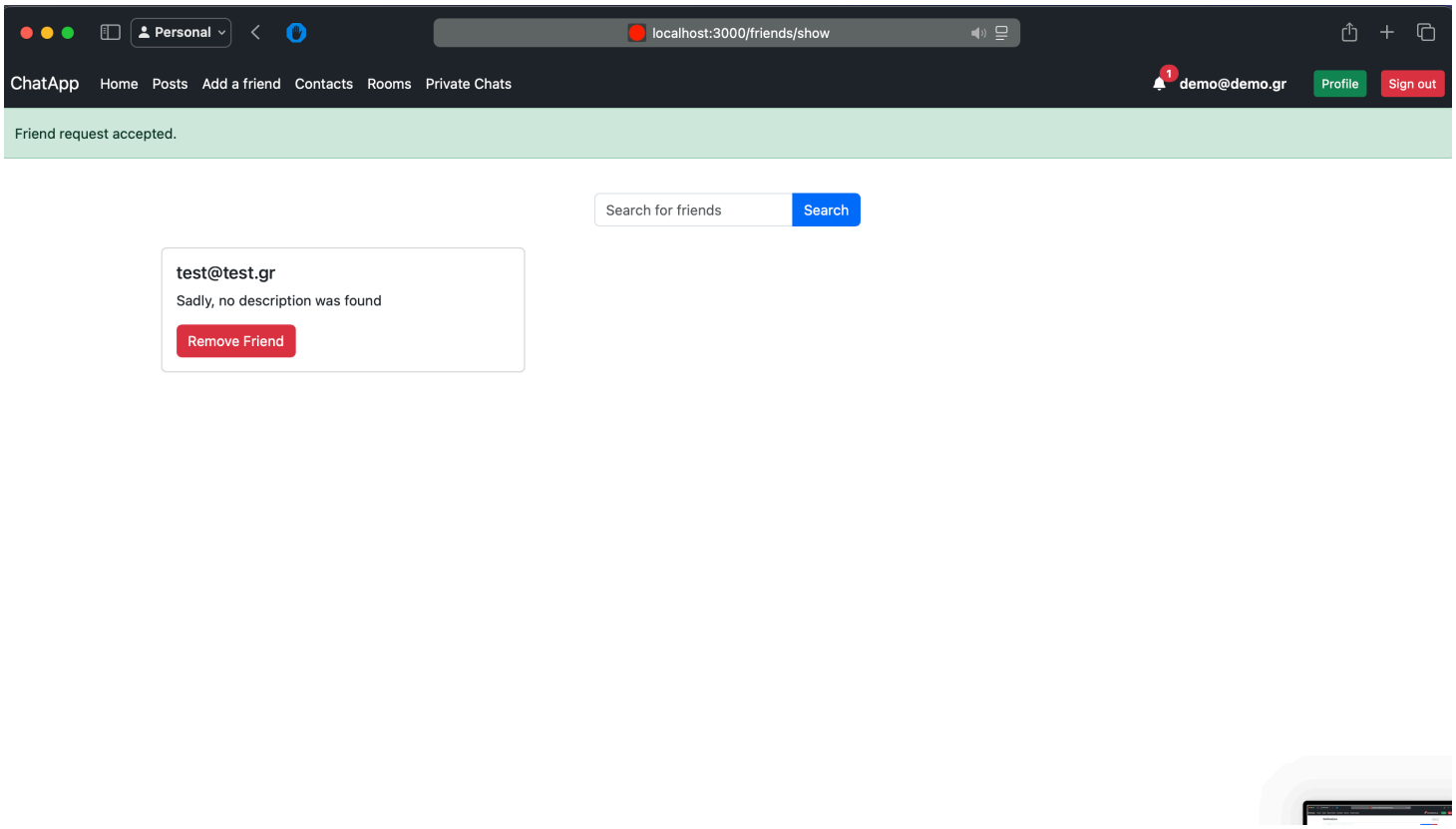
Viewing a specific post



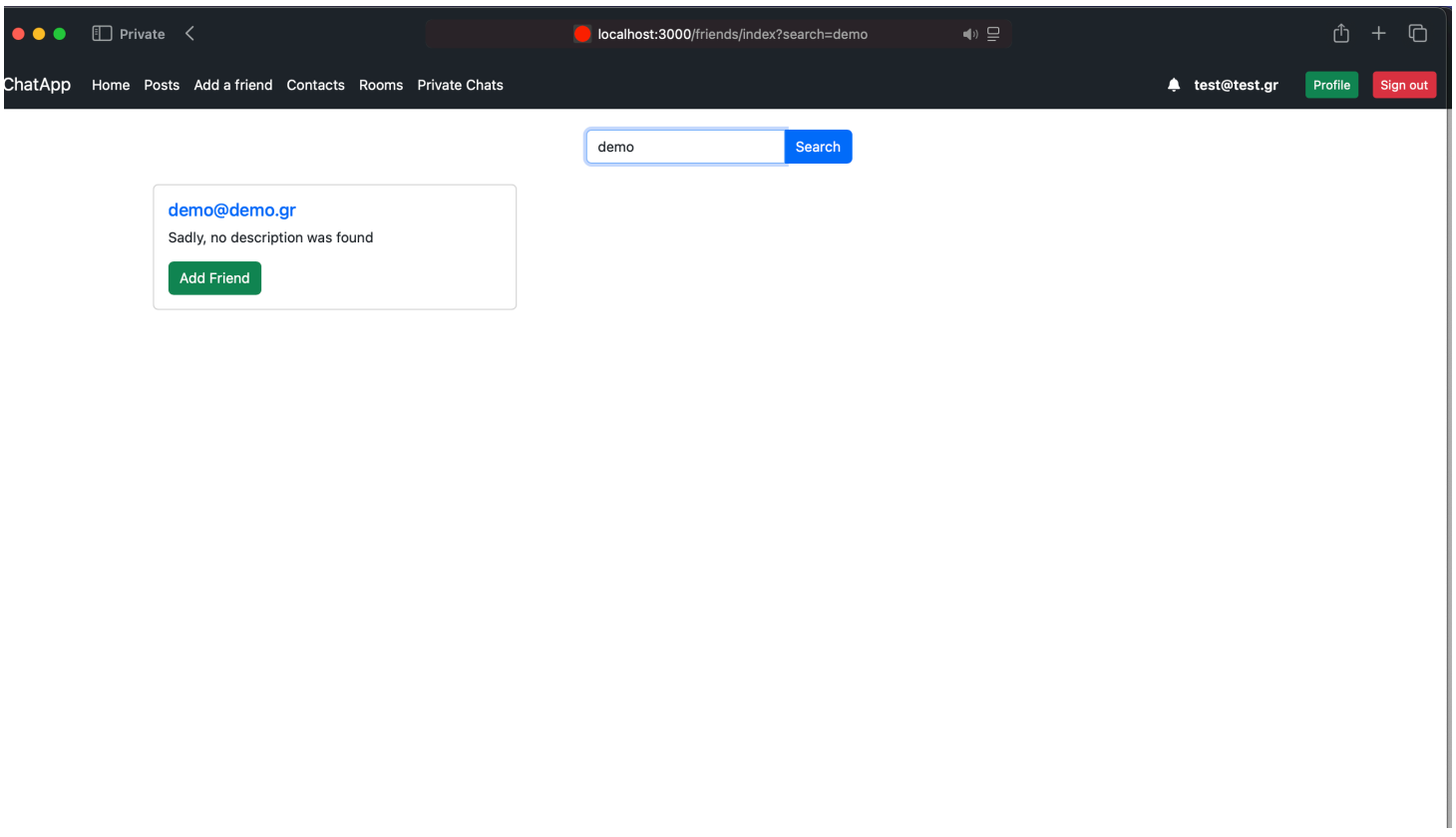
Creating a new post

Friends functionality

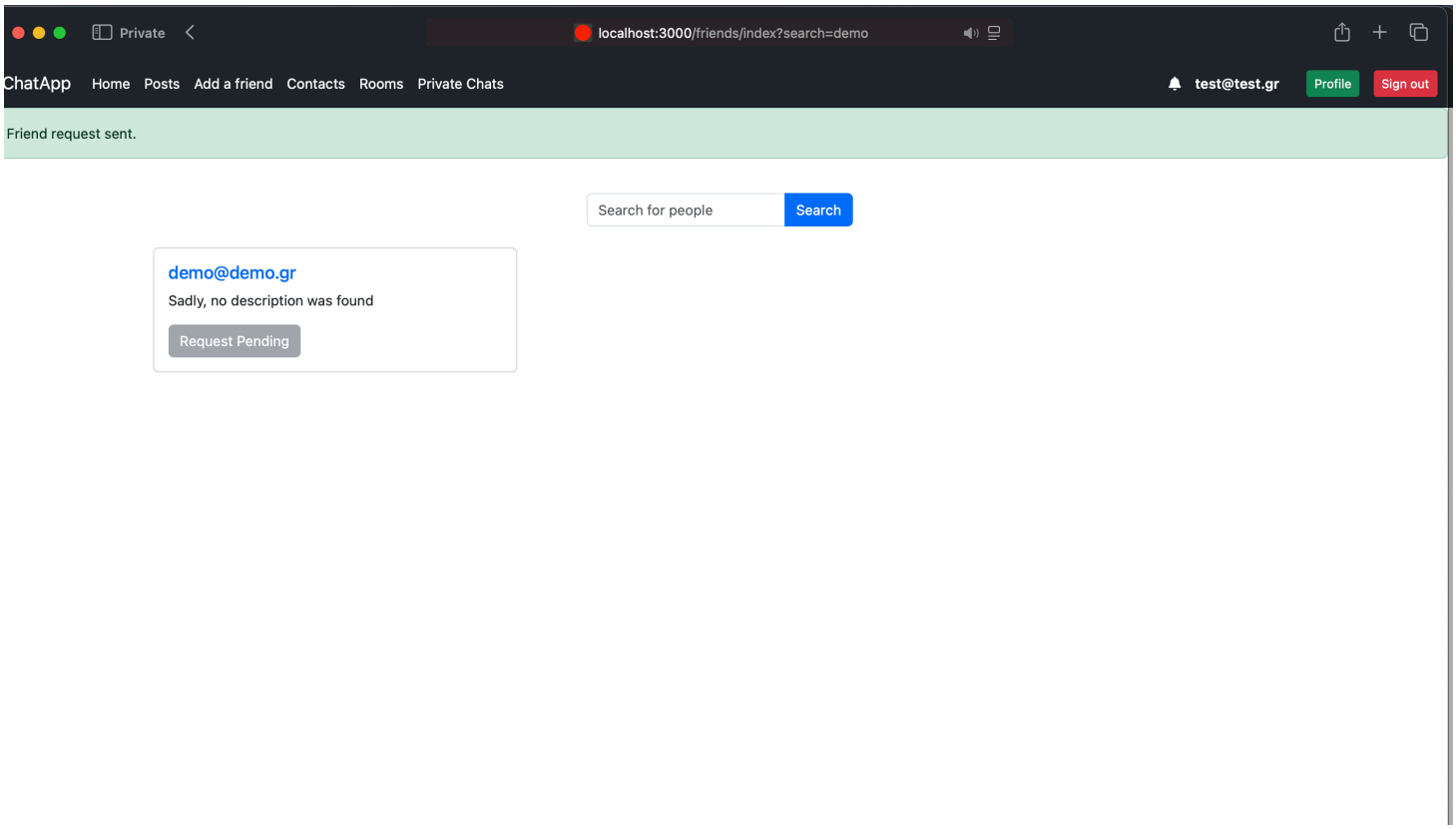
As we mentioned above, the app offers a cool friend functionality! As seen in the images below, anyone can search for a friend - by looking up their email, or check their contacts and see any updates on their friend requests!
You can find friends by clicking the `add a friend` link in the hotbar or click the `add a friend` button in the main page.
Either way, you are redirected to the search a friend page, where you can easily type anything and relevant users will be loaded, thus giving you the option to spectate their profile - by clicking on their email - or to add them as a friend!



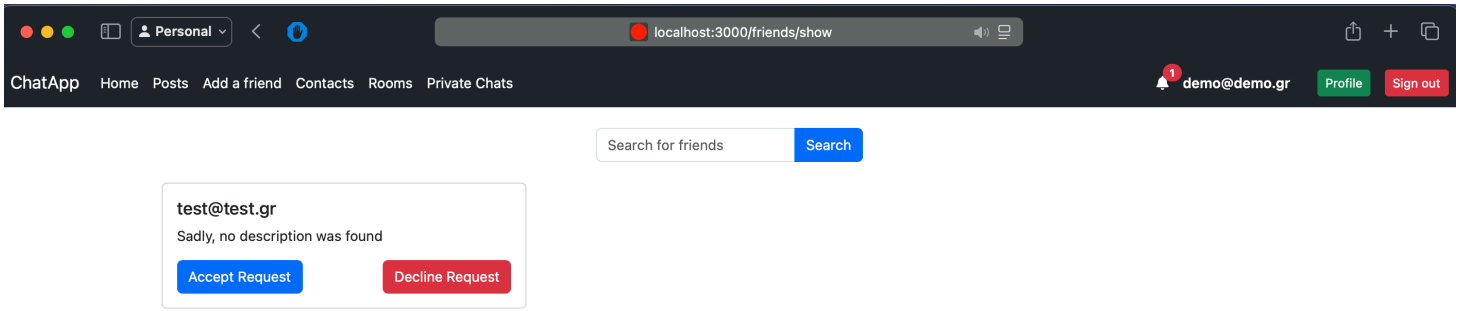
Contacts page



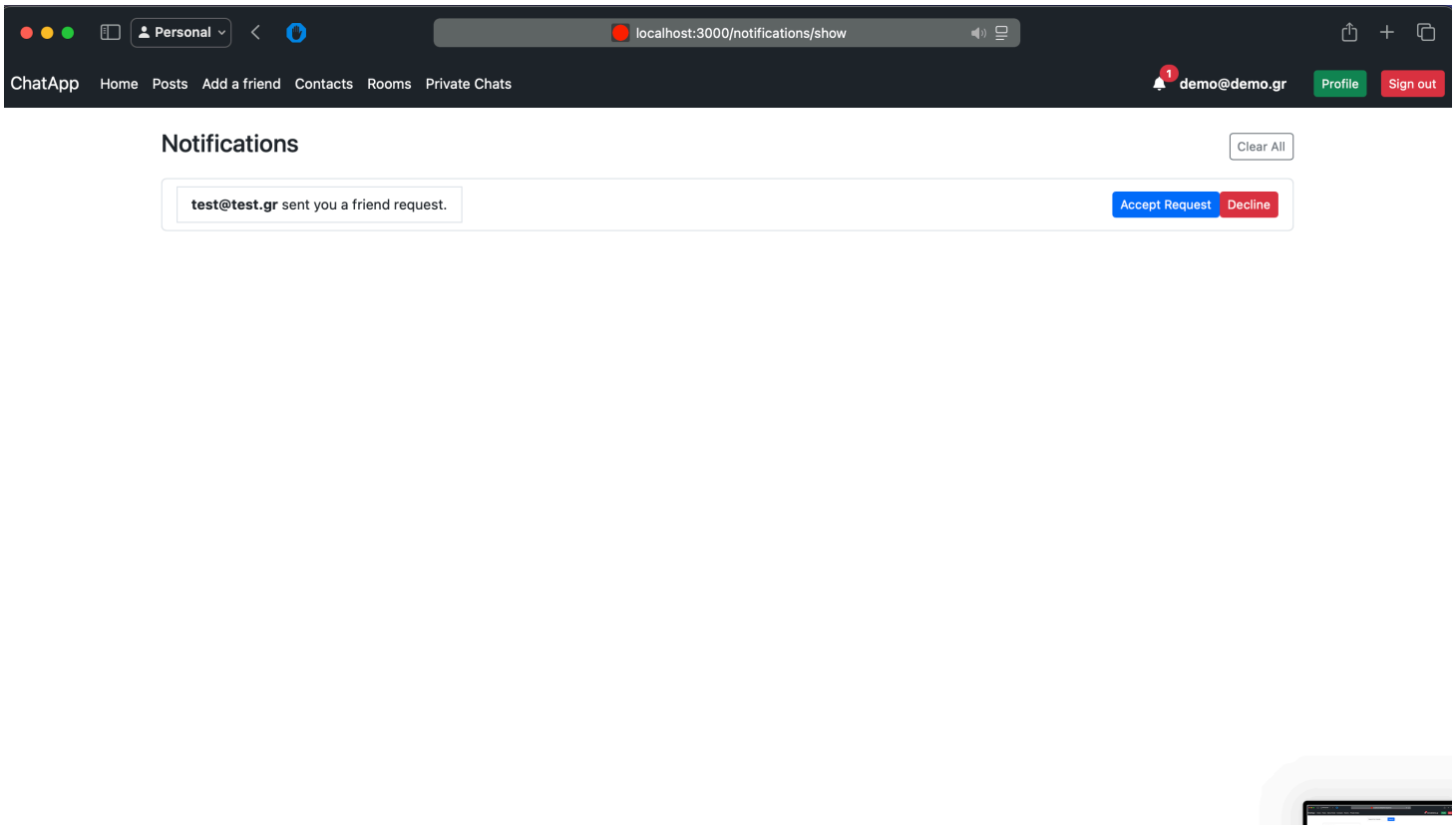
Search a friend page



Friend request sent!



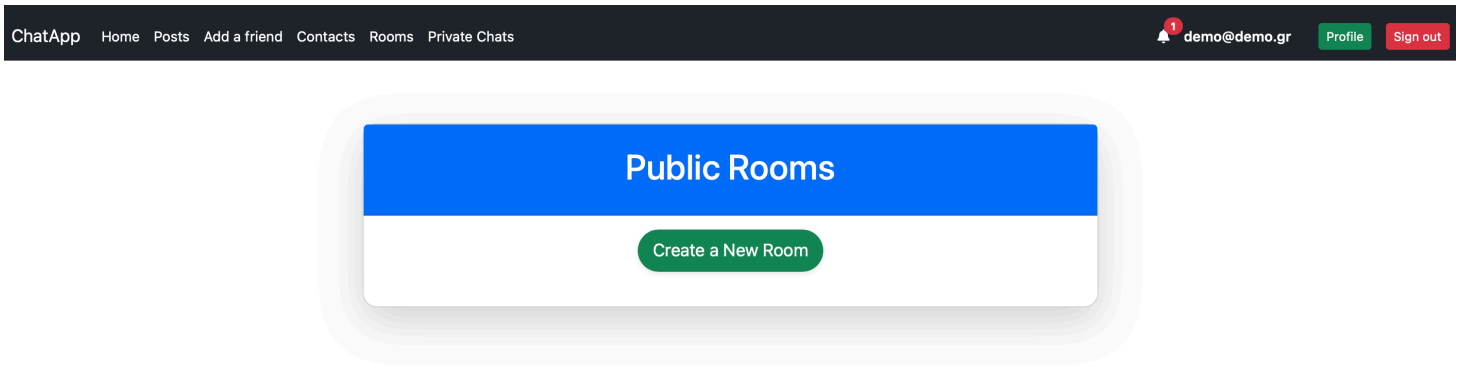
Receiving a friend request as another user!



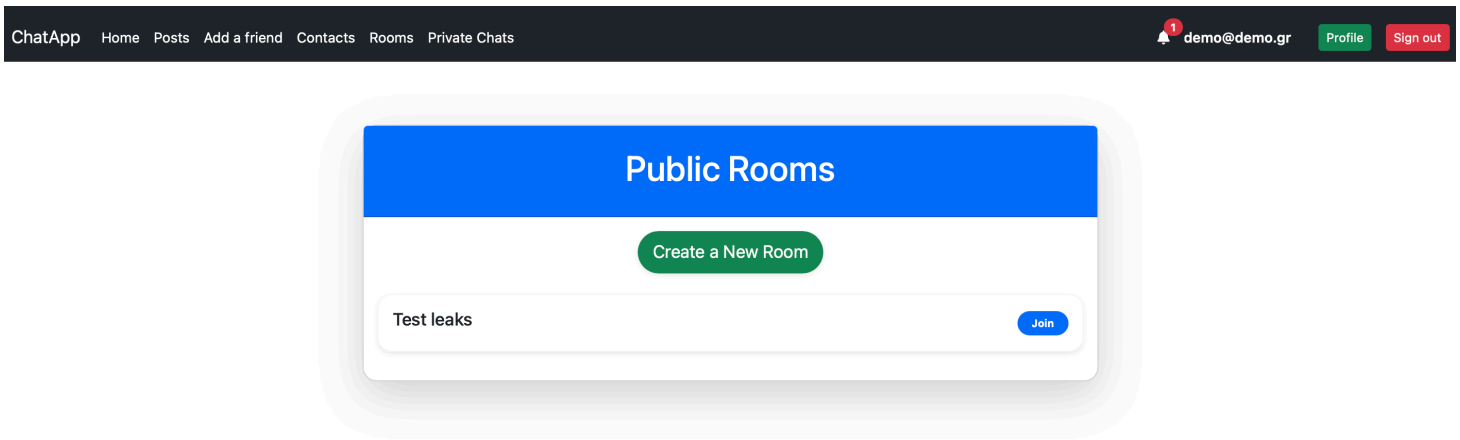
Receiving a friend request notification!

Group chats

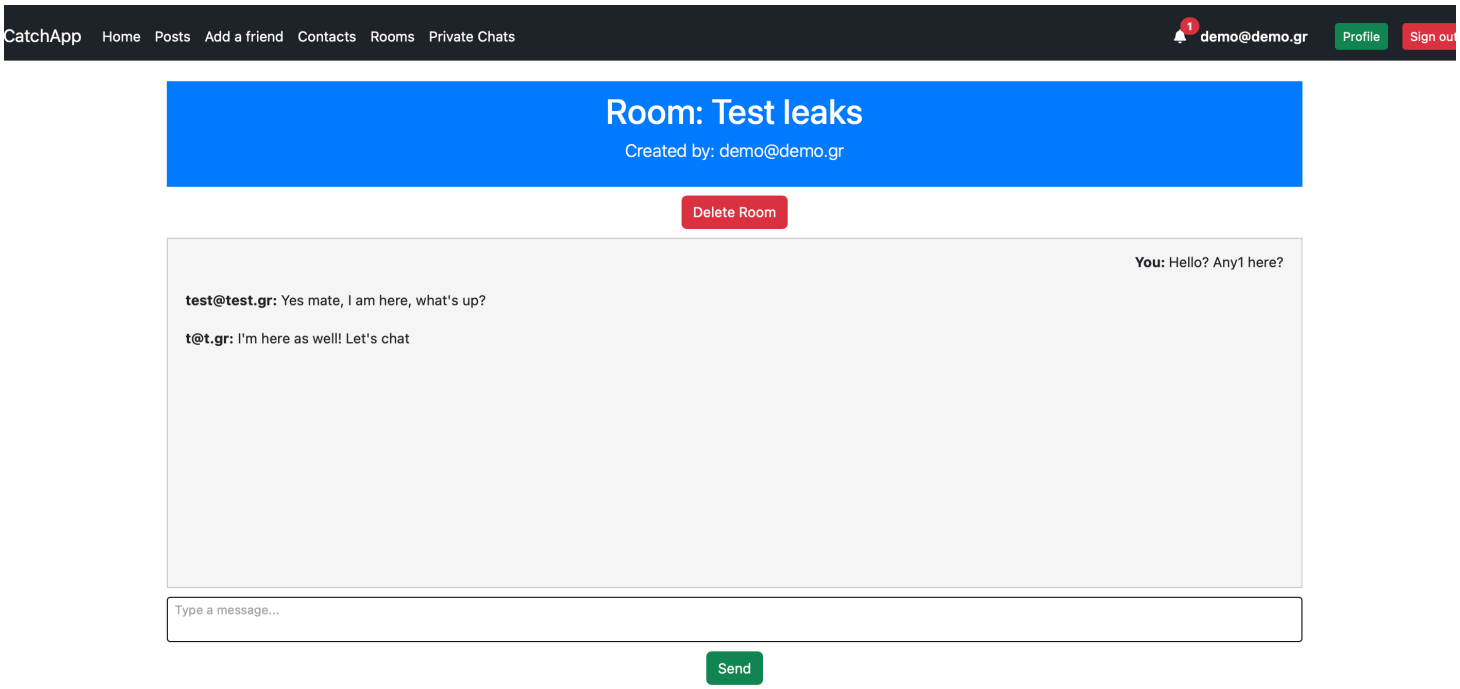
To access the currently available and open group chats, you can click on the `Rooms` button in your hotbar. Doing so, will redirect you to the group chats page. Here, anyone can join the available group chats or even create a new one!
Please note that the group chats support real time messaging - using action cable! Cool right? 😄
The messages that are sent by you will appear on the right side of the chat, whereas the messages that are sent by other users will appear on the left!



Empty group chat page



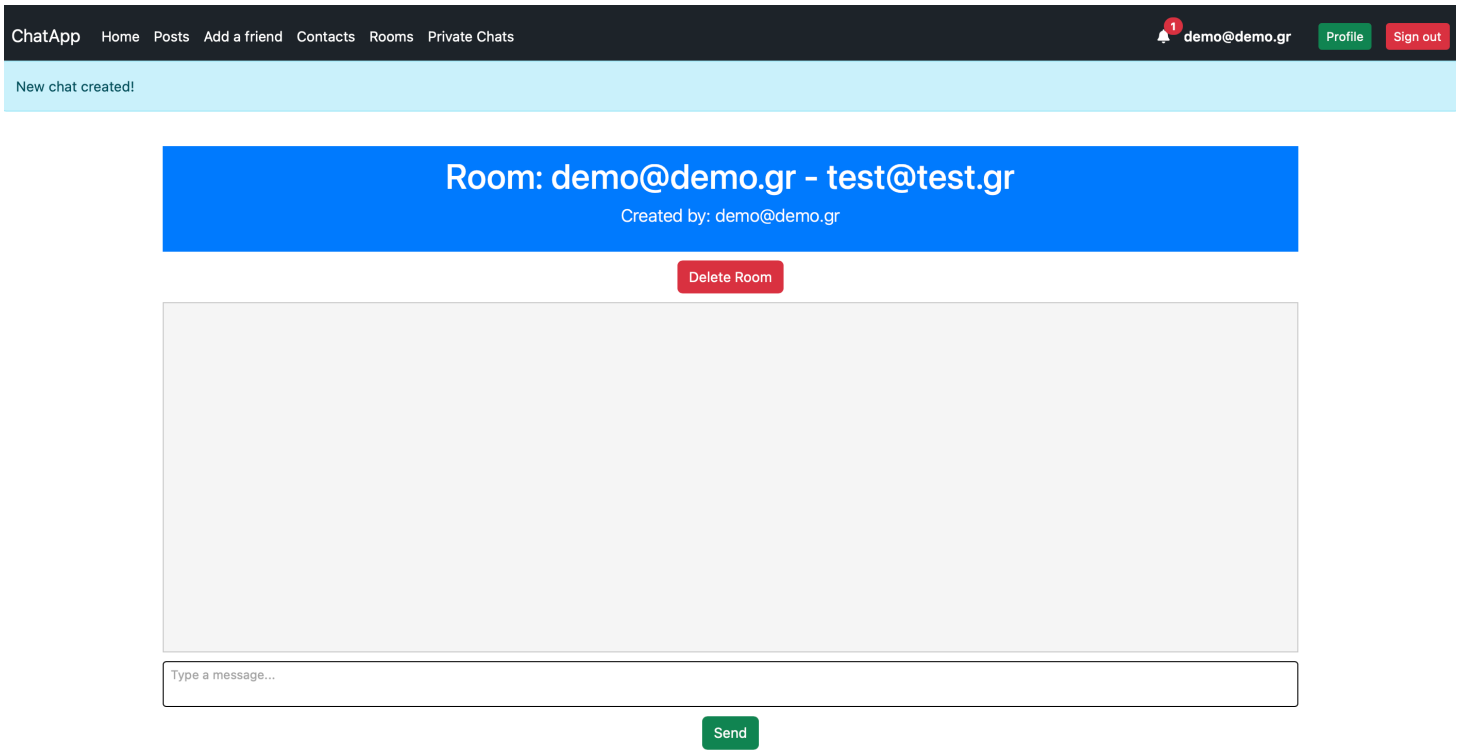
Group chat page showing the available rooms!



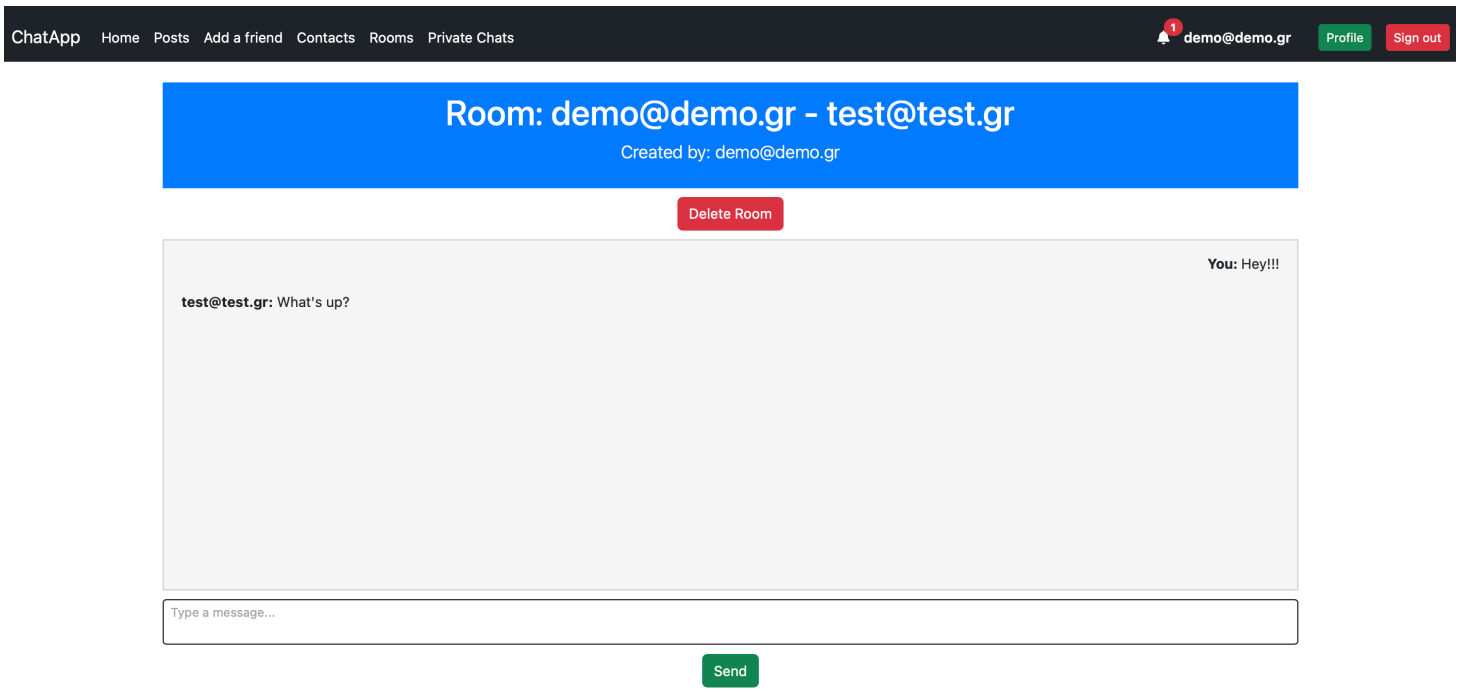
A random group chat

Personal messaging and profiles

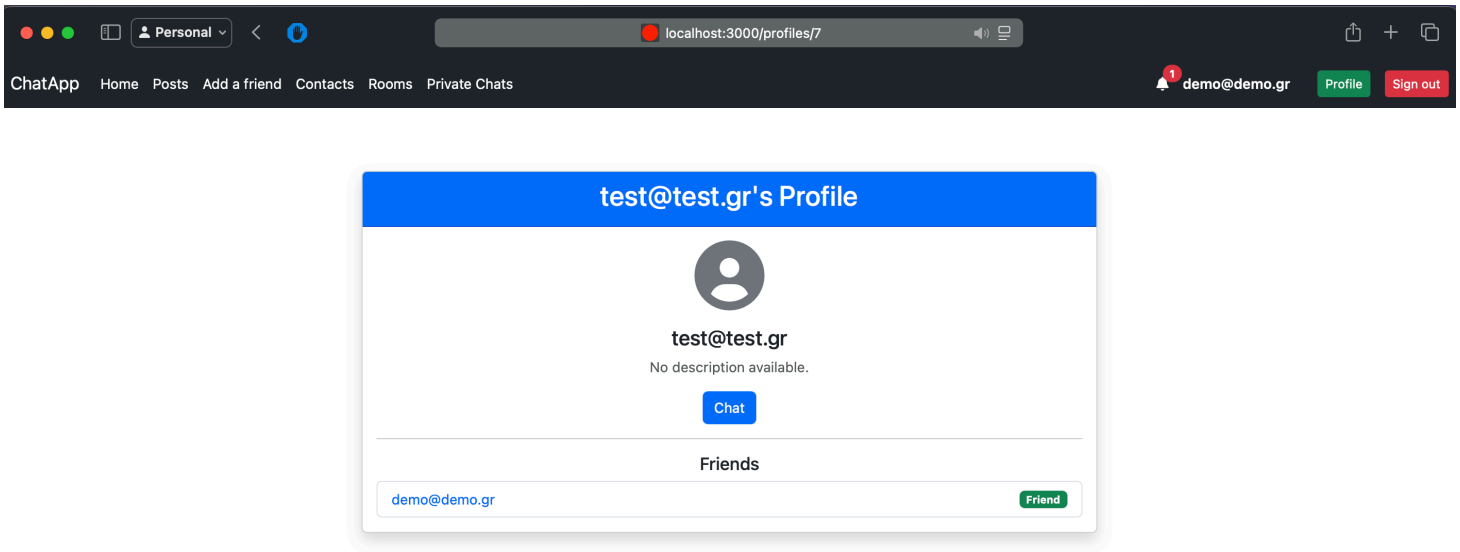
The implementation of personal messaging is an extent of the public rooms feature. Personal chats are essentially public rooms that have a parameter `Private: True`. This enables the application to distinguish between these two and give access only to the required parties. For example, if you try to chat with a user for the first time, we create a new private chat - based on the aforementioned logic - and only you and the other user can join the chat.
To chat with someone you need to simply click on their email - wherever it appears, all messages all clickable - and you will be presented with their profile. Their profile contains information about them - their description and their friends - while also providing a `chat` button.
Clicking this creates the room - if not already created by the other user or in the past - and you are free to chat with the other user!



Freshly created private chat



Privately chatting with a friend



A user's profile

API documentation

Getting started

1. We need to create the Todo & Todo_item models. Run this in your terminal

```
rails g model Todos title:string description:text
rails g model TodoItem todo:references content:text completed:boolean
rails db:migrate
```

2. We also need to make a migration to add a user reference in TODO model

```
rails generate migration AddUserIdToTodo user:references
rails db:migrate
```

3. Expose the necessary endpoints in config/routes.rb.

```
Rails.application.routes.draw do
    post "/signup", to: "registrations#create"
    post "/auth.login", to: "sessions#create"
    get "/auth.logout", to: "sessions#destroy"

resources :todo do
  resources :todo_item, only: [ :show, :create, :update, :destroy ]
    end
end
```

This will expose the endpoints as follows:

- [x] POST /signup -> Signup
- [x] POST /auth.login -> Login
- [x] GET /auth/logout -> Logout
- [x] GET /todos -> List all todos and todo items
- [x] POST /todos -> Create a new todo
- [x] GET /todos/:id -> Get a todo
- [x] PUT /todos/:id -> Update a todo
- [x] DELETE /todos/:id -> Delete a todo and its items
- [x] GET /todos/:id/items/:iid -> Get a todo item
- [x] POST /todos/:id/items -> Create a new todo item
- [x] PUT /todos/:id/items/:iid -> Update a todo item
- [x] Delete /todos/:id/items/:iid -> Delete a todo item

4. To handle the api requests we need to setup the necessary controllers for each action. Lets create them now 😄

    ```
    rails g controller Todos
    rails g controller TodoItem
    rails g controller Sessions
    rails g controller Registrations
    ```

    This will create 4 new controllers that will let us implement our nice functions.

5. Setting up the required functions for each controller
   Each of the following controllers may have the following functions that are associated with different HTTP requests.
    i. `index` function is the GET request
    ii. `create` function is the post request
    iii. `show` function is the GET request with a specific ID
    iv. `update` function is the PUT request

v. `destroy` function is the DELETE request.

todo_controller

This controller will provide functionality for everything that involves the creation, presentation, update and deletions of the todo lists.

```ruby
class TodoController < ApplicationController
    before_action :authenticate_user!
    def index
        @todo = current_user.todos
        render json: @todo
    end
    def create
        @todo = current_user.todos.create(todo_params)
        if @todo.save
        render json: @todo, status: :created
        else
        render json: @todo.errors, status: :unprocessable_entity
        end
    end

    def show
        @todo = current_user.todos.find(params[:id])
        render json: @todo.as_json(include: :todo_items)
    end

    def update
        @todo = current_user.todos.find(params[:id])
        if @todo.update(todo_params)
        render json: @todo
        else
        render json: @todo.errors, status: :unprocessable_entity
        end
    end

    def destroy
        @todo = current_user.todos.find(params[:id])
        @todo.destroy
        head :no_content
    end

    private

    def todo_params
        params.require(:todo).permit(:title, :description)
    end
end
```

todo_item_controller

This controller will provide functionality for everything that involves the creation, presentation, update and deletions of the TODO items that are present in each TODO list.

```ruby
class TodoItemController < ApplicationController
    before_action :authenticate_user!

    def show
        @todo_item = current_user.todos.find(params[:todo_id]).todo_items.find(params[:
        render json: @todo_item
    end

    def create
        @todo = current_user.todos.find(params[:todo_id])
        @todo_item = @todo.todo_items.create(todo_item_params)
        if @todo_item.save
        render json: @todo_item, status: :created
        else
        render json: @todo_item.errors, status: :unprocessable_entity
        end
    end

    def update
        @todo_item = current_user.todos.find(params[:todo_id]).todo_items.find(params[:
        if @todo_item.update(todo_item_params)
        render json: @todo_item
        else
        render json: @todo_item.errors, status: :unprocessable_entity
        end
    end

    def destroy
        @todo_item = current_user.todos.find(params[:todo_id]).todo_items.find(params[:
        @todo_item.destroy
        head :no_content
    end

    private

    def todo_item_params
        params.require(:todo_item).permit(:content, :completed)
    end
end
```

sessions_controller

This controller is responsible for handling the sessions of each user. This means that it is responsible for the creation and deletion of each session. It does so by utilizing the `devise's` User table and functions.

```
class SessionsController < ApplicationController
    def create
        user = User.find_for_database_authentication(email: params[:email])
        if user&.valid_password?(params[:password])
        sign_in user
        render json: { message: "Logged in successfully" }
        else
        render json: { error: "Invalid email or password" }, status: :unauthorized
        end
    end

    def destroy
        sign_out current_user
        render json: { message: "Logged out successfully" }
    end
end
```

registrations_controller

This controller is responsible for the sign-ups of each user. This controller also utilizes `devise's` User table and functions.

```
class RegistrationsController < ApplicationController
    def create
        user = User.new(user_params)
        if user.save
        render json: { message: "User created successfully" }, status: :created
        else
        render json: { error: "Error creating user" }, status: :unprocessable_entit
        end
    end

    private

    def user_params
        params.require(:user).permit(:email, :password, :password_confirmation)
    end
end
```

Final steps

The final step before we are ready is to make sure our models are ok. Make sure that your models look like this.

1. User model

```
class User < ApplicationRecord
    has_many :todos
    #Other stuff ...
end
```

2. Todo model

```
class Todo < ApplicationRecord
    has_many :todo_items
    belongs_to :user
end
```

3. Todo item model

```
class TodoItem < ApplicationRecord
    belongs_to :todo
end
```

Testing the APIs

Custom python script

To test the `api` functions we used a script that we made that is named `api_script.py`

```
import requests
from time import sleep
session_key = None

def create_todo():
    if session_key:
        title = input('Title:')
        Description = input('Description:')
        response = requests.post('http://localhost:3000/todo',cookies={'_app_session':s
          "todo[title]":title,
          "todo[description]":Description
        })
        print(response.text)
    else:
        print("No session key found, please login first")

def show_todo_id():#this also gets todo_items
    if session_key:
        id = input('ID of the todo:')
        response = requests.get(f'http://localhost:3000/todo/{id}',cookies={'_app_sessi
        print(response.text)
    else:
        print("No session key found, please login first")

def update_todo():
    if session_key:
        id = input("ID:")
        title = input('Title (blank for no change):')
        Description = input('Description (blank for no change):')
        data = {}
        if title:
            data["todo[title]"] = title
        if Description:
            data["todo[description]"] = Description
        response = requests.put(f'http://localhost:3000/todo/{id}',cookies={'_app_sessi
        print(response.text)
    else:
        print("No session key found, please login first")

def delete_todo():
    if session_key:
        id = input('ID:')
        response = requests.delete(f'http://localhost:3000/todo/{id}',cookies={'_app_se
```

```
        if response.status_code == 204:
            print('Deleted!')
        else:
            print(response.text)
    else:
        print("No session key found, please login first")


def get_todo_item():
    if session_key:
        id = input('ID of the todo:')
        id2 = input('ID of the todo_item:')
        response = requests.get(f'http://localhost:3000/todo/{id}/todo_item/{id2}',cook
        print(response.text)
    else:
        print("No session key found, please login first")

def create_todo_item():
    if session_key:
        id = input('TODO ID:')
        while True:
            content = input('Content:')
            completed = input('Completed? (y/n)')
            if completed == 'y' or completed == 'n':
                break
            else:
                print('completed must be y or n')
        if completed == 'y':
            completed = True
        else:
            completed = False

        data = { "todo_item[content]":content, "todo_item[completed]":completed }
        response = requests.post(f'http://localhost:3000/todo/{id}/todo_item',cookies={
        print(response.text)
    else:
        print("No session key found, please login first")

def update_todo_item():
    if session_key:
        data = {}
        id = input('TODO ID:')
        id2 = input('TODO_ITEM ID:')
```

```
        while True:
            content = input('Content: (blank to skip)')
            completed = input('Completed? (y/n) (blank to skip)')
            if completed == 'y' or completed == 'n' or completed == '':
                break
            else:
                print('completed must be y or n or blank')
        if completed == 'y':
            data['todo_item[completed]'] = True
        if completed == 'n':
            data['todo_item[completed]'] = False
        if content:
            data['todo_item[content]'] = content
        if data:
            response = requests.put(f'http://localhost:3000/todo/{id}/todo_item/{id2}',
            print(response.text)
        else:
            print('no change provided')
    else:
        print("No session key found, please login first")

def delete_todo_item():
    if session_key:
        id = input('TODO ID:')
        id2 = input('TODO_ITEM ID:')
        response = requests.delete(f'http://localhost:3000/todo/{id}/todo_item/{id2}',c
        if response.status_code == 204:
            print('Deleted!')
        else:
            print(response.text)
    else:
        print("No session key found, please login first")

def signup():
    #http POST http://localhost:3000/signup 'user[email]=l@l.gr' 'user[password]=123456
    email = input('Email:')
    password = input('Password:')
    password_eval = input('Password confirmation:')
    response = requests.post("http://localhost:3000/signup",data = {
    "user[email]": email,
    "user[password]": password,
    "user[password_confirmation]": password_eval
})
```

```
    print(response.text)


def login():
    #http POST http://localhost:3000/auth.login email="n@n.gr" password="123456"
    global session_key
    email = input('Email:')
    password = input('Password:')
    response = requests.post("http://localhost:3000/auth.login",json={"email":email, "p
    print(response.text)
    session_key = response.cookies.get_dict()['_app_session']

def logout():
    print(requests.get('http://localhost:3000/auth.logout').json())

def get_all_todo():
    #http get http://localhost:3000/todo Cookie=' _app_session=bYNcJ58EJWUHQY%2BftD9pPy
    if session_key:
        response = requests.get('http://localhost:3000/todo',cookies={'_app_session':se
        print(response.text)
    else:
        print("No session key found, please login first")

try:
    while True:
        go = False
        go2 = False
        num = input('''1. Login\n2. Sign up\n3. Logout\n4. List all todos\n5. Create a
        try:
            num = int(num)
            go = True
        except ValueError:
            print('Please provide a number!')
        try:
            if num < 1 or num > 13:
                raise Exception
            go2=True
        except Exception:
            print('Number must be from 1 to 12')

        if go and go2:
            match num:
                case 1:
```

```
                    login()
                case 2:
                    signup()
                case 3:
                    logout()
                case 4:
                    get_all_todo()
                case 5:
                    create_todo()
                case 6:
                    show_todo_id()
                case 7:
                    update_todo()
                case 8:
                    delete_todo()
                case 9:
                    get_todo_item()
                case 10:
                    create_todo_item()
                case 11:
                    update_todo_item()
                case 12:
                    delete_todo_item()
                case 13:
                    logout()
                    exit()
                case _:
                    print('Something went wrong')

            input('Continue?...')
    except KeyboardInterrupt:
        exit()
```

This serves as a testing terminal application that allows the user to test all the functions of our `REST API`

How the API works

For all the functions mentioned bellow you can use `HTTPie CLI` or our `python script`. We will demonstrate how to use the API with `HTTPie CLI` tool, because the use of our script is really easy.

Login

The first thing you must do is, of course, to login to receive a session.

```
http POST http://localhost:3000/auth.login email="demo@demo.gr" password="demoPass"
```

By doing so the request returns a response like this

```
HTTP/1.1 200 OK
Content-Length: 36
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"d7798a9ef354a1fcb018db70449e0f38"
referrer-policy: strict-origin-when-cross-origin
server-timing: sql.active_record;dur=12.87, start_processing.action_controller;dur=0.01
set-cookie: _app_session=DaAOzKatEfTuFIDuBVKCdVmF1YbQfZBhVTuh%2F%2BPpoLKwjwsQHpHeFFxHk%
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 944f8ca0-84fe-4e79-9c69-bb4667300539
x-runtime: 0.418303
x-xss-protection: 0

{
    "message": "Logged in successfully"
}
```

Please notice the `set-cookie:` variable in the response. This is our session token. This token can be used to access the rest of the app - as it serves as our authentication. You must store this somewhere in order to proceed.

Sign-up

If you don't have an account, yet, you can create one like so

```
http POST http://localhost:3000/signup 'user[email]=demo@demo.gr' 'user[password]=demoP
```

Please note that providing `user[attribute]` instead of `attribute` is really important. It won't work otherwise!

This will return a message like this, so you know it is successful

```
HTTP/1.1 201 Created
Content-Length: 39
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"8778e358444656690b5d998f57f47a99"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, sql.active_record;dur=3.22,
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 73e44ba5-4e99-4b6c-a12e-84596188f984
x-runtime: 0.277723
x-xss-protection: 0

{
    "message": "User created successfully"
}
```

Logout

Logging-out is also crucial! To do so after you are done using the api, you can simply send a GET request to auth/logout to delete your session.

```
http GET http://localhost:3000/auth.logout
```

This should return something like this

```
HTTP/1.1 200 OK
Content-Length: 37
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"28b8f3fe95843547dbb631cb6c66aaff"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, process_action.action_contr
set-cookie: _app_session=b5F12fxEKY1b1ggVua6NZhTnGsZD0OqcXivh8wd%2BqYnVQlU3FtNm1IP2gBRw
vary: Accept
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: b29f94b9-f2db-48bf-99ef-566c8004bd1b
x-runtime: 0.048605
x-xss-protection: 0

{
    "message": "Logged out successfully"
}
```

Listing all TODOs

To list all of your current TODO lists you can make a get request to /todo like so

```
http GET http://localhost:3000/todo Cookie:'_app_session{Your app session}'
```

This will return a list containing your todo items like so

```
HTTP/1.1 200 OK
Content-Length: 269
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"16ed39b1c75aa7848f6f4ed7b080ce58"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, sql.active_record;dur=2.59,
set-cookie: _app_session=K4frLgE9yrWhN0Paoery5ylruVcFo%2B7M4U2SKEnXYpWQKZe7levJB%2FNu0k
vary: Accept
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: e5f12fe9-4ec2-43f9-a453-dc7bb1f174db
x-runtime: 0.058747
x-xss-protection: 0

[
    {
        "created_at": "2025-01-23T21:37:02.687Z",
        "description": "t",
        "id": 4,
        "title": "t",
        "updated_at": "2025-01-23T21:37:02.687Z",
        "user_id": 3
    },
    {
        "created_at": "2025-01-23T22:14:26.897Z",
        "description": "1234",
        "id": 5,
        "title": "todo",
        "updated_at": "2025-01-23T22:14:26.897Z",
        "user_id": 3
    }
]
```

Create a new TODO

To create a new todo you must make a post request to `/todo` like so

```
http POST http://localhost:3000/todo 'todo[title]'='Demo' 'todo[description]'='DemoDesc
```

This will return something like this

```
HTTP/1.1 201 Created
Content-Length: 147
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"a3b78460d3cb523819a58ec04def9fcb"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.00, sql.active_record;dur=7.78,
set-cookie: _app_session=l%2Fmf8ln%2BVPL9AO2ljMG39BgPrmfRd%2Bpmeu920vh%2BblX8EwD7%2FbdH
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 47afa720-2765-4046-86ef-d14496353db8
x-runtime: 0.052266
x-xss-protection: 0

{
    "created_at": "2025-01-24T13:40:24.660Z",
    "description": "DemoDescription",
    "id": 6,
    "title": "Demo",
    "updated_at": "2025-01-24T13:40:24.660Z",
    "user_id": 3
}
```

Get a specific TODO list with its TODO items

You can access any of your TODO lists by using a get request in /todo/id like so

```
http get http://localhost:3000/todo{ID} Cookie:"_app_session{Your session key}'
```

This will return something like

```
HTTP/1.1 200 OK
Content-Length: 152
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"3bee32c6b65694be75e5f289e2915184"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, sql.active_record;dur=0.92,
set-cookie: _app_session=BoAnhYvZz7IC4yu9ttz1km2D400OxdRcDqW6fnC%2FJev2KnsMcL4MIHgaDGjy
vary: Accept
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: dddcc658-783e-4b0d-8703-31c2938cdc10
x-runtime: 0.072019
x-xss-protection: 0

{
    "created_at": "2025-01-23T22:14:26.897Z",
    "description": "1234",
    "id": 5,
    "title": "todo",
    "todo_items": [],
    "updated_at": "2025-01-23T22:14:26.897Z",
    "user_id": 3
}
```

Update a TODO

To update a todo you can simply make a PUT request to /todo/id with the field you want to change like so

```
http PUT http://localhost:3000/todo{ID} 'todo[title]'='ChangedDemo' Cookie:"_app_sessio
```

This will return something like this

```
HTTP/1.1 200 OK
Content-Length: 143
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"804fbce768e779921810dfcc5ac6b47d"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.03, sql.active_record;dur=2.59,
set-cookie: _app_session=UmJuMBnQGucFt8LPSMW3UgaMvF4%2BJiKvcx5G0Z5an7QHhy7W05SNQME5GGTp
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 505689f2-3fd9-4a06-b46d-4005527d1021
x-runtime: 0.064931
x-xss-protection: 0

{
    "created_at": "2025-01-23T22:14:26.897Z",
    "description": "1234",
    "id": 5,
    "title": "ChangedDemo",
    "updated_at": "2025-01-24T13:45:41.851Z",
    "user_id": 3
}
```

Please keep in mind that you must only include the field you want to change. If you include a field it WILL get changed! For example if you type `'todo[title]'='ChangedDemo'` , it will only change the title of your TODO NOT the description and vice versa.

Deleting a todo and its items

To delete a todo and its items you can simply send a `DELETE` request to /todos/id.

```
http DELETE http://localhost:3000/todo{ID} Cookie:"_app_session{your session key}"
```

Creating a new todo item

To create a new TODO item you can do so by making a `POST` request in todo/{id}/todo_item.

```
http POST http://localhost:3000/todo/{ID}/todo_item "todo_item[content]"='content' "tod
```

Note that you `todo_item[completed]` can only be True or False.
This will return something like this

```
HTTP/1.1 201 Created
Content-Length: 137
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"2f84c529cdb09d58745f9fa31088754e"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, sql.active_record;dur=1.67,
set-cookie: _app_session=fpsgTBz6CteR6Y%2Fesadh6UZlX1eDZf5URgWbUNWmZzNZ7MOZTQQNpAlNuGIT
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 26fcd662-b661-4f59-90e8-0a5e6dab251b
x-runtime: 0.040519
x-xss-protection: 0

{
    "completed": true,
    "content": "content",
    "created_at": "2025-01-24T13:55:59.178Z",
    "id": 3,
    "todo_id": 4,
    "updated_at": "2025-01-24T13:55:59.178Z"
}
```

Get a specific TODO item

To get a specific TODO item you must make a `GET` request to todo/{ID}/todo_item{ID2} where ID2 is the id of the TODO item.

```
http GET http://localhost:3000/todo/{ID}/todo_item/{ID2} Cookie:"_app_session={Your ses
```

This will return something like this

```
HTTP/1.1 200 OK
Content-Length: 137
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"2f84c529cdb09d58745f9fa31088754e"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, sql.active_record;dur=0.17,
set-cookie: _app_session=o0v43%2FR%2BjNjaLaPqnsyArzNDG6HLtoQGd1TAa1TaoiwFZQKLsqzK7mLddE
vary: Accept
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 7ad8ac64-9923-4298-9116-bb932ff2d1c5
x-runtime: 0.036748
x-xss-protection: 0

{
    "completed": true,
    "content": "content",
    "created_at": "2025-01-24T13:55:59.178Z",
    "id": 3,
    "todo_id": 4,
    "updated_at": "2025-01-24T13:55:59.178Z"
}
```

Updating a todo item

To update a todo item you must make a `PUT` request to /todo/{id}/todo_item/{id2}

```
http PUT http://localhost:3000/todo/{ID}/todo_item/{ID2} "todo_item[completed]"=True Co
```

As we also saw in the other PUT request, include only the fields you want to be changed!
This will return something like this

```
HTTP/1.1 200 OK
Content-Length: 137
cache-control: max-age=0, private, must-revalidate
content-type: application/json; charset=utf-8
etag: W/"443231586ea9eee471804b76499dee45"
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.01, sql.active_record;dur=0.46,
set-cookie: _app_session=0%2FSKQwrXuN%2F8ZaV0YRcttJ4lFbXJ4XCTKhjzSAK0b9u%2B5BuE3H69FOfv
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 6adc4609-5656-4182-97fd-95bf8426117f
x-runtime: 0.057253
x-xss-protection: 0

{
    "completed": true,
    "content": "content",
    "created_at": "2025-01-24T13:55:59.178Z",
    "id": 3,
    "todo_id": 4,
    "updated_at": "2025-01-24T13:55:59.178Z"
}
```

Deleting a TODO item

To delete a TODO item from a todo list you must make a `DELETE` request to todo/{ID}/todo_item/{ID2}.

```
http DELETE http://localhost:3000/todo/{ID}/todo_item/{ID2} Cookie:"_app_session={Your
```

This will return something like this

```
HTTP/1.1 204 No Content
cache-control: no-cache
referrer-policy: strict-origin-when-cross-origin
server-timing: start_processing.action_controller;dur=0.02, sql.active_record;dur=1.46,
set-cookie: _app_session=KxqSg6hu4zcotZisVYPWLsGRyjjUCkwFkOfnX8vBjig8cYYtOfC%2FZhpurzL3
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 750f46b9-7a8c-462b-bf13-2c81423c03c9
x-runtime: 0.066243
x-xss-protection: 0
```

If the HTTP/1.1 return code is 204 (No Content) it means you successfully deleted the TODO item.

Technologies used

- Users -> Device gem
- Realtime messaging -> Action cable
- Notifications -> Noticed gem