Додаток А

Програмні коди

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNet.Mvc;
using SmartParking.ViewModels.ParkingDto;
using Mocoding.EasyDocDb.Common;
using Microsoft.AspNet.Authorization;
using SmartParking.Models;

namespace SmartParking.Api
{
    [Route("api/[controller]")]
    [Authorize(Roles = Constants.Roles.Admin)]
    public class ParkingManagmentController : Controller
    {
        private readonly IDocumentCollection<Parking> _rep;

        public ParkingManagmentController(IRepository rep)
        {
            _rep = rep.GetCollection<Parking>();
        }

        [HttpGet]
        public IEnumerable<ParkingResultDto> Get()
        {
```

```
            return _rep.GetAll().Select(p => new ParkingResultDto(p));
        }

        [HttpGet("{id}")]
        public ParkingResultDto Get(Guid id)
        {
            return new ParkingResultDto(_rep.GetAll().FirstOrDefault(i => i.Id == id));
        }

        [HttpPost]
        public ParkingResultDto Post([FromBody]ParkingDto parkingDto)
        {
            var parking = _rep.New();
            parking.Address = parkingDto.Address;
            parking.CostPerHour = parkingDto.CostPerHour;
            parking.Latitude = parkingDto.Latitude;
            parking.Longtitude = parkingDto.Longtitude;
            parking.CreateSpots(parkingDto.CountOfSpots);
            parking.SaveChanges();
            return new ParkingResultDto(parking);
        }

        [HttpDelete("{id}")]
        public bool Delete(Guid id)
        {
            _rep.Delete(i => i.Id == id);
            return true;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNet.Mvc;
using SmartParking.ViewModels.ParkingDto;
using Mocoding.EasyDocDb.Common;
using Microsoft.AspNet.Authorization;
using SmartParking.Models;
using SmartParking.ViewModels;

namespace SmartParking.Api
{
    [Route("api/[controller]")]
    [Authorize]
    public class ParkingController : Controller
    {
        private readonly IDocumentCollection<Parking> _rep;

        public ParkingController(IRepository rep)
        {
            _rep = rep.GetCollection<Parking>();
        }

        [HttpGet]
        public IEnumerable<ParkingResultDto> Get()
        {
            return _rep.GetAll().Select(p => new ParkingResultDto(p));
```

```
    }

    [HttpGet("{id}")]
    public ParkingResultDto Get(Guid id)
    {
        var dto = new ParkingResultDto(_rep.GetAll().FirstOrDefault(i => i.Id == id));
        var user = HttpContext.User.Identity.Name;
        foreach (var spot in dto.Spots)
        {
            spot.HasPermision = spot.History?.User == user;
        }

        return dto;
    }

    [HttpGet("{id}/lock/{spotId}")]
    public IActionResult Lock(Guid id, int spotId)
    {
        var parking = _rep.GetAll().FirstOrDefault(i => i.Id == id);
        var spot = parking.Spots.FirstOrDefault(i => i.Id == spotId);
        spot.Histories.Add(new History() {Start = DateTime.Now, User =
HttpContext.User.Identity.Name});
        spot.IsFree = false;
        parking.SaveChanges();
        return Ok();
    }

    [HttpGet("{id}/unlock/{spotId}")]
    public IActionResult Unlock(Guid id, int spotId)
    {
```

```
    var parking = _rep.GetAll().FirstOrDefault(i => i.Id == id);
    var spot = parking.Spots.FirstOrDefault(i => i.Id == spotId);
    spot.Histories.Last().Finish = DateTime.Now;
    spot.IsFree = true;
    parking.SaveChanges();
    return Ok();
}

[HttpGet("status")]
public HistoriesDto GetStatus()
{
    var user = HttpContext.User.Identity.Name;
    var parking = _rep.GetAll();
    var histories = new HistoriesDto();
    foreach (var p in parking)
    {
        foreach (var s in p.Spots)
        {
            foreach (var history in s.Histories.Where(i => i.User == user))
            {
                if (history.Finish != null)
                {
                    histories.Histories.Add(new HistoryDto()
                    {
                        Date = history.Finish.Value.ToString("F"),
                        Type = HistoryType.unlocking
                    });
                    histories.Past++;
                }
                else
```

```
                {
                    histories.Histories.Add(new HistoryDto()
                    {
                        Date = history.Start.ToString("F"),
                        Type = HistoryType.locking
                    });
                    histories.Active++;
                    histories.Good++;
                }
            }
        }
    }
    histories.Histories = histories.Histories.OrderBy(i => i.Date).ToList();
    return histories;
  }
 }
}

using Microsoft.AspNet.Identity.EntityFramework;
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Authorization;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Mvc;
using Mocoding.Audit.Models;
using Mocoding.Audit.Services;
using Mocoding.Common.Auth.Models.DTO;
using System.Collections.Generic;
```

```
using Mocoding.Common.Web;
using Newtonsoft.Json;
using SmartParking.ViewModels;
using SmartParking.Models;

namespace SmartParking.Api
{
    [Route("api/accounts")]
    public class AccountsController : ApiController
    {
        [ActionContext]
        public ActionContext _actionContext { get; set; }

        readonly UserManager<SystemUser> _userManager;
        readonly SignInManager<SystemUser> _signInManager;
        readonly IAuditService _audit;

        public AccountsController(UserManager<SystemUser> userManager,
SignInManager<SystemUser> signInManager, IAuditService audit)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _audit = audit;
        }

        [Route("login")]
        [HttpPost]
        public async Task<IActionResult> Login([FromBody] LoginDto model)
        {
            if (!_actionContext.ModelState.IsValid)
```

```
            return
HttpBadRequest(_actionContext.ModelState.ValidationState.ToString());

        var user = await _userManager.FindByNameAsync(model.Name);
        if (user == null)
            return HttpBadRequest("Can't find such user");

        if (await _userManager.IsLockedOutAsync(user))
            return HttpBadRequest("User was deleted!");

        var valid = await _userManager.CheckPasswordAsync(user, model.Password);
        if (valid)
        {
            await _signInManager.SignInAsync(user, false);
            return await OnLogin(user);
        }
        return Unauthorized();
    }

    [Route("registration")]
    [HttpPost]
    public async Task<IActionResult> Registration([FromBody] RegistrationDto
model)
    {
        if (!_actionContext.ModelState.IsValid)
            return
HttpBadRequest(_actionContext.ModelState.ValidationState.ToString());

        var user = new SystemUser() { UserName = model.Name };
```

```
            var result = await _userManager.CreateAsync(user, model.Password);

            if (result.Succeeded)
            {
                await _userManager.AddToRoleAsync(user, Constants.Roles.User);
                await _signInManager.SignInAsync(user, false);
                return await OnLogin(user);
            }

            return HttpBadRequest(result.Errors.FirstOrDefault()?.Description);
        }

        [Route("logoff")]
        [HttpGet]
        [Authorize]
        public async Task LogOff()
        {
            var role =
_actionContext.HttpContext.User.FindFirst(ClaimTypes.Role)?.Value;
            await _signInManager.SignOutAsync();
        }


        [HttpGet]
        public IEnumerable<string> GetUserSuggestion()
        {
            var users = _userManager.Users.Skip(1).Where(u => u.LockoutEnd ==
null).Select(_ => _.UserName).ToArray();
            return users;
        }
```

```
        [HttpGet("user.js")]
        public string GetAngularService()
        {
            var userInfo = !_actionContext.HttpContext.User.IsSignedIn() ? "null" :
JsonConvert.SerializeObject(new UserResultDto()
            {
                Role = _actionContext.HttpContext.User.FindFirst(ClaimTypes.Role)?.Value,
                Name =
_actionContext.HttpContext.User.FindFirst(ClaimTypes.Name)?.Value
            });

            var script = $"app.service('userContainer', function Service() {{ this.user =
{userInfo}; }});";

            _actionContext.HttpContext.Response.ContentType = "test/javascript";

            return script;
        }

        private async Task<IActionResult> OnLogin(SystemUser user)
        {
            var roles = await _userManager.GetRolesAsync(user);
            var role = roles.Count > 0 ? roles[0] : "user";
            return CreateResult(new { role = role, name = user.UserName });
        }
    }
}

'use strict';
```

```
angular.module('app.parking').controller('parkingManagementCtrl',
    function ($scope, $state, parkingApi) {
        var markers = [];

        var getMarkers = function () {
            parkingApi.getAdminParkings().then(
                function (data) {
                    for (var i = 0; i < data.length; i++) {
                        addMarkerWithTimeout(data[i], i * 200);
                    }
                },
                function (error) {
                    console.log("Get markers", error);
                }
            );
        };

        var addMarkerWithTimeout = function(position, timeout) {
            window.setTimeout(function() {
                var marker = new google.maps.Marker({
                    position: {lat: position.latitude, lng: position.longitude},
                    map: map,
                    animation: google.maps.Animation.DROP
                });
                var infowindow = new google.maps.InfoWindow({
                    content: position.title
                });
                marker.addListener('click', function() {
                    infowindow.open(map, marker);
```

```
            });
            markers.push(marker);
        }, timeout);
    }



    $scope.init = function () {
        fixMapHeight();
        getMarkers();
    }

    $scope.init();
});
```

Додаток Б

Слайди перезентації

Атестаційна робота
магістра

# Дослідження методів кластеризації даних про використання розумних паркінгів з метою прогнозування їх завантаження

Виконав: ст.гр. ІПЗм-17-1 Хомишин Д. О.
Керівник: доц. каф. ПІ Лещинський В. О.

1

# Актуальність

- Росповсюдження Smart Technologies.
- Збільшення кількості розумних паркінгів (наприклад, прокат велосипедів, електросамокатів, каршеринг тощо).
- Оптимізація та автоматизація роботи паркінгів зменшить витрати.

2

# Мета роботи

- Дослідити та порівняти методи кластеризації даних.
- Визначити, який з методів кластеризації вирішує поставлену задачу найефективнішим шляхом.
- Реалізувати програмну систему.

3

# Постановка задачі

- Провести аналіз аналогів. Їх плюси та мінуси.
- Провести аналіз предметної області: кількість можливих користувачів та їх потреби.
- Дослідити та порівняти методи кластеризації. Обрати найефективніший.
- Спроектувати архітектуру програмного продукту.
- Розробка програмного продукту.

4

# Аналіз аналогів: fastprk

- Використовує методики глубокого навчання для прогнозування завантаження
- Інтеграція тільки з власним спорядженням
- Працює тільки з паркінгами для автомобілів

5

# Аналіз предметної області

- Популярність сервісів спільного споживання у світі (короткострокова оренда транспорту) продовжує збільшуватися
- Кількість автомобілів у світі збільшується кожного року приблизно на 4-5 відсотків
- 55 мільйонів автомобілів у Німеччині на 82 мільйони населення
- 300 мільйонів автомобілів у Китаї на 1.3 мільярда населення

6

# Методи кластеризації

- DBSCAN - є алгоритмом кластеризації заснованим на щільності: для заданої множини точок у деякому просторі він відносить в одну групу точки, які розташовані найбільш щільно (точки з багатьма сусідами) та розмічає точки, які лежать в областях з невеликою щільністю (чиї сусіди розташовані занадто далеко) як викиди.

7

# Методи кластеризації

- K-means - розбиває безліч елементів векторного простору на заздалегідь відоме число кластерів k. Основна ідея полягає в тому, що на кожній ітерації переобчислюють центр мас для кожного кластера, отриманого на попередньому кроці, потім вектори розбиваються на кластери знову відповідно до того, який з нових центрів виявився ближчим за обраною метрикою. Алгоритм завершується, коли на якийсь ітерації не відбувається зміни внутрікластерної відстані.

8

# Методи кластеризації

- Self Organizing Maps - являє собою один з варіантів кластеризації багатомірних векторів. Важливою відмінністю алгоритму SOM від k-means є те, що в ній всі нейрони упорядковані в двомірну сітку. При цьому в навчанні модифікується не тільки нейрон-переможець, але і його сусіди.

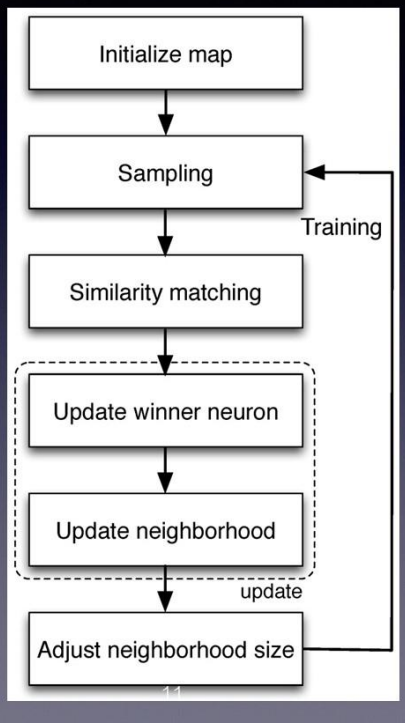
9

# Порівняння

- Виявилося, що k-means та DBSCAN мають проблеми з класифікацією і не можуть відокремити відхилення від інших кластерів, тоді як SOM-схема працює задовільно, досягаючи найкращої класифікації при тестуванні на синтезованих подіях паркування і надійно виявляє всі викиди, які застосовуються до реальних даних.
- Усі алгоритми не є самонавчаючими.

10

# Самонавчаючий SOM




11

# Архітектура та технології

- Domain Driven Design
- MVC
- Dependency Injection
- ACP.NET Core
- AngularJS

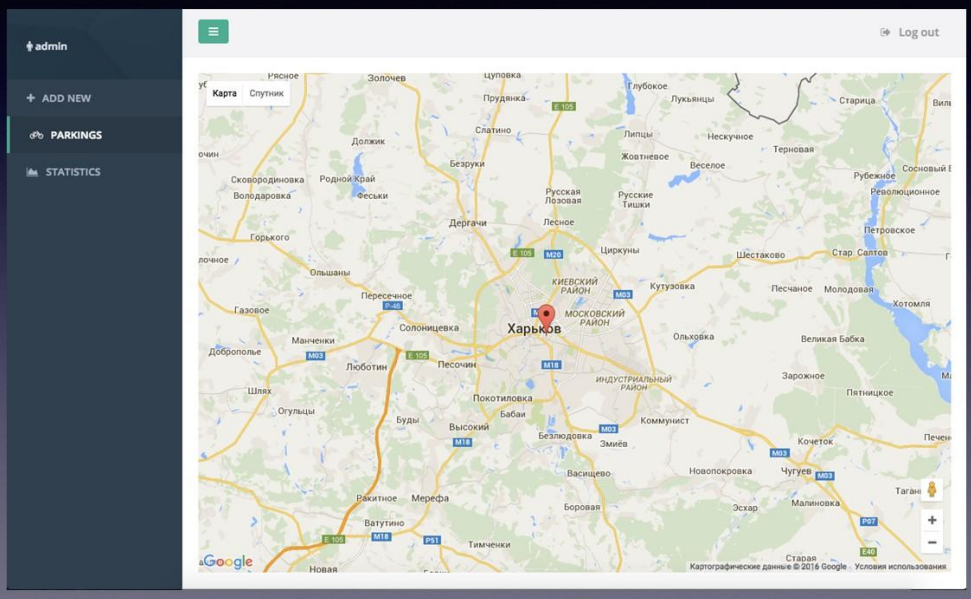
12

# Функціонал

- Реєстрація нових користувачів
- Дві ролі (адмін та користувач)
- Управління паркінгами, аналіз даних, отримання статистики для адміна
- Оренда парковочного місця для користувачів

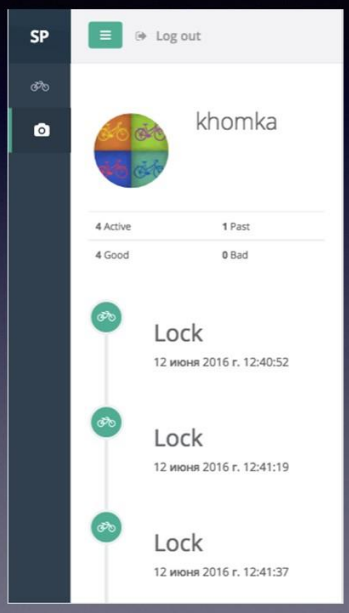
13

# Інтерфейс програми




14

# Інтерфейс програми




15

# Плани на майбутнє

- Продовжити дослідження SOM based алгоритму кластеризації для його покращення
- Оптимізація  швидкості опрацювання великих даних
- Інтеграція з апаратним забезпеченням

16

# Висновки

В ході виконання атестаційної роботи:

- проведено підготовку до створення програмного продукту;
- проведений аналіз ринку та конкурентів;
- досліджені та порівняні методи кластеризації;
- проектування програмної системи;
- створено UML-діаграми;
- розроблено програмний продукт;
- проведено тестування програмного продукту.

# Додаток В

## Подані тези

*Хомишин Д.О.*

*Харківський національний університет радіоелектроніки, м. Харків*

*Кафедра програмної інженерії*

### ДОСЛІДЖЕННЯ МЕТОДІВ КЛАСТЕРИЗАЦІЇ ДАНИХ ПРО ВИКОРИСТАННЯ РОЗУМНИХ ПАРКІНГІВ З МЕТОЮ ПРОГНОЗУВАННЯ ЇХ ЗАВАНТАЖЕННЯ

Дослідження даних про зайнятість паркінгів має велику актуальність, зважаючи на те, що завдяки цьому з'являється можливість отримувати актуальні данні пересування населення містом. Моя мета полягає в тому, щоб зробити ці дані корисними для таких користувачів як паркувальники, комерційні бізнеси, міська влада тощо.

Використання автоматизованих приладів для моніторингу на вуличних паркінгах стало популярним у кількох містах світу. У існуючих середовищах невеликі пристрої для зчитування зазвичай розміщуються в кожному місці для паркування. Це необхідно для моніторингу великих міських територій. Гарними прикладами серед багатьох інших є Лос-Анджелес, Сан-Франциско і Барселона. Перший шар цих складних систем складається з датчиків паркування на вулиці: дрібними бездротовими пристроями, що використовуються для моніторингу наявності транспортних засобів. Кожен датчик періодично прокидається, щоб перевірити стан зайнятості призначеної стоянки. Отримані дані надсилаються до серверу для подальшої обробки, управління віддаленим паркінгом та візуалізацією.

Основна мета цих систем полягає в підвищенні ефективності роботи громадської паркінгу, що досягається за рахунок збору інформації про зайнятість стоянки. Зібрані дані аналізуються та надаються відділу управління паркінгами міста через відповідні інформаційні панелі. Крім того, наявність інформації про паркування в режимі реального часу також дає змогу надавати нові послуги, забезпечуючи покращення роботи користувачам. Як приклад,

системи паркування та інформації допомагають водіям більш ефективно знаходити місця для паркування, тим самим вирішуючи проблему довгого пошуку вільного місця.

Для досягнення поставленої мети застосуємо оригінальну техніку кластеризації за допомогою самоорганізуючих карт (SOM), які є самокерованими нейронними мережами, що здатні вивчати прототипи в багатовимірних векторних просторах.

SOM є нейро-обчислювальним алгоритмом, який перетворює високовимірні дані в одно- або двовимірний простір через нелінійний, конкурентний і без нагляду процес навчання. SOM відрізняється від інших штучних нейронних мереж, оскільки використовує функцію сусідства для збереження топологічних властивостей вхідного простору. Вона вивчається за допомогою вхідних прикладів, а вхідний простір відображається в двовимірну решітку нейронів, зберігаючи властивість, що подібні вхідній структурі відображені сусідніми нейронами на карті.

Розглянемо одно- та двовимірні карти, які відповідно складаються з послідовностей M × 1 нейронів і решітки $\ell$ = M × M нейронів, з M> 1. Ці карти вибірково налаштовані на вхідні структури через безкваліфікований (також називається конкурентним) процес навчання. Коли навчання прогресує, ваги нейронів мають тенденцію ставати впорядкованими по відношенню один до одного таким чином, що над решіткою створюється значна система координат для різних функцій введення. Іншими словами, SOM створює топографічну карту вхідного простору даних, де просторові місця розташування або координати нейронів в решітці відповідають певній області або власній статистичній функції вхідних даних. Примітно, що це досягається без необхідності будь-яких попередніх знань щодо розподілу вхідних даних.

За допомогою $\mathcal{X} \subset \mathbb{R}K$ вказуємо множину ознак (вхід), а $xi \in \mathcal{X}$ - вхідний вектор ознак, пов'язаний з сенсором $i = 1, \ldots, N$, де $xi = [xi1, xi2, \cdots, xiK]T$. З $N$ ми маємо на увазі число датчиків, і $|\mathcal{X}| = N$. Нехай $\mathcal{L}$ - решітка. Кожен нейрон з'єднаний з кожним компонентом вхідного вектора, як показано на рисунку 1.

Зв'язки між вхідним вектором і нейронами зважені, так що *j*-й нейрон пов'язаний з синаптичним ваговим вектором wj ∈ RK, де $wj = [wi1, wi2, \cdots, wiK]T$.

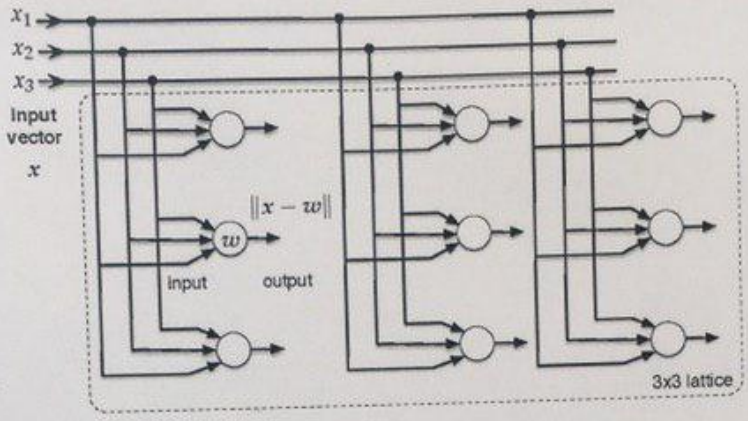



Рисунок 1 - Схема процесу навчання SOM

На рисунку 2 зображено схему роботи алгоритму SOM в загальному вигляді.

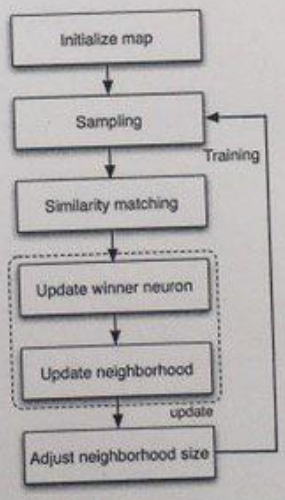



Рисунок 2 – Алгоритм SOM в загальному вигляді