ДОДАТОК А

Технічні характеристики використаних пристроїв та датчиків

Таблиця А.1 – Технічні характеристики ESP32-cam

Назва	Значення
Розмір	27*40.5*4.5（±0.2）мм^3.
Flash пам’ять	32 мбіт.
Ram пам’ять	Внутрішня пам'ять 520KB + зовнішня 8MB PSRAM.
Wi-Fi	802.11 b/g/n/e/i.
Роз’ємів вводів/виводів	9 штук.
Швидкість послідовного порту	115200 біт/с.
Діапазон живлення	4.75-5.25В

Таблиця А.2 – Технічні характеристики ESP8266 WEMOS D1 R2

Назва	Значення
Розмір	32*26*6（±0.2）мм^3.
Flash пам’ять	32 мбіт.
Ram пам’ять	Внутрішня пам'ять 520KB + зовнішня 8MB PSRAM.
Wi-Fi	802.11 b/g/n/e/i.
Роз’ємів вводів/виводів	11 штук.

Продовження таблиці А.2

Назва	Значення
Швидкість послідовного порту	115200 біт/с.
Діапазон живлення	3.7-12 В

Таблиця А.3 – Технічні характеристики HC-SR501

Назва	Значення
Максимальний струм споживання	< 2 мА.
Затримка	Від 1 секунди до 3 хвилин.
Відстань виявлення	3 - 7 метрів.
Кут виявлення	120 градусів

А.4 – Технічні характеристики SEN18

Назва	Значення
Напруга живлення	3 – 5 В.
Максимальний струм споживання	< 20 мА.
Ділянка виявлення	40 × 16 мм.

А.5 – Технічні характеристики SG90

Назва	Значення
Напруга живлення	4,8 – 6,0 В.
Крутний момент	1,8 кг·см (при 4,8 В).
Швидкість обертання	0,1 с / 60° (при 4,8 В)
Кут повороту	180°

ДОДАТОК Б

Реалізація мапера та адаптера для сутності FlowDefinition

```
    public class FlowDefinitionMapperProfile : Profile
    {
        public FlowDefinitionMapperProfile()
        {
            CreateMap<FlowDefinition, FlowDefinitionDto>()
                .ReverseMap();

            CreateMap<FlowStep, FlowStepDto>()
                .ForMember(dest => dest.GuidId, opt => opt.MapFrom(src
=> src.GuidId))
                .ForMember(dest     =>     dest.Parameters,     opt     =>
opt.MapFrom(src => src.Parameters))
                .ReverseMap()
                .ForMember(dest => dest.GuidId, opt => opt.MapFrom(src
=> src.GuidId))
                .ForMember(dest     =>     dest.Parameters,     opt     =>
opt.MapFrom(src => src.Parameters));

            CreateMap<FlowStepTransition, FlowStepTransitionDto>()
                .ForMember(dest    =>    dest.FromStepGuid,    opt    =>
opt.MapFrom(src => src.FromStep.GuidId))
                .ForMember(dest     =>     dest.ToStepGuid,     opt     =>
opt.MapFrom(src => src.ToStep.GuidId))
                .ReverseMap()
                .ForMember(dest => dest.FromStep, opt => opt.Ignore())
                .ForMember(dest => dest.ToStep, opt => opt.Ignore());

            CreateMap<FlowStepParameter,
FlowStepParameterDto>().ReverseMap();
        }
    }
    public           class           FlowDefinitionDtoAdapter           :
IFlowDefinitionDtoAdapter<FlowDefinitionDto, FlowDefinition>
    {
        private readonly IMapper _mapper;
        public FlowDefinitionDtoAdapter(IMapper mapper)
        {
            _mapper = mapper;
        }

        public                                      Task<FlowDefinitionDto>
GetFlowDefinitionDtoAsync(FlowDefinition flowDefinition)
        {
            var                          mappedDto                          =
_mapper.Map<FlowDefinitionDto>(flowDefinition);
            return Task.FromResult(mappedDto);
        }

        public                                          Task<FlowDefinition>
GetFlowDefinitionAsync(FlowDefinitionDto flowDefinitionDto)
```

```
        {
            var                    flowDefinition                    =
_mapper.Map<FlowDefinition>(flowDefinitionDto);

            if (flowDefinition.Steps != null)
            {
                foreach (var step in flowDefinition.Steps)
                {
                    step.FlowDefinition = flowDefinition;
                    step.OutgoingTransitions.Clear();
                }

                foreach (var stepDto in flowDefinitionDto.Steps)
                {
                    var               correspondingStep               =
flowDefinition.Steps.FirstOrDefault(s => s.GuidId == stepDto.GuidId);
                    if      (correspondingStep      !=      null      &&
stepDto.OutgoingTransitions != null)
                    {
                        foreach        (var        transitionDto        in
stepDto.OutgoingTransitions)
                        {
                            var               targetStep               =
flowDefinition.Steps.FirstOrDefault(s        =>        s.GuidId        ==
transitionDto.ToStepGuid);
                            if (targetStep != null)
                            {
                                var transition = new FlowStepTransition
                                {
                                    Condition = transitionDto.Condition,
                                    FromStep = correspondingStep,
                                    ToStep = targetStep
                                };

correspondingStep.OutgoingTransitions.Add(transition);
                            }
                        }
                    }
                }
            }

            return Task.FromResult(flowDefinition);
        }

        public                        Task<ICollection<FlowDefinitionDto>>
GetFlowDefinitionDtosAsync(ICollection<FlowDefinition> flowDefinitions)
        {
            ICollection<FlowDefinitionDto>      mappedDtos      =      new
List<FlowDefinitionDto>();
            foreach (var flowDefinition in flowDefinitions)
            {
                var                       mappedDto                       =
_mapper.Map<FlowDefinitionDto>(flowDefinition);
                mappedDtos.Add(mappedDto);
            }
            return Task.FromResult(mappedDtos);
```

```
        }
    }
```

ДОДАТОК В

Реалізація перетворення текста у SHA256 та створення токенів

```
    public static class StringHasher
    {
        public static string HashStringSHA256(string input)
        {
            using HashAlgorithm algorithm = SHA256.Create();

            byte[]                  encryptedByteArray                  =
algorithm.ComputeHash(Encoding.ASCII.GetBytes(input));

            return Convert.ToBase64String(encryptedByteArray);
        }
    }
    public class JwtTokenHandler : IJwtTokenHandler
    {
        private readonly JwtConfiguration _jwtConfiguration;

        public                JwtTokenHandler(IOptions<JwtConfiguration>
jwtConfigurations)
        {
            _jwtConfiguration  =  jwtConfigurations.Value  ??  throw  new
ArgumentNullException("There are no jwt configurations on server");
        }

        public JwtSecurityToken CreateAccessToken(List<Claim> claims)
        {
            SymmetricSecurityKey           key           =           new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(_jwtConfiguration.AccessSec
retCode));

            var token = new JwtSecurityToken(
                claims: claims,
                signingCredentials:      new      SigningCredentials(key,
SecurityAlgorithms.HmacSha256Signature),
                issuer: _jwtConfiguration.Issuer,
                audience: _jwtConfiguration.Audience,
                expires:
DateTime.Now.AddHours(_jwtConfiguration.AccessLifetime)
            );

            return token;
        }

        public JwtSecurityToken CreateRefreshToken(List<Claim> claims)
        {
            SymmetricSecurityKey           key           =           new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(_jwtConfiguration.RefreshSe
cretCode));

            var token = new JwtSecurityToken(
                claims: claims,
```

```
                signingCredentials:      new      SigningCredentials(key,
SecurityAlgorithms.HmacSha256Signature),
                issuer: _jwtConfiguration.Issuer,
                audience: _jwtConfiguration.Audience,
                expires:
DateTime.Now.AddHours(_jwtConfiguration.RefreshLifetime)
            );

            return token;
        }

        public   ClaimsPrincipal   ValidateToken(string   strToken,   bool
isRefreshToken = false)
        {
            JwtSecurityTokenHandler         handler         =         new
JwtSecurityTokenHandler();

            string        secret        =        isRefreshToken        ?
_jwtConfiguration.RefreshSecretCode                                     :
_jwtConfiguration.AccessSecretCode;

            var validationParameters = new TokenValidationParameters()
            {
                IssuerSigningKey                    =                   new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(secret)),
                ValidateAudience = false,
                ValidateIssuer = false,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
            };

            SecurityToken token;
            ClaimsPrincipal    cp    =    handler.ValidateToken(strToken,
validationParameters, out token);

            return cp;
        }
    }
```

ДОДАТОК Г

Реалізація шаблону Unit of Work та базового репозиторію

```
    internal class UnitOfWork : IUnitOfWork
    {
        private readonly ServerContext _context;
        private readonly Dictionary<Type, object> _repositories;
        private   static   readonly   ConcurrentDictionary<Type,   Type>
RepositoryTypeCache = new();
        public UnitOfWork(ServerContext context)
        {
            _context = context;
            _repositories = new Dictionary<Type, object>();
        }
        public   Task<IRepository<T>>   GetRepository<T>()   where   T   :
BaseEntity
        {
            var entityType = typeof(T);
            if (_repositories.TryGetValue(entityType, out var repo))
            {
                return Task.FromResult((IRepository<T>)repo);
            }

            var                        repositoryType                        =
RepositoryTypeCache.GetOrAdd(entityType, type =>
            {
                var repoType = typeof(UnitOfWork).Assembly
                    .GetTypes()
                    .FirstOrDefault(t                                        =>
typeof(IRepository<T>).IsAssignableFrom(t));
                return repoType ?? typeof(BaseRepository<T>);
            });

            var                    repositoryInstance                    =
Activator.CreateInstance(repositoryType, _context)
                                    ??             throw             new
ArgumentException($"Unable    to    resolve    repository    with    type
{entityType.Name}");
            _repositories[entityType] = repositoryInstance;
            return Task.FromResult((IRepository<T>)repositoryInstance);
        }
        public Task CommitAsync()
        {
            return _context.SaveChangesAsync();
        }
    }
    public class BaseRepository<T> : IRepository<T> where T : BaseEntity
    {
        protected readonly DbContext _context;
        protected readonly DbSet<T> _dbSet;

        public BaseRepository(DbContext context)
        {
```

```
            _context = context;
            _dbSet = _context.Set<T>();
        }

        public virtual async Task<T> CreateAsync(T entity,
CancellationToken cancellationToken = default)
        {
            var result = await _dbSet.AddAsync(entity,
cancellationToken);
            await _context.SaveChangesAsync(cancellationToken);
            return result.Entity;
        }

        public virtual async Task<T?> ReadEntityByIdAsync(int id,
CancellationToken cancellationToken = default)
        {
            return await _dbSet.FindAsync(new object[] { id },
cancellationToken);
        }

        public virtual async Task<IEnumerable<T>>
ReadEntitiesByPredicate(Expression<Func<T, bool>> predicate,
IEnumerable<KeyValuePair<Expression<Func<T, object>>, bool>>? orderBy =
null, CancellationToken cancellationToken = default)
        {
            IQueryable<T> query = _dbSet.Where(predicate);

            if (orderBy is not null && orderBy.Any())
            {
                foreach (var order in orderBy)
                {
                    query = order.Value ? query.OrderBy(order.Key) :
query.OrderByDescending(order.Key);
                }
            }

            return await query.ToListAsync(cancellationToken);
        }

        public virtual async Task<T> UpdateAsync(T entity,
CancellationToken cancellationToken = default)
        {
            var existingEntity = await ReadEntityByIdAsync(entity.Id,
cancellationToken);

            if (existingEntity is null) return entity;

            var oldEntityEntry = _context.Entry(existingEntity)!;
            oldEntityEntry.CurrentValues.SetValues(entity);

            await _context.SaveChangesAsync(cancellationToken);
            return entity;
        }

        public virtual async Task<bool> DeleteAsync(int id,
CancellationToken cancellationToken = default)
```

```
        {
            var entity = await _dbSet.FindAsync(new object[] { id },
cancellationToken);
            if (entity == null)
            {
                return false;
            }

            _dbSet.Remove(entity);
            await _context.SaveChangesAsync(cancellationToken);
            return true;
        }
```

ДОДАТОК Д
Демонстраційні матеріали

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк.	Примітки
			Текстові документи		
А4	1	ГЮІК 5505900.051 ПЗ	Пояснювальна записка		62 с.
			Інші документи		
	2		Демонстраційний матеріал		12 с.

					ГЮІК 5505900.051 ВД
Змін.	Арк	Номер докум.	Підпис	Дата	
Розроб.		Шарлай А.Ю.			Розроблення системи автоматизації процесу збирання та обробки даних з пристроїв IoT на інтелектуальному виробництві Відомість кваліфікаційної роботи
Перевір.		Новоселов С.П.			
Н.контр.		Демська Н. П.			
Затв.		Невлюдов			

Літера	Арку	Аркуш
Н	1	1

Кафедра КІТАР ХНУРЕ