ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОЇ СИСТЕМИ

```
using System.Collections.Generic;
using TimeZilla.BL.Interfaces;
using TimeZilla.DataLayer.Interfaces;
using TimeZilla.DataLayer.Models;
using Task = TimeZilla.BL.Models.Task;

namespace TimeZilla.BL.Managers
{
   class TasksManager : IManager<Task>
   {
      private readonly IRepository<TaskDl> _tasksRepository;

      public TasksManager(IRepository<TaskDl> tasksRepository)
      {
         _tasksRepository = tasksRepository;
      }


      public Task GetById(int id)
      {
         return _tasksRepository.GetById(id).ConvertFromTaskDlToTask();
      }

      public void Update(Task task)
      {
         _tasksRepository.Update(task.ConvertFromTaskToTaskDl());
      }

      public void Delete(int id)
      {
         _tasksRepository.Delete(id);
      }

      public void Delete(Task task)
      {
         _tasksRepository.Delete(task.ConvertFromTaskToTaskDl());
      }

      public void Insert(Task task)
      {
         _tasksRepository.Insert(task.ConvertFromTaskToTaskDl());
      }

      public IEnumerable<Task> Get()
      {
         return _tasksRepository.Get().ConvertEnumerableTaskDlToEnumerableTask();
      }
   }
}

using System;
using System.IO;
using Mono.Data.Sqlite;
using TimeZilla.DataLayer.Models;

namespace TimeZilla.DataLayer.Database
{
   internal class DbInitializer : DropOrUpdateOnModelChanged
   {
      static DbInitializer()
```

```
        {
            String pathToDatabase = GlobalEnvironment.PathToDatabase;

            bool exists = File.Exists(pathToDatabase);

            if (!exists)
            {
                SqliteConnection.CreateFile(pathToDatabase);
            }

            UpdateTables = true;

            #region Tables

            Activator.CreateInstance<GoalDl>();

            Activator.CreateInstance<CategoryDl>();

            Activator.CreateInstance<TaskDl>();

            #endregion

        }
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using Mono.Data.Sqlite;
using TimeZilla.DataLayer.Attributes;
using TimeZilla.DataLayer.Enums;
using TimeZilla.DataLayer.ExtensionMethods;

namespace TimeZilla.DataLayer.Database
{
    internal class DbSet<TEntity> where TEntity : class,new()
    {
        internal bool Insert(TEntity model)
        {
            using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
            {
                try
                {
                    sqliteConnection.Open();
                    Dictionary<string, string> parametersDictionary;
                    using (var sqliteCommand = new SqliteCommand(GenerateQuery(CrudOptions.Insert, out
parametersDictionary),sqliteConnection))
                    {
                        foreach (var modelInfo in model.GetType().GetProperties())
                        {
                            if (modelInfo.Name == "Id" || modelInfo.CheckNavigationPropertyAttribute())
                            {
                                //skips insert for id and navigation property
                            }
                            else
                            {
                                sqliteCommand.Parameters.AddWithValue(parametersDictionary[modelInfo.Name],
                                    GetPropValue(model, modelInfo.Name));
                            }
                        }
                        if (sqliteCommand.ExecuteNonQuery() > 0)
                        {
                            return true;
                        }
```

```
                return false;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
    return true;
}

internal TEntity GetById(Int32 id)
{
    var tEntity = new TEntity();
    using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
    {
        try
        {
            sqliteConnection.Open();
            Dictionary<string, string> parametersDictionary;
            using (var sqliteCommand = new SqliteCommand(GenerateQuery(CrudOptions.GetById, out
parametersDictionary), sqliteConnection))
            {
                sqliteCommand.Parameters.AddWithValue(parametersDictionary["Id"], id);
                var reader = sqliteCommand.ExecuteReader();
                if (reader.HasRows)
                {
                    while (reader.Read())
                    {
                        foreach (var propertyInfo in tEntity.GetType().GetProperties())
                        {
                            propertyInfo.SetValue(tEntity, reader[propertyInfo.Name]);
                        }
                    }
                }
                return tEntity;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
    return null;
}

internal IEnumerable<TEntity> GetAll()
{
    var list = new List<TEntity>();
    using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
    {
        try
        {
            sqliteConnection.Open();
            Dictionary<string, string> parametersDictionary;
            using (var sqliteCommand = new SqliteCommand(GenerateQuery(CrudOptions.GetAll, out
parametersDictionary), sqliteConnection))
            {
                var reader = sqliteCommand.ExecuteReader();
                if (reader.HasRows)
                {
                    while (reader.Read())
                    {
                        var tEntity = new TEntity();
                        foreach (var propertyInfo in tEntity.GetType().GetProperties())
```

```
                    {
                        propertyInfo.SetValue(tEntity, reader[propertyInfo.Name]);
                    }
                    list.Add(tEntity);
                }
            }
            return list;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
return list;
}

internal bool Update(TEntity model)
{
    using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
    {
        try
        {
            sqliteConnection.Open();
            Dictionary<string, string> parametersDictionary;
            using (var sqliteCommand = new SqliteCommand(GenerateQuery(CrudOptions.Update, out
parametersDictionary), sqliteConnection))
            {
                foreach (var modelInfo in model.GetType().GetProperties())
                {
                    if (!modelInfo.CheckNavigationPropertyAttribute())
                    {
                        sqliteCommand.Parameters.AddWithValue(parametersDictionary[modelInfo.Name],
                            GetPropValue(model, modelInfo.Name));
                    }
                }
                if (sqliteCommand.ExecuteNonQuery() > 0)
                {
                    return true;
                }
                return false;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
    return true;

}

internal bool Delete(TEntity model)
{
    Int32 id = Convert.ToInt32(GetPropValue(model, "Id"));
    if (Delete(id))
    {
        return true;

    }
    return false;
}

internal bool Delete(Int32 id)
{
```

```
        using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
        {
            try
            {
                sqliteConnection.Open();
                Dictionary<string, string> parametersDictionary;
                using (var sqliteCommand = new SqliteCommand(GenerateQuery(CrudOptions.Delete, out
parametersDictionary), sqliteConnection))
                {
                    sqliteCommand.Parameters.AddWithValue(parametersDictionary["Id"],id);
                    if (sqliteCommand.ExecuteNonQuery() > 0)
                    {
                        return true;
                    }
                    return false;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
        return true;
    }

    private String GenerateQuery(CrudOptions options, out Dictionary<String, String> parameterDictionary)
    {
        var forEntity = new TEntity();
        parameterDictionary = new Dictionary<String, string>();
        var entityProperties = forEntity.GetType().GetProperties();
        String tableName;
        try
        {
            var attribute = (TableName)Attribute.GetCustomAttribute(forEntity.GetType(), typeof(TableName));
            tableName = attribute.Name;
        }
        catch (NullReferenceException)
        {
            tableName = forEntity.GetType().Name;
        }
        var command = new StringBuilder();
        switch (options)
        {
            case CrudOptions.Delete:
                command.Append("DELETE FROM [" + tableName + "] WHERE Id = @Id");
                parameterDictionary.Add("Id", "@Id");
                return command.ToString();
            case CrudOptions.GetById:
                command.Append("SELECT * FROM [" + tableName + "] WHERE Id = @Id");
                parameterDictionary.Add("Id", "@Id");
                return command.ToString();
            case CrudOptions.GetAll:
                command.Append("SELECT * FROM [" + tableName + "]");
                return command.ToString();
            case CrudOptions.Insert:
                command.Append("INSERT INTO [" + tableName + "] ");
                command.Append("(");
                foreach (var valueName in entityProperties)
                {
                    if (valueName.Name == "Id" || valueName.CheckNavigationPropertyAttribute())
                    {
                        //skips insert for id and navigation property
                    }
                    else
                    {
```

```
                    command.Append("[" + valueName.Name + "],");
                }
            }
            command.TrimLastChar();
            command.Append(") VALUES (");
            foreach (var valueValue in entityProperties)
            {
                String addValue = "@" + valueValue.Name;
                if (valueValue.Name == "Id" || valueValue.CheckNavigationPropertyAttribute())
                {
                    //skips adding value for id and navigation property
                }
                else
                {
                    parameterDictionary.Add(valueValue.Name, addValue);
                    command.Append(addValue + ",");
                }
            }
            command.TrimLastChar();
            return command.Append(")").ToString();
        case CrudOptions.Update:
            command.Append("UPDATE [" + tableName + "] SET ");
            foreach (var valueName in entityProperties)
            {
                if (!valueName.CheckNavigationPropertyAttribute()&& valueName.Name!="Id")
                {
                    String parameter = "@" + valueName.Name;
                    parameterDictionary.Add(valueName.Name, parameter);
                    command.Append(valueName.Name + "=" + parameter + ",");
                }

            }
            command.TrimLastChar();
            command.Append(" WHERE Id = @Id");
            parameterDictionary.Add("Id","@Id");
            return command.ToString();
        default:
            throw new Exception("Error happened");
    }
}

private static object GetPropValue(object source, string propName)
{
    return source.GetType().GetProperty(propName).GetValue(source, null);
}
}
} using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using Mono.Data.Sqlite;
using TimeZilla.DataLayer.Attributes;
using TimeZilla.DataLayer.ExtensionMethods;

namespace TimeZilla.DataLayer.Database
{
    internal class DropOrUpdateOnModelChanged
    {
        protected static bool UpdateTables = false;
        private static readonly List<String> EntityNames = new List<string>();
        private static readonly String Path;

        static DropOrUpdateOnModelChanged()
        {
            var documents = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
```

```
    var directoryname = System.IO.Path.Combine(documents, "Objects");
    Directory.CreateDirectory(directoryname);
    Path = directoryname;
    WriteToFile(Entities.EntitiesList);
    Cleanup(EntityNames);
}

private static void Cleanup(List<String> list)
{
    var tablesToDelete = new List<String>();
    var files = Directory.GetFiles(Path, "*.txt");
    foreach (var file in files)
    {
        var fileName = file.Replace(Path, "").Replace(".txt", "");
        if (!list.Contains(fileName))
        {
            File.Delete(file);
            tablesToDelete.Add(fileName);
        }
    }
    using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
    {
        try
        {
            sqliteConnection.Open();
            using (var sqliteCommand = new SqliteCommand("",sqliteConnection))
            {
                var sb = new StringBuilder("DROP TABLE ");
                foreach (var table in tablesToDelete)
                {
                    sb.Append(table + "; DROP TABLE");
                }
                sb.Remove(sb.Length - 11, 11);
                sqliteCommand.CommandText = sb.ToString();
                sqliteCommand.ExecuteNonQuery();
            }

        }
        catch (Exception exception)
        {
            Logger.Log(exception.Message);
        }
    }
}

private static String ParseObject(Type type)
{
    String tableName;
    try
    {
        var attribute = (TableName)Attribute.GetCustomAttribute(type, typeof(TableName));
        tableName = attribute.Name;
    }
    catch (NullReferenceException)
    {
        tableName = type.Name;
    }
    String comparerString = null;
    comparerString += tableName + "\r\n";
    foreach (var propertyInfo in type.GetProperties())
    {
        if (propertyInfo.CheckNavigationPropertyAttribute())
        {

        }
```

```
            else
            {
                comparerString += ("Type=" + propertyInfo.PropertyType.Name + ",Name=" + propertyInfo.Name +
"\r\n");
            }
        }
        return comparerString;
    }

    private static void WriteToFile(IEnumerable<object> list)
    {
        foreach (var obj in list)
        {
            String tableName;
            var objectType = obj.GetType();
            try
            {
                var attribute = (TableName)Attribute.GetCustomAttribute(objectType, typeof(TableName));
                tableName = attribute.Name;
            }
            catch (NullReferenceException)
            {
                tableName = objectType.Name;
            }
            EntityNames.Add(tableName);
            if (File.Exists(Path + @"\" + tableName + ".txt"))
            {
                bool state = true; //determines if data should be rewritten to file
                using (var sr = new StreamReader(Path + @"\" + tableName + ".txt"))
                {
                    String oldTableName = sr.ReadLine();
                    var data = oldTableName + "\r\n" + sr.ReadToEnd();
                    if (data == ParseObject(objectType))
                    {
                        state = false;
                    }
                    if (UpdateTables)
                    {
                        UpdateTable(obj, oldTableName);
                    }
                    else
                    {
                        DeleteTable(oldTableName);
                        var typeArguments = Type.GetType(objectType.Namespace + "." + objectType.Name);
                        Type template = typeof(Table<>);
                        Type genericType = template.MakeGenericType(typeArguments);
                        Activator.CreateInstance(genericType);
                    }

                }
                if (state)
                {
                    WriteToFile(objectType, tableName);
                }
            }
            else
            {
                WriteToFile(objectType, tableName);
            }
        }
    }

    private static void WriteToFile(Type objectType, String tableName)
    {
        using (var sw = new StreamWriter(Path + @"\" + objectType.Name + ".txt"))
```

```
        {
            sw.WriteLine(tableName);
            foreach (var propertyInfo in objectType.GetProperties())
            {
                if (propertyInfo.CheckNavigationPropertyAttribute())
                {
                    //skips for navigation properties
                }
                else
                {
                    sw.WriteLine("Type=" + propertyInfo.PropertyType.Name + ",Name=" + propertyInfo.Name);

                }
            }
        }
    }

    private static void DeleteTable(String tableName)
    {
        using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
        {
            try
            {
                sqliteConnection.Open();
                using (var sqliteCommand = new SqliteCommand("DROP TABLE " + tableName,sqliteConnection))
                {
                    sqliteCommand.ExecuteNonQuery();
                }
            }
            catch (Exception exception)
            {
                Logger.Log(exception.Message);
            }
        }

    }

    private static void UpdateTable(Object table, String oldTableName)
    {
        var tableType = table.GetType();
        var tableProperties = tableType.GetProperties();
        using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
        {
            try
            {
                sqliteConnection.Open();
                using (var sqliteCommand = new SqliteCommand("",sqliteConnection))
                {
                    var stringBuilder = new StringBuilder(
                        "CREATE TEMPORARY TABLE t1_backup(");
                    foreach (var tableValue in tableProperties)
                    {
                        if (tableValue.CheckNavigationPropertyAttribute())
                        {
                            //skips the navigation property from creation. Navigatin properties are only for fast object retrieval
                            //from Get or GetAll.
                        }
                        else
                        {
                            stringBuilder.Append(tableValue.Name + " " + Helpers.SwapToDbType(tableValue.PropertyType)
+ " ,");

                        }
                    }
                    stringBuilder.TrimLastChar();
                    stringBuilder.Append(");");
```

```
                    stringBuilder.Append("INSERT INTO t1_backup SELECT * FROM " + oldTableName + ";" +
                       "DROP TABLE " + oldTableName + ";");
                    stringBuilder.Append("CREATE TABLE [" + tableType.Name + "] (");
                    foreach (var tableValue in tableProperties)
                    {
                       if (tableValue.CheckNavigationPropertyAttribute())
                       {
                          //skips the navigation property from creation. Navigatin properties are only for fast object retrieval
                          //from Get or GetAll.
                       }
                       else
                       {
                          stringBuilder.Append(tableValue.Name + " " + Helpers.SwapToDbType(tableValue.PropertyType)
+ " ,");
                       }
                    }
                    stringBuilder.TrimLastChar();
                    stringBuilder.Append(");");
                    stringBuilder.Append("INSERT INTO [" + tableType.Name + "]" + "SELECT * FROM t1_backup;" +
                                "DROP TABLE t1_backup");
                    sqliteCommand.CommandText = stringBuilder.ToString();
                    sqliteCommand.ExecuteNonQuery();

                }
             }
             catch (Exception exception)
             {
                Logger.Log(exception.Message);
             }

          }
       }
    }
}
using System.Collections.Generic;
using TimeZilla.BL.Interfaces;
using TimeZilla.BL.Models;
using TimeZilla.DataLayer.Interfaces;
using TimeZilla.DataLayer.Models;

namespace TimeZilla.BL.Managers
{
    public class CategoriesManager : IManager<Category>
    {
       private readonly IRepository<CategoryDl> _categoriesRepository;

       public CategoriesManager(IRepository<CategoryDl> categoriesRepository)
       {
          _categoriesRepository = categoriesRepository;
       }

       public Category GetById(int id)
       {
          return _categoriesRepository.GetById(id).ConvertFromCategoryDlToCategory();
       }

       public void Update(Category category)
       {
          _categoriesRepository.Update(category.ConvertFromCategoryToCategoryDl());
       }

       public void Delete(int id)
       {
          _categoriesRepository.Delete(id);
       }
```

```
        public void Delete(Category category)
        {
            _categoriesRepository.Delete(category.ConvertFromCategoryToCategoryDl());
        }

        public void Insert(Category category)
        {
            _categoriesRepository.Insert(category.ConvertFromCategoryToCategoryDl());
        }

        public IEnumerable<Category> Get()
        {
            return _categoriesRepository.Get().ConvertEnumerableCategoryDlToEnumerableCategory();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Reflection;
using Mono.Data.Sqlite;
using TimeZilla.DataLayer.Attributes;
using TimeZilla.DataLayer.ExtensionMethods;

namespace TimeZilla.DataLayer.Database
{
    internal class Table<TEntity> where TEntity : class, new()
    {
        static Table()
        {
            Entities.EntitiesList.Add(Activator.CreateInstance<TEntity>());
        }

        public Table()
        {
            var entity = new TEntity();
            if (entity.GetType().GetProperty("Id") == null)
            {
                throw new Exception("The model doesnt have an Id property specified.");
            }
            var entityType = entity.GetType();
            String entityName = entityType.Name;
            using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
            {
                try
                {
                    String tableName;
                    try
                    {
                        var attribute = (TableName)Attribute.GetCustomAttribute(entityType, typeof(TableName));
                        tableName = attribute.Name;
                    }
                    catch (NullReferenceException)
                    {
                        tableName = entityName;
                    }
                    var tableValues = entityType.GetProperties();
                    sqliteConnection.Open();
                    using (var sqlCmd = new SqliteCommand("",sqliteConnection))
                    {
                        String query = "CREATE TABLE IF NOT EXISTS [" + tableName + "] (";

                        foreach (var tableValue in tableValues)
                        {
```

```
                    if (tableValue.CheckNavigationPropertyAttribute())
                    {
                        //skips the navigation property from creation. Navigatin properties are only for fast object retrieval
                        //from Get or GetAll.
                    }
                    else
                    {
                        query += tableValue.Name + " " + Helpers.SwapToDbType(tableValue.PropertyType) + " ,";
                    }
                }

                foreach (var tableValue in tableValues)
                {
                    if (tableValue.Name.EndsWith("Id") && tableValue.Name != "Id")
                    {
                        String localPropertyName = tableValue.Name.TrimEnd('d').TrimEnd('I');
                        String referencedTableName;
                        String typeString = entityType.Namespace + "." + localPropertyName + "," + entityType.Assembly;
                        var type = Type.GetType(typeString, true);
                        try
                        {
                            var obj = Activator.CreateInstance(type);
                            var attr = obj.GetType().GetCustomAttribute(typeof(TableName));
                            referencedTableName = ((TableName)attr).Name;
                        }
                        catch (NullReferenceException)
                        {
                            referencedTableName = localPropertyName;
                        }
                        if (referencedTableName == tableName)
                        {
                            query += " FOREIGN KEY (" + tableValue.Name + ")" + " REFERENCES " +
referencedTableName + " (Id),";
                        }
                        else
                        {
                            var entitiesList = new List<String>();
                            Type template = typeof(Table<>);
                            Type genericType = template.MakeGenericType(type);
                            entitiesList.Add(tableName);
                            Activator.CreateInstance(genericType, entitiesList);
                            query += "FOREIGN KEY (" + tableValue.Name + ")" + " REFERENCES " +
referencedTableName + " (Id),";

                        }
                    }
                }
                sqlCmd.CommandText = query.TrimEnd(',') + ")";
                sqlCmd.ExecuteNonQuery();
            }
        }
        catch (Exception exception)
        {
            Logger.Log(exception.Message);
        }
    }
}

public Table(List<String> entitiesList)
{
    var entity = new TEntity();
    var entityType = entity.GetType();
    String entityName = entityType.Name;
    using (var sqliteConnection = new SqliteConnection(GlobalEnvironment.ConnectionString))
    {
```

```
            try
            {
                String tableName;
                try
                {
                    var attribute = (TableName)Attribute.GetCustomAttribute(entityType, typeof(TableName));
                    tableName = attribute.Name;
                }
                catch (NullReferenceException)
                {
                    tableName = entityName;
                }
                var tableValues = entityType.GetProperties();

                sqliteConnection.Open();

                using (var sqliteCommand = new SqliteCommand("",sqliteConnection))
                {
                    String query = "CREATE TABLE IF NOT EXISTS [" + tableName + "] (";

                    foreach (var tableValue in tableValues)
                    {
                        if (tableValue.CheckNavigationPropertyAttribute())
                        {
                            //skips the navigation property from creation. Navigatin properties are only for fast object retrieval
                            //from Get or GetAll.
                        }
                        else
                        {
                            query += tableValue.Name + " " + Helpers.SwapToDbType(tableValue.PropertyType) + " ,";
                        }
                    }
                    foreach (var tableValue in tableValues)
                    {
                        if (tableValue.Name.EndsWith("Id") && tableValue.Name != "Id")
                        {
                            String localPropertyName = tableValue.Name.TrimEnd('d').TrimEnd('I');
                            String referencedTableName;
                            try
                            {
                                var attr = tableValue.GetCustomAttribute(typeof(TableName));
                                referencedTableName = ((TableName)attr).Name;
                            }
                            catch (NullReferenceException)
                            {
                                referencedTableName = localPropertyName;
                            }
                            if (referencedTableName == tableName || entitiesList.Contains(referencedTableName))
                            {
                                query += " FOREIGN KEY (" + tableValue.Name + ")" + " REFERENCES " +
referencedTableName + " (Id),";
                            }
                            else
                            {
                                entitiesList.Add(tableName);
                                var t = Type.GetType(entity.GetType().Namespace + "." + localPropertyName);
                                Type template = typeof(Table<>);
                                Type genericType = template.MakeGenericType(t);
                                Activator.CreateInstance(genericType, entitiesList);
                                query += "FOREIGN KEY (" + tableValue.Name + ")" + " REFERENCES " +
referencedTableName + " (Id),";
                            }
                        }
                    }
                    sqliteCommand.CommandText = query.TrimEnd(',') + ")";
```

```
                    sqliteCommand.ExecuteNonQuery();
                }
            }
            catch (Exception exception)
            {
                Logger.Log(exception.Message);
            }
        }
    }


  }
}

using System.Collections.Generic;
using TimeZilla.BL.Interfaces;
using TimeZilla.BL.Models;
using TimeZilla.DataLayer.Interfaces;
using TimeZilla.DataLayer.Models;

namespace TimeZilla.BL.Managers
{
  class GoalsManager : IManager<Goal>
  {
    private readonly IRepository<GoalDl> _goalsRepository;

    public GoalsManager(IRepository<GoalDl> goalsRepository)
    {
      _goalsRepository = goalsRepository;
    }

    public Goal GetById(int id)
    {
      return _goalsRepository.GetById(id).ConvertFromGoalDlToGoal();
    }

    public void Update(Goal goal)
    {
      _goalsRepository.Update(goal.ConvertFromGoalToGoalDl());
    }

    public void Delete(int id)
    {
      _goalsRepository.Delete(id);
    }

    public void Delete(Goal goal)
    {
      _goalsRepository.Delete(goal.ConvertFromGoalToGoalDl());
    }

    public void Insert(Goal goal)
    {
      _goalsRepository.Insert(goal.ConvertFromGoalToGoalDl());
    }

    public IEnumerable<Goal> Get()
    {
      return _goalsRepository.Get().ConvertEnumerableGoalDlToEnumerableGoal();
    }
  }
}
```

ДОДАТОК В

ЛІСТИНГ РОЗДІЛЮВАЧА ТЕКСТУ

```
>>> from nltk.chunk import *
>>> from nltk.chunk.util import *
>>> from nltk.chunk.regexp import *
>>> from nltk import Tree
>>> tagged_text = "[ The/DT cat/NN ] sat/VBD on/IN [ the/DT mat/NN ] [ the/DT dog/NN ] chewed/VBD ./."
>>> gold_chunked_text = tagstr2tree(tagged_text)
>>> unchunked_text = gold_chunked_text.flatten()
>>> tag_pattern = "<DT>?<JJ>*<NN.*>"
>>> regexp_pattern = tag_pattern2re_pattern(tag_pattern)
>>> regexp_pattern
'(<(DT)>)?(<(JJ)>)*(<(NN[^\\{\\}<>]*)>)'
>>> chunk_rule = ChunkRule("<.*>+", "Chunk everything")
>>> chink_rule = ChinkRule("<VBD|IN|\.>", "Chink on verbs/prepositions")
>>> split_rule = SplitRule("<DT><NN>", "<DT><NN>",
...                        "Split successive determiner/noun pairs")
>>> chunk_parser = RegexpChunkParser([chunk_rule], chunk_label='NP')
>>> chunked_text = chunk_parser.parse(unchunked_text)
>>> print(chunked_text)
(S
  (NP
    The/DT
    cat/NN
    sat/VBD
    on/IN
    the/DT
    mat/NN
```

```
    the/DT
    dog/NN
    chewed/VBD
    ./.))
>>> chunkscore = ChunkScore()
>>> chunkscore.score(gold_chunked_text, chunked_text)
>>> print(chunkscore.precision())
0.0
>>> print(chunkscore.recall())
0.0
>>> print(chunkscore.f_measure())
0
>>> for chunk in sorted(chunkscore.missed()): print(chunk)
(NP The/DT cat/NN)
(NP the/DT dog/NN)
(NP the/DT mat/NN)
>>> for chunk in chunkscore.incorrect(): print(chunk)
(NP
  The/DT
  cat/NN
  sat/VBD
  on/IN
  the/DT
  mat/NN
  the/DT
  dog/NN
  chewed/VBD
  ./.)
>>> chunk_parser = RegexpChunkParser([chunk_rule, chink_rule],
...                                  chunk_label='NP')
>>> chunked_text = chunk_parser.parse(unchunked_text)
```

```
>>> print(chunked_text)
(S
  (NP The/DT cat/NN)
  sat/VBD
  on/IN
  (NP the/DT mat/NN the/DT dog/NN)
  chewed/VBD
  ./.)
>>> assert chunked_text == chunk_parser.parse(list(unchunked_text))
>>> chunkscore = ChunkScore()
>>> chunkscore.score(gold_chunked_text, chunked_text)
>>> chunkscore.precision()
0.5
>>> print(chunkscore.recall())
0.33333333...
>>> print(chunkscore.f_measure())
0.4
>>> for chunk in sorted(chunkscore.missed()): print(chunk)
(NP the/DT dog/NN)
(NP the/DT mat/NN)
>>> for chunk in chunkscore.incorrect(): print(chunk)
(NP the/DT mat/NN the/DT dog/NN)
>>> chunk_parser = RegexpChunkParser([chunk_rule, chink_rule,
split_rule],
...                                   chunk_label='NP')
>>> chunked_text = chunk_parser.parse(unchunked_text, trace=True)
# Input:
 <DT> <NN> <VBD> <IN> <DT> <NN> <DT> <NN> <VBD>
<.>
# Chunk everything:
```

{<DT> <NN> <VBD> <IN> <DT> <NN> <DT> <NN> <VBD> <.>}

Chink on verbs/prepositions:

{<DT> <NN>} <VBD> <IN> {<DT> <NN> <DT> <NN>} <VBD> <.>

Split successive determiner/noun pairs:

{<DT> <NN>} <VBD> <IN> {<DT> <NN>}{<DT> <NN>} <VBD> <.>

```
>>> print(chunked_text)
(S
  (NP The/DT cat/NN)
  sat/VBD
  on/IN
  (NP the/DT mat/NN)
  (NP the/DT dog/NN)
  chewed/VBD
  ./.)
>>> chunkscore = ChunkScore()
>>> chunkscore.score(gold_chunked_text, chunked_text)
>>> chunkscore.precision()
1.0
>>> chunkscore.recall()
1.0
>>> chunkscore.f_measure()
1.0
>>> chunkscore.missed()
[]
>>> chunkscore.incorrect()
[]
>>> chunk_parser.rules() # doctest: +NORMALIZE_WHITESPACE
[<ChunkRule: '<.*>+'>, <ChinkRule: '<VBD|IN|\\.>'>,
```

```
 <SplitRule: '<DT><NN>', '<DT><NN>'>]
>>> print(repr(chunk_parser))
<RegexpChunkParser with 3 rules>
>>> print(chunk_parser)
RegexpChunkParser with 3 rules:
    Chunk everything
      <ChunkRule: '<.*>+'>
    Chink on verbs/prepositions
      <ChinkRule: '<VBD|IN|\\.>'>
    Split successive determiner/noun pairs
      <SplitRule: '<DT><NN>', '<DT><NN>'>
>>> ChunkParserI().parse([])
Traceback (most recent call last):
  . . .
NotImplementedError
>>> t1 = Tree('S', [('w%d' % i, 't%d' % i) for i in range(10)])
>>> t2 = Tree('S', [Tree('t0', []), Tree('t1', ['c1'])])
>>> t3 = Tree('S', [('w0', 't0'), Tree('t1', ['c1'])])
>>> ChunkString(t1)
<ChunkString: '<t0><t1><t2><t3><t4><t5><t6><t7><t8><t9>'>
>>> ChunkString(t2)
<ChunkString: '<t0><t1>'>
>>> ChunkString(t3)
<ChunkString: '<t0><t1>'>
>>> ChunkString(Tree('S', ['x']))
Traceback (most recent call last):
>>> cs = ChunkString(t1)
>>> print(cs)
 <t0> <t1> <t2> <t3> <t4> <t5> <t6> <t7> <t8> <t9>
>>> cs.xform('<t3>', '{<t3>}')
>>> print(cs)
```

```
 <t0> <t1> <t2> {<t3>} <t4> <t5> <t6> <t7> <t8> <t9>
>>> cs = ChunkString(t1, debug_level=3)
>>> # tag not marked with <...>:
>>> cs.xform('<t3>', 't3')
Traceback (most recent call last):
  <t0><t1><t2>t3<t4><t5><t6><t7><t8><t9>
>>> # brackets not balanced:
>>> cs.xform('<t3>', '{<t3>')
t0><t1><t2>{<t3><t4><t5><t6><t7><t8><t9>
>>> # nested brackets:
>>> cs.xform('<t3><t4><t5>', '{<t3>{<t4>}<t5>}')
Traceback (most recent call last):
  . . .
  <t0><t1><t2>{<t3>{<t4>}<t5>}<t6><t7><t8><t9>
>>> # modified tags:
>>> cs.xform('<t3>', '<t9>')
Traceback (most recent call last):
  . . .
>>> # added tags:
>>> cs.xform('<t9>', '<t9><t10>')
Traceback (most recent call last):
  . . .
>>> r1 = RegexpChunkRule('<a|b>'+ChunkString.IN_CHINK_PATTERN,
...                       '{<a|b>}', 'chunk <a> and <b>')
>>> r2 = RegexpChunkRule(re.compile('<a|b>'+ChunkString.IN_CHINK_PATTERN),
...                       '{<a|b>}', 'chunk <a> and <b>')
>>> r3 = ChunkRule('<a|b>', 'chunk <a> and <b>')
>>> r4 = ChinkRule('<a|b>', 'chink <a> and <b>')
```

```
>>> r5 = UnChunkRule('<a|b>', 'unchunk <a> and <b>')
>>> r6 = MergeRule('<a>', '<b>', 'merge <a> w/ <b>')
>>> r7 = SplitRule('<a>', '<b>', 'split <a> from <b>')
>>> r8 = ExpandLeftRule('<a>', '<b>', 'expand left <a> <b>')
>>> r9 = ExpandRightRule('<a>', '<b>', 'expand right <a> <b>')
>>> for rule in r1, r2, r3, r4, r5, r6, r7, r8, r9:
...     print(rule)
<RegexpChunkRule: '<a|b>(?=[^\\}]*(\\{|$))'->'{<a|b>}'>
<RegexpChunkRule: '<a|b>(?=[^\\}]*(\\{|$))'->'{<a|b>}'>
<ChunkRule: '<a|b>'>
<ChinkRule: '<a|b>'>
<UnChunkRule: '<a|b>'>
<MergeRule: '<a>', '<b>'>
<SplitRule: '<a>', '<b>'>
<ExpandLeftRule: '<a>', '<b>'>
<ExpandRightRule: '<a>', '<b>'>
>>> tag_pattern2re_pattern('{}')
Traceback (most recent call last):
  . . .
>>> parser = RegexpChunkParser([ChunkRule('<a>', '')])
>>> parser.parse(Tree('S', []))
Warning: parsing empty text
Tree('S', [])
>>> parser = RegexpParser('''
... NP: {<DT>? <JJ>* <NN>*} # NP
... P: {<IN>}           # Preposition
... V: {<V.*>}          # Verb
... PP: {<P> <NP>}      # PP -> P NP
... VP: {<V> <NP|PP>*}  # VP -> V (NP|PP)*
... ''')
>>> print(repr(parser))
```

```
<chunk.RegexpParser with 5 stages>
>>> print(parser)
chunk.RegexpParser with 5 stages:
RegexpChunkParser with 1 rules:
   NP   <ChunkRule: '<DT>? <JJ>* <NN>*'>
RegexpChunkParser with 1 rules:
   Preposition   <ChunkRule: '<IN>'>
RegexpChunkParser with 1 rules:
   Verb   <ChunkRule: '<V.*>'>
RegexpChunkParser with 1 rules:
   PP -> P NP   <ChunkRule: '<P> <NP>'>
RegexpChunkParser with 1 rules:
   VP -> V (NP|PP)*   <ChunkRule: '<V> <NP|PP>*'>
>>> print(parser.parse(unchunked_text, trace=True))
# Input:
 <DT>  <NN>  <VBD>  <IN>  <DT>  <NN>  <DT>  <NN>  <VBD>
<.>
# NP:
{<DT>   <NN>}  <VBD>   <IN>  {<DT>   <NN>}{<DT>   <NN>}
<VBD>  <.>
# Input:
 <NP>  <VBD>  <IN>  <NP>  <NP>  <VBD>  <.>
# Preposition:
 <NP>  <VBD> {<IN>} <NP>  <NP>  <VBD>  <.>
# Input:
 <NP>  <VBD>  <P>  <NP>  <NP>  <VBD>  <.>
# Verb:
 <NP> {<VBD>} <P>  <NP>  <NP> {<VBD>} <.>
# Input:
 <NP>  <V>  <P>  <NP>  <NP>  <V>  <.>
# PP -> P NP:
```

```
 <NP> <V> {<P> <NP>} <NP> <V> <.>
# Input:
 <NP> <V> <PP> <NP> <V> <.>
# VP -> V (NP|PP)*:
 <NP> {<V> <PP> <NP>}{<V>} <.>
(S
  (NP The/DT cat/NN)
  (VP
    (V sat/VBD)
    (PP (P on/IN) (NP the/DT mat/NN))
    (NP the/DT dog/NN))
  (VP (V chewed/VBD))
  ./.)
>>> print(RegexpParser('''
... X:
...   }<a><b>{     # chink rule
...   <a>}{<b>     # split rule
...   <a>{}<b>     # merge rule
...   <a>{<b>}<c>  # chunk rule w/ context
... '''))
chunk.RegexpParser with 1 stages:
RegexpChunkParser with 4 rules:
    chink rule             <ChinkRule: '<a><b>'>
    split rule             <SplitRule: '<a>', '<b>'>
    merge rule             <MergeRule: '<a>', '<b>'>
    chunk rule w/ context  <ChunkRuleWithContext: '<a>', '<b>',
'<c>'>
```

Illegal patterns give an error message:

```
>>> print(RegexpParser('X: {<foo>} {<bar>}'))
Traceback (most recent call last):
```

```
  . . .
ValueError: Illegal chunk pattern: {<foo>} {<bar>}

class ConsecutiveNPChunkTagger(nltk.TaggerI):

    def __init__(self, train_sents):
        train_set = []
        for tagged_sent in train_sents:
            untagged_sent = nltk.tag.untag(tagged_sent)
            history = []
            for i, (word, tag) in enumerate(tagged_sent):
                featureset = npchunk_features(untagged_sent, i, history)
                train_set.append( (featureset, tag) )
                history.append(tag)
        self.classifier = nltk.MaxentClassifier.train(
            train_set, algorithm='megam', trace=0)

    def tag(self, sentence):
        history = []
        for i, word in enumerate(sentence):
            featureset = npchunk_features(sentence, i, history)
            tag = self.classifier.classify(featureset)
            history.append(tag)
        return zip(sentence, history)

class ConsecutiveNPChunker(nltk.ChunkParserI):
    def __init__(self, train_sents):
        tagged_sents = [[((w,t),c) for (w,t,c) in
                         nltk.chunk.tree2conlltags(sent)]
                        for sent in train_sents]
        self.tagger = ConsecutiveNPChunkTagger(tagged_sents)
```

```
def parse(self, sentence):
    tagged_sents = self.tagger.tag(sentence)
    conlltags = [(w,t,c) for ((w,t),c) in tagged_sents]
    return nltk.chunk.conlltags2tree(conlltags)
```

ДОДАТОК Д

ТЕСТОВІ ВИПАДКИ

Таблиця Д.1

Тестовий випадок перегляду статистики

№ кроку	Опис кроку	Очікуваний результат	Фактичний результат	Стан
1	Обрати пункт меню «Статистика»	Виконання запиту успішно виконалось і результат відобразився на екрані.	Виконання запиту успішно виконалось і результат відобразився на екрані.	Успішно
2	Обрати підпункт «перегляд статистики»			Успішно
3	Вибрати межі.			Успішно

Таблиця Д.2

Тестовий випадок реєстрації

№ кроку	Опис кроку	Очікуваний результат	Фактичний результат	Стан
1	Обрати пункт меню «Реєстрація»	Виконання запиту успішно виконалось.	Виконання запиту успішно виконалось.	Успішно
2	Ввести необхідні дані			Успішно
3	Натиснути кнопку «Зареєструватися»			Успішно
4	Авторизуватися після успішно реєстрації.			Успішно

Таблиця Д.3

Тестовий випадок створення провайдера

№ кроку	Опис кроку	Очікуваний результат	Фактичний результат	Стан
1	Авторизуватися як адміністратор.	Виконання запиту успішно виконалось і результат відобразився у формі із списком провайдерів	Виконання запиту успішно виконалось і результат відобразився у формі із списком провайдерів	Успішно
2	Обрати підпункт «Провайдери»			Успішно
3	Вибрати «Додати нового»			Успішно
4	Ввести необхідні дані			Успішно
5	Натиснути на кнопку «Зберегти»			Успішно

ДОДАТОК Е

КОПІЇ ПУБЛІКАЦІЇ

(кластери), знаходження спільних груп і в результаті розсилання кореспонденції тільки тим хто найбільш зацікавлений у ній.

Ця система дозволить краще оптимізувати процес розповсюдження інформації в колах користувача системи.

Список використаних джерел

1. Лекции по алгоритмам кластеризации и многомерного шкалирования [Електронний ресурс] //К. В. Воронцов - 21.12.2007. - 18 с. Режим доступу до ресурсу: http://www.ccas.ru/voron/download/Clustering.pdf.
2. Обзор алгоритмов кластеризации данных [Електронний ресурс] – 2010 - Режим доступу до ресурсу: https://habrahabr.ru/post/101338/
3. Применение методов кластеризации для обработки новостного потока [Електронний ресурс] – Режим доступу до ресурсу: http://www.moluch.ru/conf/tech/archive/2/207/

УДК 004.62

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСУ ЗАПОВНЕННЯ СХОВИЩА ДАНИХ ДЛЯ СИСТЕМИ ТАЙМ МЕНЕДЖМЕНТУ

Пойдич В.С.[1], Струбицька І.П.[2]

Тернопільський національний економічний університет

[1] магістрант; [2] к.т.н., доцент

І. Постановка проблеми

Тайм менеджмент – один з найважливіших критеріїв успішності у бізнесі. «Час це гроші», - не даремно так кажуть. Тому в період новітніх технологій необхідно максимально автоматизовувати процеси, які більшою мірою є марудними при ручному виконанні, але які є істотними для успішності в різних сферах діяльності. Розподіл часу використовується всюди: від простого нагадування про те що і коли треба зробити, до потужного графіку, від якого залежить успішність підприємства. І як вже було сказано, заповнення сховища даних іноді є довгим процесом, що безпосередньо потребує ще часу, який би міг бути використаний на інші важливіші справи. Було розроблено багато методологій по правильному менеджменті часу, як і коли отримувати найкращий результат, як виділяти категорії, як все ж таки встигати і керувати бізнесом і проводити час з сім'єю [1-3].

ІІ. Мета роботи

Метою дослідження є розробка алгоритму, який буде виконувати функцію ефективного розподілу часу відносно певних критеріїв, які можуть бути задані користувачем або визначені системою, і відповідно до результатів будувати максимально гнучкий графік виконання поставлених задач. Також система буде сама навчатись відносно реакцій користувача.

ІІІ. Особливості програмної реалізації продукту

Система буде реалізована на мові C# з використанням технології Xamarin [4], оскільки вона буде орієнтована на мобільні платформи з ОС Android або iOS. Програмна система буде розроблена з використанням алгоритмів, які будуть активно слухати і запам'ятовувати вибори користувача, відносно однієї або іншої категорій. При розробці, як основний критерій, було вирішено використати вже готове розділення Ейзенхауера. Логіка полягає в тому що завдання діляться на 4 категорії:

1. Термінові-важливі;
2. Термінові-неважливі;
3. Нетермінові-важливі;
4. Нетермінові – неважливі.

Саме такий розподіл дозволяє найкраще відобразити все сховище даних, і краще аналізувати дані та подавати користувачу найвідповідніші задачі. На рис. 1 показано куб Ейзенхауера.

	Термінові	Нетермінові
Важливі	Важливі та термінові	Важливі,але нетермінові
Неважливі	Термінові,але неважливі	Неважливі і нетермінові

Рисунок 1 – Куб Ейзенхауера

Все ж таки, на реальному прикладі задачі з категорії неважливі-нетермінові практично ніколи не зустрічаються, тому у розроблюваній системі ця частина куба буде опущена, що дозволить пришвидшити аналіз даних, оскільки відбулося зменшення кількості категорій.

Метод Ейзенхауера не єдине від чого буде відштовхуватися система, оскільки завдання можуть бути розділені і за іншими категоріями, наприклад час виконання, пора року, частина дня (обід, ранок), або категоріями, які будуть безпосередньо встановленими користувачами. Але основним критерієм при аналізі будуть саме ці 3 категорії. На рис. 2 представлено схему процесу заповнення сховища.

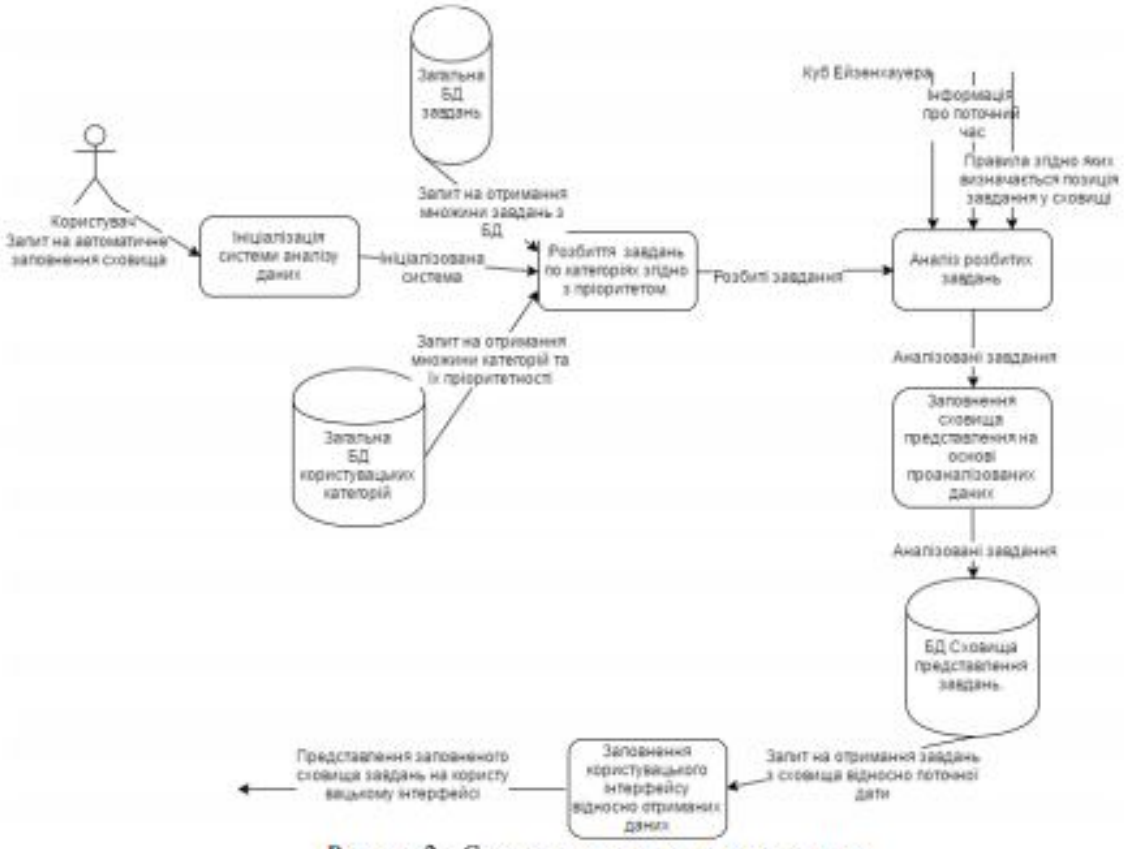

Рисунок 2 – Схема процесу заповнення сховища.

Висновок

У роботі досліджено задачу оптимізації процесу заповнення сховища даних, запропоновано розробити алгоритм аналізу завдань на основі методу Ейзенхауера, а саме – виділення основних трьох категорій, на основі яких буде будуватись фінальний результат, при зменшенні кількості категорій на 1, що дозволить пришвидшити процес аналізу. Для більшої гнучкості роботи було вирішено надати користувачу доступ до створення додаткових критеріїв та передбачення обробки цих критеріїв алгоритмом.

Список використаних джерел

1. Stephen Covey The Seven Habits of Highly Effective People - 1980. – 380с.
2. Le Blanc, Raymond. Achieving Objectives Made Easy! Practical goal setting tools & proven time management techniques. Maarheeze: Cranendonck Coaching - 2008. - 140с.
3. Lewis-Beck, Michael S. Data Analysis: an Introduction – 1995. – 77с.
4. Xamarin documentation [Електронний ресурс]. Режим доступу: https://docs.xamarin.com (дата звернення 26.04.2016) – Cross-Platform.

УДК 004.89

ОСОБЛИВОСТІ ВИКОРИСТАННЯ BEHAVIORAL TARGETING ДЛЯ ЗАДАЧІ ТАЙМ МЕНЕДЖМЕНТУ

Поляруш О.В.[1], Струбицька І.П.[2]

Тернопільський національний економічний університет

[1] магістрант; [2] к.т.н., доцент

І. Постановка проблеми

На сьогоднішній день питання правильно розподілу часу є одним із найважливіших для людини, адже питання правильного розподілу часу та розставлення пріоритетів задач є одним із найважливіших для людини. Закони та принципи тайм менеджменту широко використовуються серед різних видів діяльності. Це дає змогу пришвидшити та зекономити витрати на виробництво.

ІІ. Мета роботи

Метою дослідження є аналіз можливостей використання «Behavioral targeting» для доцільного розподілу задач згідно вподобань користувача. Основними функціональними вимогами до програмного продукту є:

- підбір подій та задач згідно інтересів користувача;
- побудови достовірного рейтингу подій;
- виявлення схованих зв'язків між задачами.

ІІІ. Особливості програмної реалізації системи розподілу задач

Аналіз потреб користувача базується на отри манній базі даних, котра формується протягом використання програмного продукту. Система має певні критерії розподілу отриманих даних, які можуть бути внесені користувачем або розподілені на системні категорії. Кожна категорія буде містити пріоритет для користувача, що дає змогу сформулювати карту потреб. На базі отриманої карти, ми можемо заповнити календар подій. Також використовуються отримані дані при аналізі оточення та дані стосовно поточного розміщення користувача.

Для аналізу та формування календаря подій в системі планується реалізувати модуль аналізу даних із використання підходу «Behavioral targeting», що дасть змогу змоделювати карту потреб користувача із використанням отриманих під час використання програми.