Додаток А

**Технічне завдання**

1. Загальне положення

1.1. Найменування програмного засобу

Повне найменування програмної системи: "База даних виконаних робіт з ландшафтного дизайну" (надалі "програма"). Коротка назва програмної системи - "Дизайн"

1.2. Призначення розробки та область застосування

Програмна система "База даних виконаних робіт з ландшафтного дизайну" призначена для збору, обробки та відображення даних про виконані роботи у сфері ландшафтного дизайну. В програмі передбачено використання механізму пошуку по ландшафтним дизайнерам – виконавцям, та з формуванням списку усіх виконаних ними робіт.

Система може бути впроваджена у роботу агентства по ландшафтному дизайну з кількістю ландшафтних дизайнерів до 10000.

1.3.Мета

Програмна система " База даних виконаних робіт з ландшафтного дизайну " дозволить підвищити ефективність, повноту та швидкість роботи клієнтів під час вибору необхідного їм ландшафтного дизайнера, да дизайнерам краще розуміти де їм доведеться працювати, та забезпечить керівництво підприємства своєчасною та повною інформацією про актуальний стан роботи у повному обсязі.

1.4. Найменування розробника та замовника.

Розробник даного продукту - студент групи ПІ-52 Ксенич Євгеній (надалі "розробник").

Замовник програмного продукту - кафедра програмного забезпечення обчислюваної техніки Житомирського державного технологічного університету в межах виконання курсової з дисципліни «Бази даних» (надалі замовник).

2. Підстава для розробки

2.1. Документ на підставі якого ведеться розробка

Робота ведеться на підставі навчального плану за напрямом 6.050201 «Програмна інженерія» наказу про закріплення тем курсових робіт за студентами.

3. Вимоги до програми

3.1. Вимоги до функціональних характеристик.

3.1.1. Загальні вимоги

Програмна система має забезпечувати:

* оптимальне збереження даних (за обсягом та структурою);
* аутентифікацію користувачів та захист інформації від несанкціонованого доступу;
* надійне збереження даних.

3.1.1.Складвиконуванихфункцій

Розробити база даних виконаних робіт з ландшафтного дизайну, що підтримує виконання наступних транзакцій:

1. Реєстрацію користувачів;
2. Додавання користувачів до таблиці Замовників.
3. Закріплення території до Замовника.
4. Додавання користувачів до таблиці Виконавців.
5. Додавання виконаних робіт до замовлення.
6. Додавання труднощів, що були подолані до замовлення.
7. Додавання замовлень.
8. Пошук та відбір замовлень за мультикритеріями (виконавець, замовник, дата початку, дата кінця тощо).
9. Відбір та впорядкування виконавців за мультикритеріями (ім’я, прізвище, тощо).
10. Формування звітності щодо замовлень, середньої кількості виконаних робіт у одному замовленні. Перелік звітних форм наступний:

* Звіт по замовленням по датам, ціні, кількості виконаних робіт;
* Звіт по середній кількості виконаних робіт у 1 замовленні.

3.1.2. Організація вхідних і вихідних даних

Вхідними даними є дані про замовлення, виконавців, замовників.

Організація вхідних і вихідних даних повинна відповідати інформаційній структурі виконуваних з операцій, вхідним та вихідним паперовим документами.

Введення оперативних даних повинно виконуватися з використанням діалогових екранних форм, побудованих на основі візуальних компонентів.

Будь який звіт повинен мати можливість виводу на друк

3.1.3. Часові характеристики і розмір пам'яті, необхідної для роботи програми.

Час реакції програми на дії користувача (маніпуляції з пристроями введення даних) не повинен перевищувати 0,25 с.

Час виконання команд меню не більше 1 с.

Відображення масивів даних за запитами не більше 3 хвилин.

Доступність БД – 90% цілодобово.

Операції з’єднання з БД не більше 1 хвилини.

Обсяг оперативної пам'яті, необхідний для роботи програми не менше 2Гб.

Дисковій простір, необхідний для збереження програми і файлів даних не більше 300 Мбат для робочої станції та 20 ГБайт..

Інсталяційний пакет програми, що містить у складі БД не повинні перевищувати 100 Мбайт.

3.2. Вимоги до надійності.

3.2.1. Вимоги до надійного функціонування

Програма повинна нормально функціонувати при безперебійній роботі ПК. Доступність БД 90% при одночасному доступі 30 користувачів.

При апаратних збоях, відновлення нормальної роботи програми повинне виконуватися після:

а) апаратні збої сервера - перезавантаження ОС сервера, запуск сервера БД (запуску резервного сервера, використання технологій RAID для збереження даних);

б)апаратні збої робочої станції – перезавантаження ОС ПК, перехід на адресу веб-ресурсу.

При збоях програмного забезпечення:

а) система повинна забезпечувати можливість відновлення даних та фіксацію і «відкат» транзакцій.

б) в системі має бути реалізована коректна обробка виняткових ситуацій.

3.2.2.Контроль вхідної і вихідної інформації

Для контролю коректності вхідної інформації та захисту від помилок оператора:

* Перевірка відповідності даних доменам інформаційних атрибутів;
* Використанням механізму авто заповнення та вибору за переліком для зв’язаних даних;

Визначені некоректні дії повинні супроводжуватись повідомленнями про помилку і блокуванням операцій оновлення даних. В системі має бути передбачений захист від загального блокування.

Для вихідної інформації передбачити:

* відсутність блокування даних через багато користувальницький доступ;
* постійне оновлення даних у відображених на екрані звітних формах.

3.2.3. Час відновлення після відмови.

Час відновлення після відмови, не пов'язаною з роботою програми, повинен складатися із: часу перезапуску операційної системи; часу запуску сервера БД (підключення до сервера ) запуску виконуваного файлу, часу повторного введення або зчитування даних.

3.3. Умови експлуатації і збереження

Програма використовується у багато користувальницькому середовище. Регламенті операції проводити за графіком:

* створення резервних копій даних: 1 раз на тиждень;
* оновлення системи ідентифікації та аутентифікації користувачів – 1 раз на місяць та за потребою.

Збереження архівних копій - на визначених носіях. Збереження резервних копій – на визначеному диску без перезапису. Знищення копій за регламентом.

3.4. Вимоги до інформаційної і програмної сумісності.

3.4.2. Вимоги до методів рішення і мов програмування

Вибір методів рішення здійснюється розробникам без узгодження з замовником. СУБД обирається у відповідності до характеристик визначених в п.3. З замовником погоджується вибір варіанту за вартісною ознакою.

3.4.3. Вимоги до системи програмних засобів.

Вимоги до програмного забезпечення сервера:

ОС – серверна версія ОС Windows не нижче Windows 2000 Server, IIS (або інший сервер), СУБД визначається встановлюється та налаштовується розробником (ліцензування СУБД виконується замовником).

Вимоги до програмного забезпечення робочої станції:

Доступ до ресурсу інтернет.

3.5. Вимоги до складу і параметрів технічних засобів.

Вимоги до складу технічних засобів:

1. Сервер:

* сервер у базовій конфігурації із підтримкою RAID;
* з’ємний запам’ятовуючий пристрій для архівування даних;

2. Робоча станція:

* оперативна пам’ять – 1Гб;
* доступ до ресурсів інтернет.

4. Вимоги до програмної документації

Програмна документація повинна включати наступні відомості:

1. Керівництво користувача" складається з опису послідовності опису роботи з веб-додатком, основних режимів роботи, опису основних форм, переліку виняткових ситуацій та реакції користувача на них, порядку виконання завдань в системі. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться в 3 розділі курсової роботи.
2. "Керівництво адміністратора баз даних", складається з опису складу таблиць бази даних та правил доступів до них, опису та послідовності робіт по обслуговуванню бази даних (архівування, резервне копіювання, з вказівкою періодичності виконання та засобів, що для цього використовуються), порядку налаштування серверу та клієнтських додатків. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться в 2,3 та 4 розділах курсової роботи згідно плану.

public partial class LD\_kursEntities : DbContext

{

public LD\_kursEntities()

: base("name=LD\_kursEntities")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

public virtual DbSet<customer> customer { get; set; }

public virtual DbSet<designer> designer { get; set; }

public virtual DbSet<difficulties> difficulties { get; set; }

public virtual DbSet<land> land { get; set; }

public virtual DbSet<sysdiagrams> sysdiagrams { get; set; }

public virtual DbSet<work> work { get; set; }

public virtual DbSet<zakaz> zakaz { get; set; }

}

public partial class customer

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public customer()

{

this.land = new HashSet<land>();

}

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public System.DateTime dateOfBirth { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<land> land { get; set; }

}

public partial class designer

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public designer()

{

this.zakaz = new HashSet<zakaz>();

}

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public System.DateTime dateOfBirth { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<zakaz> zakaz { get; set; }

}

public partial class difficulties

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public difficulties()

{

this.zakaz = new HashSet<zakaz>();

}

public int id { get; set; }

public string subj { get; set; }

public int price { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<zakaz> zakaz { get; set; }

}

public partial class land

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public land()

{

this.zakaz = new HashSet<zakaz>();

}

public int id { get; set; }

public string name { get; set; }

public int customer\_id { get; set; }

public int size { get; set; }

public string addres { get; set; }

public virtual customer customer { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<zakaz> zakaz { get; set; }

}

public partial class work

{

public int id { get; set; }

public string typee { get; set; }

public int countt { get; set; }

public int price { get; set; }

public int zakazId { get; set; }

public virtual zakaz zakaz { get; set; }

}

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public zakaz()

{

this.work = new HashSet<work>();

this.difficulties = new HashSet<difficulties>();

}

public int id { get; set; }

public int designer\_id { get; set; }

public int land\_id { get; set; }

//public int price { get; set; }

public System.DateTime start\_time { get; set; }

public System.DateTime end\_time { get; set; }

public virtual designer designer { get; set; }

public virtual land land { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<work> work { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<difficulties> difficulties { get; set; }

[DatabaseGenerated(DatabaseGeneratedOption.Computed)]

public int price

{

get

{

int sum = work.Sum(item => item.price \* item.countt);

if (land != null)

sum += land.size \* 15;

sum += difficulties.Sum(item => item.price);

return sum;

}

private set { /\* needed for EF \*/ }

}

public class ZakazInfoRepositories : IZakazInfo<ZakazInfo, zakaz>

{

public IEnumerable<ZakazInfo> GetList()

{

List<zakaz> zakaz;

List<ZakazInfo> InfoList = new List<ZakazInfo>();

using (LD\_kursEntities db = new LD\_kursEntities())

{

zakaz = db.zakaz.ToList<zakaz>();

foreach (var item in zakaz)

{

InfoList.Add(new ZakazInfo()

{

zakaz = Mapper.Map<Zakaz>(item),

land = Mapper.Map<Land>(item.land),

customer = Mapper.Map<Customer>(item.land.customer),

designer = Mapper.Map<Designer>(item.designer),

difficults = Mapper.Map<List<Difficulties>>(item.difficulties),

work = Mapper.Map<List<Work>>(item.work)

});

}

}

return InfoList;

}

public IEnumerable<ZakazInfo> GetList(List<zakaz> zakaz)

{

List<ZakazInfo> InfoList = new List<ZakazInfo>();

using (LD\_kursEntities db = new LD\_kursEntities())

{

foreach (var item in zakaz)

{

InfoList.Add(new ZakazInfo()

{

zakaz = Mapper.Map<Zakaz>(item),

land = Mapper.Map<Land>(item.land),

customer = Mapper.Map<Customer>(item.land.customer),

designer = Mapper.Map<Designer>(item.designer),

difficults = Mapper.Map<List<Difficulties>>(item.difficulties),

work = Mapper.Map<List<Work>>(item.work)

});

}

}

return InfoList;

}

public IEnumerable<ZakazInfo> GetList(string nameDesigner, string surnameDesigner,

string nameCustomer, string surnameCustomer, string price,

string startDate, string endDate)

{

IEnumerable<ZakazInfo> Zakaz;

using (LD\_kursEntities db = new LD\_kursEntities())

{

var zakaz = db.zakaz

.Where(item => item.designer.name.Contains(nameDesigner))

.Where(item => item.designer.surname.Contains(surnameDesigner))

.Where(item => item.land.customer.name.Contains(nameCustomer))

.Where(item => item.land.customer.surname.Contains(surnameCustomer))

.Where(item => item.price.ToString().Contains(price))

.Where(item => item.start\_time.ToString().Contains(startDate))

.Where(item => item.end\_time.ToString().Contains(endDate))

.ToList();

Zakaz = GetList(zakaz);

}

return Zakaz;

}

public ZakazInfo GetbyId(int id)

{

ZakazInfo ZkazInfo = new ZakazInfo();

using (LD\_kursEntities db = new LD\_kursEntities())

{

var zakaz = db.zakaz.SingleOrDefault(c => c.id == id);

ZkazInfo = new ZakazInfo()

{

zakaz = Mapper.Map<Zakaz>(zakaz),

land = Mapper.Map<Land>(zakaz.land),

customer = Mapper.Map<Customer>(zakaz.land.customer),

designer = Mapper.Map<Designer>(zakaz.designer),

difficults = Mapper.Map<List<Difficulties>>(zakaz.difficulties),

work = Mapper.Map<List<Work>>(zakaz.work)

};

}

return ZkazInfo;

}

public ZakazInfo Delete(int id)

{

ZakazInfo Info = GetbyId(id);

using (LD\_kursEntities db = new LD\_kursEntities())

{

var zakaz = db.zakaz.SingleOrDefault(c => c.id == id);

if (zakaz == null)

return null;

using (var transaction = db.Database.BeginTransaction())

{

try

{

foreach (var item in zakaz.difficulties.ToList())

{

db.Entry(item).State = EntityState.Deleted;

}

foreach (var item in zakaz.work.ToList())

{

db.Entry(item).State = EntityState.Deleted;

}

db.Entry(zakaz).State = EntityState.Deleted;

db.SaveChanges();

transaction.Commit();

}

catch (Exception ex)

{

transaction.Rollback();

}

}

}

return Info;

}

public OrderReport GetReport(int id)

{

OrderReport Report = new OrderReport();

List<ZakazInfo> InfoList = new List<ZakazInfo>();

using (LD\_kursEntities db = new LD\_kursEntities())

{

var zakaz = db.zakaz.SingleOrDefault(c => c.id == id);

Report.OrderId = zakaz.id.ToString();

Report.start\_time = zakaz.start\_time;

Report.end\_time = zakaz.end\_time;

Report.DesignerName = zakaz.designer.name;

Report.DesignerSurname = zakaz.designer.surname;

Report.LandName = zakaz.land.name;

Report.LandAddress = zakaz.land.addres;

Report.LandSize = zakaz.land.size.ToString();

Report.CustomerName = zakaz.land.customer.name;

Report.CustomerSurname = zakaz.land.customer.surname;

Report.WorkList = Mapper.Map<List<Work>>(zakaz.work);

Report.DifficultiesList = Mapper.Map<List<Difficulties>>(zakaz.difficulties);

}

return Report;

}

public OrderCommonDate GetOrderReport(string time)

{

var Order = new OrderCommonDate();

using (LD\_kursEntities db = new LD\_kursEntities())

{

if (time == "3")

{

Order = db.Database.SqlQuery<OrderCommonDate>("OrderReportMonth").SingleOrDefault();

Order.Period = "Month";

}

else

if (time == "2")

{

Order = db.Database.SqlQuery<OrderCommonDate>("OrderReportYear").SingleOrDefault();

Order.Period = "Year";

}

else

{

Order = db.Database.SqlQuery<OrderCommonDate>("OrderReportAllTime").SingleOrDefault();

Order.Period = "All period";

}

}

return Order;

}

public OrderExtraReport GetExtraReport(int time, int size)

{

var Order = new OrderExtraReport();

using (LD\_kursEntities db = new LD\_kursEntities())

{

int LandSize = 0;

if (size == 3)

LandSize = 500;

if (size == 2)

LandSize = 1000;

var clientIdParameter = new SqlParameter("@minLandSize", LandSize);

if (time == 3)

{

Order = db.Database.SqlQuery<OrderExtraReport>("ExtraOrderReportMonth @minLandSize", clientIdParameter).SingleOrDefault();

Order.Period = "Month";

}

else

if (time == 2)

{

Order = db.Database.SqlQuery<OrderExtraReport>("ExtraOrderReportYear @minLandSize", clientIdParameter).SingleOrDefault();

Order.Period = "Year";

}

else

{

Order = db.Database.SqlQuery<OrderExtraReport>("ExtraOrderReport @minLandSize", clientIdParameter).SingleOrDefault();

Order.Period = "Year";

}

Order.LandVariable = LandSize;

}

return Order;

}

}

}

using KR.Business.Entities;

using KR.Business.ReportBilders;

using KR.Business.Repositories;

using KR.DbEF.Repositories;

using KR.Web\_.Models;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using System.Data.Entity;

using PagedList;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using KR.DbEF;

using KR.Business.ExelHelpers;

namespace KR.Web\_.Controllers

{

public class HomeController : Controller

{

private static IZakaz<Zakaz> ZakazRepositories;

private static IWork<Work> WorkRepositories;

private static IZakazInfo<ZakazInfo, zakaz> ZakazInfoRepositories;

private static ISingleOrder<SingleOrder> SingleOrderRepo;

private static IDifficulties<Difficulties> DiffiicultsRepo;

private static ExportRepositories ExportRepositories;

public HomeController(IZakaz<Zakaz> \_ZakazRepositories,

IWork<Work> \_WorkRepositories,

IZakazInfo<ZakazInfo, zakaz> \_ZakazInfoRepositories,

ISingleOrder<SingleOrder> \_SingleOrderRepo,

IDifficulties<Difficulties> \_DiffiicultsRepo)

{

ZakazRepositories = \_ZakazRepositories;

WorkRepositories = \_WorkRepositories;

ZakazInfoRepositories = \_ZakazInfoRepositories;

SingleOrderRepo = \_SingleOrderRepo;

DiffiicultsRepo = \_DiffiicultsRepo;

ExportRepositories = new ExportRepositories();

}

[Authorize]

public ActionResult Index(int? page,

string nameDesigner, string surnameDesigner,

string nameCustomer, string surnameCustomer,

int? OrderPrice,

string startDate, string endDate)

{

string prise = String.Empty;

if (String.IsNullOrEmpty(nameDesigner))

nameDesigner = String.Empty;

if (String.IsNullOrEmpty(surnameDesigner))

surnameDesigner = String.Empty;

if (String.IsNullOrEmpty(nameCustomer))

nameCustomer = String.Empty;

if (String.IsNullOrEmpty(surnameCustomer))

surnameCustomer = String.Empty;

if (OrderPrice != null)

prise = OrderPrice.Value.ToString();

if (String.IsNullOrEmpty(startDate))

startDate = String.Empty;

if (String.IsNullOrEmpty(endDate))

endDate = String.Empty;

var List = ZakazInfoRepositories.GetList(nameDesigner, surnameDesigner,

nameCustomer, surnameCustomer, prise, startDate, endDate);

if (List == null)

List = new List<ZakazInfo>();

int pageSize = 10;

int pageNumber = (page ?? 1);

return View(List.ToPagedList(pageNumber, pageSize));

}

[HttpGet]

public ActionResult Info(int id)

{

var zakaz = SingleOrderRepo.GetbyId(id);

if (zakaz == null)

return RedirectToAction("Index");

return View(zakaz);

}

[HttpGet]

[Authorize(Roles = "admin")]

public ActionResult Edit(int id)

{

var zakaz = ZakazRepositories.Get(id);

return View(zakaz);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "admin")]

public ActionResult Edit(Zakaz zakaz)

{

if (ModelState.IsValid)

{

var \_zakaz = ZakazRepositories.Edit(zakaz);

if (\_zakaz != null)

{

return RedirectToAction("Index");

}

}

return View(zakaz);

}

[HttpGet]

[Authorize(Roles = "admin")]

public ActionResult Add(int id)

{

var zakaz = new Zakaz();

zakaz.land\_id = id;

return View(zakaz);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "admin")]

public ActionResult Add(Zakaz zakaz)

{

if (ModelState.IsValid)

{

var \_zakaz = ZakazRepositories.Save(zakaz);

if (\_zakaz != null)

{

return RedirectToAction("Index");

}

}

return View(zakaz);

}

[HttpPost]

[Authorize(Roles = "admin")]

public JsonResult Delete(int id)

{

var zakaz = ZakazInfoRepositories.Delete(id);

if (zakaz == null)

Json(-1);

return Json(id);

}

[HttpPost]

[Authorize(Roles = "admin")]

public JsonResult AddDifficults(int orderId, int difficultsId)

{

var add = DiffiicultsRepo.Save(orderId, difficultsId);

return Json(add);

}

[HttpPost]

[Authorize(Roles = "admin")]

public JsonResult DeleteDifficults(int orderId, int difficultsId)

{

var add = DiffiicultsRepo.Remove(orderId, difficultsId);

if (add == null)

return Json(-1);

return Json(add.id);

}

[HttpPost]

[Authorize(Roles = "admin")]

public JsonResult AddWork(int orderId, Work work)

{

var add = WorkRepositories.Save(orderId, work);

return Json(add);

}

[HttpPost]

[Authorize(Roles = "admin")]

public JsonResult DeleteWork(int orderId, int workId)

{

var add = WorkRepositories.Remove(orderId, workId);

if (add == null)

return Json(-1);

return Json(add.id);

}

[Authorize(Roles = "admin")]

public ActionResult Reports()

{

return View();

}

[Authorize(Roles = "admin")]

public FileResult GetAvgOrderReport(string Time)

{

var report = ZakazInfoRepositories.GetOrderReport(Time);

var file = OrderDateReportBilders.GetReport(report);

return File(file, "application/pdf", "report.pdf");

}

[Authorize(Roles = "admin")]

public FileResult GetOrderReport(int id)

{

var report = ZakazInfoRepositories.GetReport(id);

var file = OrderReportBilders.GetReport(report);

return File(file, "application/pdf", "report.pdf");

}

[Authorize(Roles = "admin")]

public FileResult GetExtraOrderReport(int Time, int size)

{

var report = ZakazInfoRepositories.GetExtraReport(Time, size);

var file = ExtraOrderReportBilders.GetReport(report);

return File(file, "application/pdf", "report.pdf");

}

[Authorize(Roles = "admin")]

public FileResult GetExelDb()

{

var List = ExportRepositories.GetList();

var file = ExportToExel.Download(List);

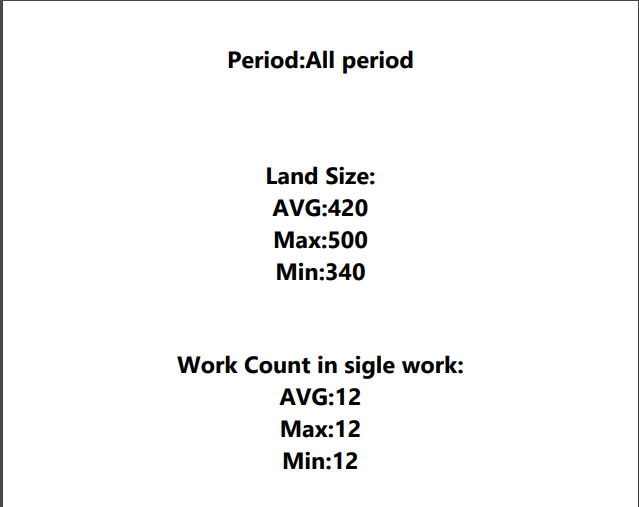
return File(file, "application/vnd.ms-excel", "export.xlsx");

}

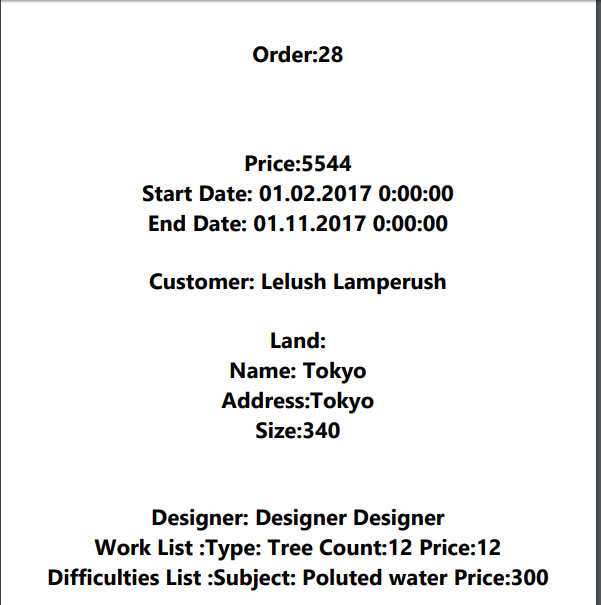
}

}

Звіт по кількості землі і кількості робіт в замовленнях:



Звіт по замовленню:



Звіт по замовленням: розмір землі:

