**РЕФЕРАТ**

Выпускная квалификационная работа с., рис., табл., \_\_\_\_\_\_\_\_\_\_источников, прил.

Ключевые слова: автоматизация бизнес-процессов, СУБД MySQL,

Объектом исследования являются методы, способы и инструменты для разработки и реализации современных информационных систем предназначенных для автоматизации бизнес-процессов.

Цель работы – автоматизация бизнес-процессов компании занимающейся проектной деятельностью по разработке и внедрению геоинженерных коммуникаций.

В процессе исследования проводились работы по разработке регламентов сотрудников компании, описанию пользовательских сценариев, внедрение программного продукта в компании, обучение сотрудников тестовой группы, доработка программного обеспечения на основе полученных отзывов сотрудников.

В результате выполнения работы разработана альфа-версия ИС, под рабочим название «Колибри», обеспечивающая взаимодействие сотрудников внутри отделов и прохождение заказа по этапам между отделами.

Основные конструктивные, технологические и технико-эксплуатационные характеристики: разработан удобный пользовательский интерфейс, разработан API и серверная часть, обеспечивающая, в масштабах организации, мгновенный доступ к данным. Во избежание потери времени на обработку задания, разработан механизм оповещения о переходе задачи в следующий статус.

Степень внедрения: разработанная информационная система внедрена для тестовой группы компании, проведено обучение сотрудников.

Область применения: разработанное в процессе выполнения работы программное обеспечение направлено на автоматизацию бизнес-процессов компании, занимающейся проектной работой по разработке и внедрению геоинженерных коммуникаций.

Экономическая эффективность/значимость работы: после внедрения информационной системы, повысился контроль над исполнением задач, согласно регламенту работы, что увеличило работоспособность компании.

В будущем планируется интегрировать информационную систему с Active Directory и 1С: Предприятием 8.0. Ввиду реализации API, возможна разработка web-интерфейса и мобильных приложений под iOS и Android. Создание инструментов отчетности и анализа.

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc530849346)

[ОБЗОР ТЕХНОЛОГИЙ И РЕШЕНИЙ 6](#_Toc530849347)

# ВВЕДЕНИЕ

В современном мире для повышения производительности труда используются различные инструменты, но ни одна из современных компаний не может обойтись без использования CRM. Обуславливается это тем, что человеческий ресурс эффективно используется при выполнении какой-либо одной решаемой в текущий момент времени задачи и дополнительные нагрузки на внимание, такие как напоминания о необходимости выполнения другой задачи, расстановка приоритетов выполнения задач и прочие отвлекающие факторы негативно сказываются на производительности. На много эффективнее, если специалисты работают над строго поставленной задачей – инструкцией.

Для обеспечения подобной работы в первую очередь необходимо строгое регламентирование бизнес-задач с указанием ответственности специалиста по срокам и качеству выполнения задачи. Регламентами возможно перекрыть много брешей бизнеса, но открытым остаётся вопрос о контроле выполнения задачи. Для этого необходима объективная оценка срока и качества. В этом вопросе человеческий ресурс не всегда эффективен, т.к. с человеком можно договориться, пойти навстречу, войти в положение и прочее, что, как правило, и является тем костылём бизнеса, из-за которого он хромает.

Наиболее эффективным решение задачи является внедрение в бизнес-процессы непредвзятой CRM-системы. Помимо контроля сроков и качества выполнения задач, на CRM возможно повесить функционал по предоставлению отчётов и аналитики.

С подобной задачей столкнулась молодая, развивающаяся компания ООО «Артания Гео». Основным продуктом деятельности компании является разработка и внедрение геоинженерных коммуникаций. На этапе зарождения компании, когда в ней, помимо Генерального директора и Главного бухгалтера, были пара бурильщиков-монтажников и конструктор-водитель было достаточно устных распоряжений и записок на стикерах. На данный момент в компании исполнительно (полевого) состава – более 20 человек. имеется группа проектировщиков, отдел закупок и отдел по работе с клиентами.

Контролировать устными договоренностями и записками такой штат уже становится невозможно. Из-за отсутствия должного контроля исполнения задач, компания выполняла много заказов, но не закрывала проекты, из-за отсутствия документации, которую должны были сделать, но по каким-то причинам исполнители переключились на другие задачи, отложив в долгий ящик задачи, являющиеся финальными для осуществления расчётов. После анализа незакрытых проектов, выяснилось, что компания, из-за упущенной прибыли, переходит в период стагнации, несмотря на то что за несколько лет до этого была признана лучшим стартапом Томска.

Данная работа нацелена на автоматизацию бизнес-процессов компании посредством внедрения CRM-системы, обеспечивающей соблюдения регламентов работы между отделами и внутри отделов. Внедрение системы автоматизирует переход задачи между подразделениями и позволит контролировать жизненный цикл с момента поступления запроса от клиента до завершения проекта. Основные цели внедрения – оптимизация и контроль работы сотрудников компании.

Основываясь на современные тенденции разработки информационных систем, сформулированы следующие требования к ИС:

1. ИС должна быть кроссплатформенной
2. Работа с данными должна быть реализована посредством API
3. Архитектура ИС должна быть многоуровневой.

# ОБЗОР ТЕХНОЛОГИЙ И РЕШЕНИЙ

Разработка многоуровневой ИС «с нуля» под бизнес-задачи заказчика – не тривиальная задача. Как и любой процесс, всё должно начинаться с планирования, и ничего не описывает лучше необходимость планирования, как крылатая фраза: «Провал в планировании подобен планированию провала». Для планирования проекта, я пользовался методологией гибкой разработки SCRUM.

Методология Scrum представляет собой итеративный процесс разработки программного обеспечения. При такой разработке для программного продукта создается много последовательных выпусков, в которых постепенно добавляется требуемая функциональность. Итеративный подход позволяет по завершению текущего итерации продемонстрировать заказчику работоспособный программный продукт, возможно с ограниченной функциональностью, получить отзыв, замечания и дополнительные требования, которые будут учтены в следующих итерациях. Основными артефактами в методологии Scrum являются рабочие элементы, отчеты, книги и панели мониторинга. Для SCRUM определены следующие роли в команде: владелец продукта (Product owner), руководитель (ScrumMaster); члены команды (Team members). Ввиду отсутствия полноценной команды, все роли исполнялись единолично, что представляло большие сложности в плане контроля выполнения сроков.

Согласно методологии жизненный цикл разработки представляется как:



Спринт - набор задач, запланированных на выполнение определенный период времени, в методологии Scrum. Первым самым важным спринтом стал – написание ТЗ.

До того, как взяться за разработку ПО, необходимо убедиться, что есть досконально разработанный регламент работы сотрудников и бизнес-процессов, в противном случае, необходимо разработать регламенты, т.е. документы, которые перечисляют и описывают по порядку этапы (шаги), которые должна предпринимать группа участников для выполнения бизнес-процессов. В общем виде регламенты имеют структуру:

Наименование

Содержание

1. Общие положения

1.1. Назначение

1.2. Область применения

1.3. Нормативные ссылки

1.4. Порядок утверждения, внесения изменений и дополнений

2. Термины, определения и сокращения

2.1. Термины и определения

2.2. Сокращения

3. Описание требований, процессов, методов работы

4. Ответственность

5. Контроль

6. Приложения

6.1. Классификаторы и справочники

6.2. Формы и правила оформления документов

6.3. Схемы процессов

6.4. Алгоритмы расчетов

6.5. Таблицы ролей исполнителей процессов

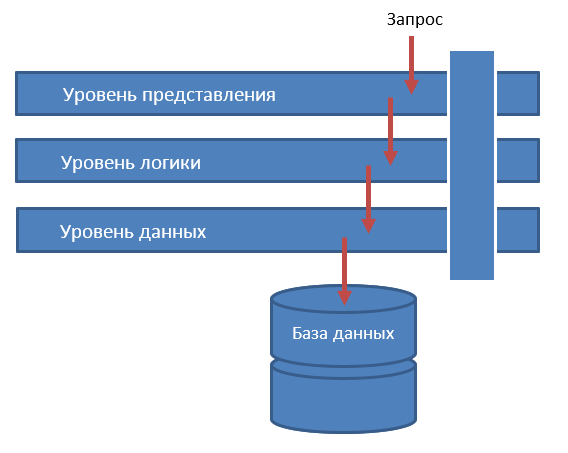
По своей сути, детально разработанный регламент – основа для написания ТЗ, что было и принято в нашем случае. Формализованное ТЗ максимально схоже с регламентами, поэтому части разработанных регламентов: USE CASE-диаграммы, целые блокои инструкций были без изменения транслированы в ТЗ.

В первую очередь, ТЗ разрабатывается для удовлетворения потребностей заказчика и защиты его интересов. Это документ. Согласно которому можно требовать от исполнителя выполнения задач, он определяет границы выполнения работы. ТЗ служит универсальным языком общения между заказчиком и разработчиком. ТЗ, в свою очередь, разрабатывалось по стандарту IEEE 29148-2011 «Разработка систем и программного обеспечения - Процессы жизненного цикла - Требования к проектированию» (пер. от - «Systems and software engineering. Life cycle processes. Requirements engineering.») с использованием шаблона спецификации требований: SRS.

SRS — Software Requirement Specification — специальная документация для ПО которая содержит в себе информацию о том, как должна себя вести система, какие функции должна выполнять, какую нагрузку должна выдерживать и тд.

Т.к. одно из требований к ИС было обеспечение многоуровневой архитектуры, т.е. клиент-серверная архитектура, в которой разделяются функции представления, обработки и хранения данных.

Многоуровневая архитектура является одной из самых известных архитектур, в которой каждый слой выполняет определенную функцию. В зависимости от ваших нужд вы можете реализовать любое количество уровней, но слишком большое их количество приведет к чрезмерному усложнению системы. Часто выделяют три основных уровня: уровень представления, уровень логики и уровень данных.



Слою не обязательно знать, что делают его соседи. Здесь проявляется такое свойство как разделение ответственности. Если все три слоя являются закрытыми, то запрос пользователя к верхнему уровню инициирует цепочку обращений с верхнего уровня до самого нижнего. В этом случае уровень представления отвечает за пользовательский интерфейс и отображение данных для пользователя и ничего не знает о существовании физического хранилища данных. Ничего о существовании базы данных не знает и уровень логики – его «беспокоят» только правила бизнес-логики. Доступ к базе данных имеет лишь через уровень управления данными.

Достоинствами применения такой архитектуры являются простота разработки (в основном из-за того, что этот вид архитектуры всем знаком) и простота тестирования.

Одним из логичных требований для построения ИС является реализация API. API (программный интерфейс приложения) (англ. application programming interface) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. Посредством API реализовано взаимодействие уровня представления с уровнем логики, что даёт возможность написания различных клиентов для пользования ИС. В рамках данной работы создано настольное клиентское приложение под ОС Windows, но реализация API даёт возможность создавать мобильные и настольные приложения под различные платформы, WEB-приложение и создание WEB-портала без изменений уровня логики и уровня данных.

FrameWork 4.7.2.

MySQL.

C#

# ПРОЕКТИРОВАНИЕ

## Постановка задачи

## Критерии оценки выполнения

# **III Практическая глава/главы**

## Реализация требований

## Создание БД

## Создание API

Данный слой – это уровень доступа к данным. Этот уровень обычно содержит все модели данных, хранящихся в БД, а также классы, через которые идет взаимодействие с БД. Все модели, объекты которых храниться в БД находятся в папке Entities с пространством имен SystemResource.DAL.Entities, здесь содержаться следующее модели:

1. Address – модель содержащая связи сущностей: город, улица.
2. City – модель информации о городе.
3. StreetAndBuilding – модель улицы и строения.
4. ApplicationRole – модель роли пользователя.
5. IdentityModel – вспомогательная модель для переопределения моделей ASP.NET Identity.
6. Company – модель предприятия.
7. CompanyTypeDevice – модель привязки предприятия к типу приборов.
8. CompanyUsers – модель привязки пользователь к компании.
9. DescriptionCompany – модель, хранящая дополнительную информацию о предприятии.
10. Device – модель, хранящая информацию о приборе.
11. FixedDeviceUser – модель привязки прибора к пользователю.
12. JournalDeviceRecords – модель, хранящая показатели или данные полученные с приборов.
13. Placement – модель, хранящая информацию о приборе.
14. TypeCompany – модель типа предприятия.
15. TypeDevice – модель типа приборов.

В пространстве имен *SystemResource.DAL.Infrastructure* имеется класс *OperationDetails.cs* для хранения информации о завершении операций на данном слое (как успешных, так и неуспешных). Этот класс имеет конструктор, который принимает начальные данные для свойств «сообщение» и «результата выполнения операции». Исходный код представлен в листинге 1.

**public** **class** OperationDetails

{

**public** OperationDetails(**bool** succedeed, **string** message)

{

Succedeed = succedeed;

Message = message;

}

**public** **bool** Succedeed { **get**; **private** **set**; }

**public** **string** Message { **get**; **private** **set**; }

}

Листинг 1 – Класс хранения результатов выполнения операций

В пространстве имен *SystemResource.DAL.EF* находится класс ApplicationContext или класс контекста данных для работы Entity Framework. У данного класса контекста данных в конструкторе есть принимаемый параметр типа string, в который требуется предать строку подключения к БД.

Для увеличения гибкости подключения к БД используются репозитории, поэтому в пространстве *SystemResource.DAL.Interfaces* есть общий интерфейс репозиториев *IRepository* представленный в листинге 2.

**public** **interface** IRepository<T, T1> : IDisposable

**where** T : **class**

**where** T1 : Infrastructure.OperationDetails

{

Task<Tuple<IEnumerable<T>, T1>> GetAll();

Task<Tuple<T, T1>> GetId(**int** id);

Task<Tuple<IEnumerable<T>, T1>> Find(Expression<Func<T, Boolean>> predicate);

Task<T1> Create(T item);

Task<T1> Update(T item);

Task<T1> Delete(**int** id);

}

Листинг 2 – Общий интерфейс репозиториев

Поскольку используется несколько репозиториев для каждой сущности, то для упрощения подключения к БД будет применен паттерн Unit Of Work. И для этого также в пространстве имен *SystemResource.DAL.Interfaces* создан интерфейс *IUnitOfWork* представленный в листинге 3.

**public** **interface** IUnitOfWork : IDisposable

{

ApplicationUserManager UserManager { **get**; }

ApplicationRoleManager RoleManager { **get**; }

IRepository<Company, Infrastructure.OperationDetails> CompanyManager { **get**; }

IRepository<TypeCompany, Infrastructure.OperationDetails> TypeCompanyManager { **get**; }

IRepository<DescriptionCompany, Infrastructure.OperationDetails> DescriptionCompanyManager { **get**; }

IRepository<City, Infrastructure.OperationDetails> CityManager { **get**; }

IRepository<StreetAndBuilding, Infrastructure.OperationDetails> StreetAndBuildingManager { **get**; }

IRepository<Address, Infrastructure.OperationDetails> AddressManager { **get**; }

IRepository<TypeDevice, Infrastructure.OperationDetails> TypeDeviceManager { **get**; }

IRepository<CompanyTypeDevice, Infrastructure.OperationDetails> CompanyTypeDeviceManager { **get**; }

IRepository<CompanyUsers, Infrastructure.OperationDetails> CompanyUsersManager { **get**; }

IRepository<FixedDeviceUser, Infrastructure.OperationDetails> FixedDeviceUserManager { **get**; }

IRepository<JournalDeviceRecords, Infrastructure.OperationDetails> JournalDeviceRecordsManager { **get**; }

IRepository<Device, Infrastructure.OperationDetails> DeviceManager { **get**; }

IRepository<InstallerData, Infrastructure.OperationDetails> InstallerDataMeneger { **get**; }

IRepository<Placement, Infrastructure.OperationDetails> PlacementManager { **get**; }

Task<Infrastructure.OperationDetails> SaveAsync();

}

Листинг 3 – Интерфейс IUnotOfWork

Для хранения реализации данных интерфейсов в пространстве имен *SystemResource.DAL.Repositories* и *SystemResource.DAL.Identity* реализованы класс реализации общего репозитория и классы для работы с моделями Identity:

1. ApplicationRoleManager.cs – реализация репозиторрия для работы с ролями пользователей системы.
2. ApplicationUserManager.cs – реализация репозитория для работы с пользователями системы.
3. UnitOfWork.cs – реализация интерфейса IUnitOfWork в конструкторе принимающего строку подключения, которая потом будет передаваться в конструктор контекста данных.
4. EFRepository.cs - реализация интерфейса IRepository в конструкторе принимающего строку подключения, которая потом будет передаваться в конструктор контекста данных.

Через UnitOfWork будет происходить взаимодействие с базой данных.

Реализация схемы базы данных, разворачивающаяся при первоначальной настройке и инициализации программы, представлена на рисунке 4.

## Создание серверной части ИС

Бизнес-уровень инкапсулирует всю бизнес-логику, все необходимые вычисления, получает объекты из уровня доступа к данным и передает их на уровень представления, либо получает данные с уровня представления и передает их на уровень данных. Поскольку бизнес-уровень будет использовать классы из уровня доступа к данным, то этот слой имеет ссылку на слой Data Access Layer.

Уровень представления не может напрямую получать данные из базы данных. В данном случае BLL будет выступать в роли посредника между двумя уровнями. Но также надо учитывать, что напрямую он не может передавать в контроллеры объекты, так как уровень представления не должен иметь доступ к функциональности уровня DAL. Поэтому в пространстве имен *SystemResource.BLL.DTO* имеются промежуточные сущности. Через них предаются объекты между уровнями. Хотя данные классы во многом похожи по определению на классы Entities в DAL, это необязательное условие. Классы DTO должны содержать только те данные, которые передаются на уровень представления или принимаются с этого уровня. То есть это то, что называется Data Transfer Object – специальная модель для передачи данных.

Кроме простых классов типа DTO, BLL может содержать классы, которые описывают бизнес-логику. Большую роль в приложении играет валидация данных. По большей части за валидацию отвечает именно BLL. В контроллере легко можно провалидировать модель через объект ModelState и при необходимости возвратить в представление сообщения об ошибках. Но на уровне BLL ModelState недоступен. Однако, все же можно использовать валидацию с передачей ошибок в уровень представления через специальный класс OperationDetails, который находится в пространстве имен *SystemResource.BLL.Infrastructure –* через данный класс будет передаваться информация об успешности операции.

Взаимодействие между уровнем представлений и данным уровнем происходит через специальные сервисы. Для большей гибкости в пространстве имен *SystemResource.BLL.Interfaces* содержатся интерфейсы реализации сервисов.

Для хранения реализаций интерфейсов в пространстве имен *SystemResource*.BLL.Services содержатся классы сервисов. Все сервисы при помощи конструкторов принимают объект IUnitOfWork, через который идет взаимодействие с уровнем DAL. Поскольку в данном случае не задается в конструкторах объект IUnitOfWork, то требуется использовать внедрение зависимостей для передачи конкретной реализации данного интерфейса в сервисы. Для этого используется Ninject, а именно в пространстве имен *SystemResource.BLL.Infrastructure* содержится класс ServiceModule, представляющий специальный модуль Ninject, который служит для организации сопоставления зависимостей. Он устанавливает использование UnitOfWork в качестве объекта IUnitOfWork. Кроме того, здесь через конструктор передается строка подключения, которая в итоге будет определяться в файле web.config проекта, представляющего уровень представления. Исходный код данного класса представлен в лист. 4.

**public** **class** ServiceModule : NinjectModule

{

**private** **string** connectionString;

**public** ServiceModule(**string** connection )

{

connectionString = connection;

}

**public** **override** **void** Load()

{

Bind<IUnitOfWork>().To<UnitOfWork>().WithConstructorArgument("connectionString", connectionString);

}

}

## Создание клиентского приложения

## Документирование

## ПСИ

# **Расчет БЖД**

# **Экономическое обоснование**

# **ЗАКЛЮЧЕНИЕ**

# **CONCLUSION**

# **СПИСОК ЛИТЕРАТУРЫ**