МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированные технологии разработки программного обеспечения»

Тема: ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА

Студенты гр. 4303, 4304 _____ Гордеева Т.В.

_____ Дронников И.М.

_____ Полушина Ю.И.

_____ Козловский А.О.

Преподаватель _____ Спицын А.В.

Санкт-Петербург
2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студенты Гордеева Т.В., Полушина Ю.И., Дронников И.М, Козловский А.О.

Группы 4303, 4304

Тема работы: Объектно-ориентированная разработка программного комплекса «Post Delivery Logistics»

Исходные данные:

На языке C++ реализовать:

- шаблон полиморфного контейнера (параллельно-последовательный граф) на основе STL-контейнера, использовать шаблоны проектирования;
- систему классов исключительных ситуаций, стратегию контроля аномального поведения программы;
- сериализацию, манипулятор, аллокатор;
- графический интерфейс, вызуализацию приложения.
- Разработать набор диаграмм UML, диаграмму Ганта, отчёт по рефакторингу, документацию по тестированию, требуемые артефакты RUP.

Содержание пояснительной записки:

«Содержание», «Введение», «Заключение», «Список использованных источников», «Приложение»

Предполагаемый объем пояснительной записки: Не менее 10 страниц.

Дата выдачи задания: 07.02.2019

Дата сдачи реферата: 06.06.2019

Дата защиты реферата: 06.06.2019

Студенты гр. 4303, 4304 _____ Гордеева Т.В.

_____ Дронников И.М.

_____ Полушина Ю.И.

_____ Козловский А.О.

Преподаватель _____ Спицын А.В.

АННОТАЦИЯ

В данной курсовой работе разработан программный комплекс «Post Delivery Logistics», который решает задачу перевозки почтовых бизнес-отправлений. Разработка велась на основе объектно-ориентированного подхода. Реализация полиморфного контейнера осуществлялась на языке программирования C++.

Разработка программного комплекса производилась в соответствии с методологией RUP. Для каждой итерации разработан соответствующий пакет документов (RUP-артефактов).

В ходе работы над проектом были разработаны следующие компоненты системы: полиморфный контейнер для хранения последовательно-параллельного графа, графический интерфейс пользователя, аллокатор памяти, алгоритм обхода.

SUMMARY

In this term paper software package "Post Delivery Logistics", which solves the problem of transportation of postal business items, has been developed. The development was conducted on the basis of an object-oriented approach. The implementation of the polymorphic container was implemented in the C ++ programming language.

The development of the software package was carried out in accordance with the RUP methodology. For each iteration, a corresponding package of documents (RUP artifacts) has been developed.

During the work on the project, the following system components were developed: polymorphic container for storing a series-parallel graph, graphical user interface, memory allocator, bypass algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данного проекта – разработка программного комплекса PostDeliveryLogistics, выполняющего функции логистического сервиса для почтовых перевозок.

Задачи проекта – разработать на языке программирования C++ программную систему, в рамках которой реализуется полиморфный контейнер. Реализуемый контейнер хранит параллельно-последовательный граф.

Разработка ведется в соответствии с методологией RUP. Процесс разработки на каждой итерации сопровождается соответствующим набором документов. Требуется проведение тестирования ПО с написанием документации.

При работе над проектом были использованы знания из следующих ранее изученных дисциплин: Объектно-ориентированное программирование, Алгоритмы и структуры данных, Проектирование человеко-машинного интерфейса, Тестирование ПО.

1. АНАЛИЗ И ПРОЕКТИРОВАНИЕ СИСТЕМЫ ЛОГИСТИКИ ПОЧТОВЫХ ОТПРАВЛЕНИЙ

1.1. О проекте

Целью проекта является разработка программного продукта для наглядного отображения порядка процесса доставки продукции по горячим точкам. И проверки осуществляемости доставки. В качестве контейнера используется параллельно последовательный граф. Разработка ведется в соответствии с методологией RUP. Процесс разработки на каждой итерации сопровождается соответствующим набором документов. Требуется проведение тестирования разрабатываемого ПО с написанием документации.

Таблица 1 – Цель проекта

Проблема	Имея множество, заказов сложно в них ориентироваться, а также сложно оптимально составить порядок доставки заказов.
Затрагивает	Работников службы доставки.
Влияет	На время, затрачиваемое на составления порядка доставки товаров.
Успешное решение	Упростит контроль заказов на доставку, позволит оптимизировать порядок выполнения доставки.

Продукт подходит для служб доставки товаров, в которых имеется множество заказов на доставку при ограниченном количестве адресов доставки.

1.2. Структура проекта

Варианты использования в системе перечислены ниже:

- Сохранение в файл.
- Загрузка из файла.
- Просмотр графа компонентов.
- Добавление компонента.

- Удаление компонента.
- Обновление.

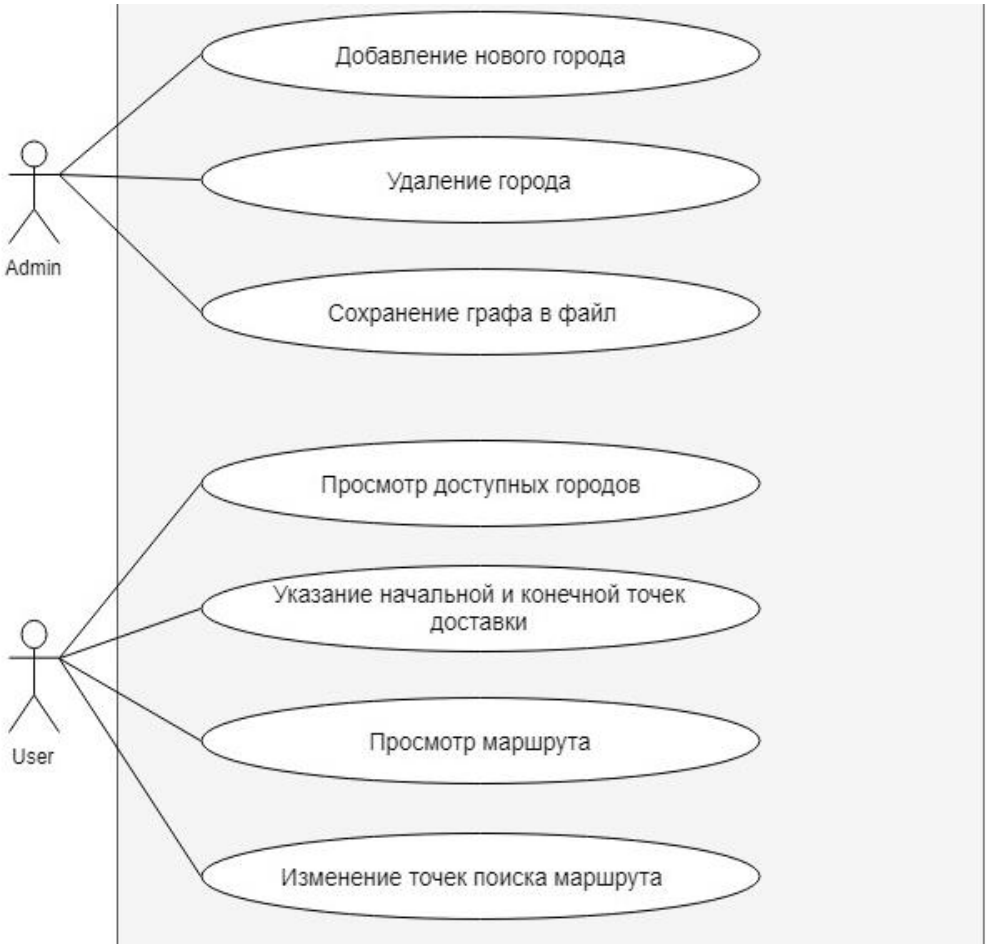


Рисунок 1 – Диаграмма вариантов использования

Логическое представление системы Post Delivery Logistics состоит из 3-х основных пакетов

- GraphWidget

Предоставляет возможности для отрисовки графа на интерфейсе.

- MainWindow

Предоставляет интерфейс пользователю и реализует основные алгоритмы программы.

- Graph

Предоставляет контейнер для хранения графа и возможности работы с контейнером. Является наиболее значимым пакетом.

В соответствии с диаграммой использования построена диаграмма состояний системы, представленная на рисунке 2.

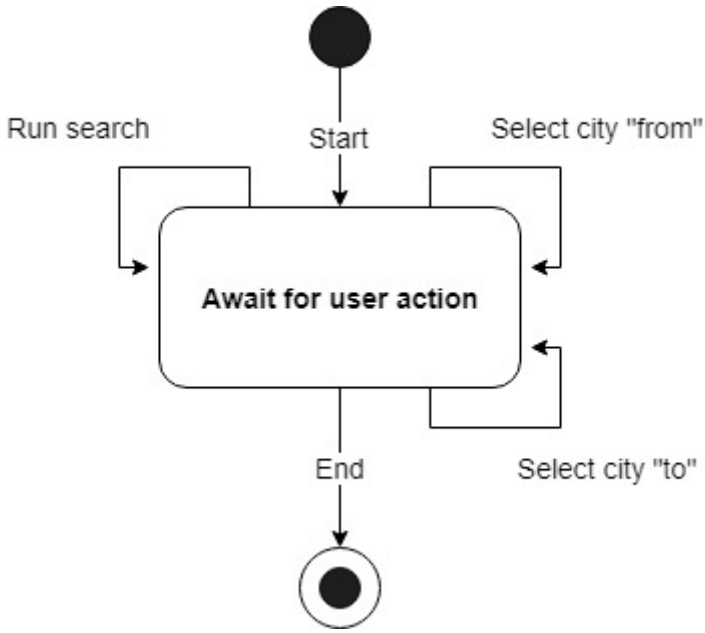


Рисунок 2 – Диаграмма состояний системы

1.3 Архитектура программного обеспечения

Архитектура представлена в виде набора представлений: представление прецедентов, логическое представление и представление реализации. Эти представления описаны как модели и используют унифицированный язык моделирования (UML).

Цели и ограничения

Несколько основных требований и системных ограничений, которые значительно влияют на архитектуру:

- Кроссплатформенность. Система должна работать на различных машинах и операционных системах.
- Все функции должны быть доступны и понятны пользователю.
- Обязательное наличие контейнера Параллельно-последовательный граф с использованием STL.
- Контейнер должен использовать собственный аллокатор, манипулятор и поддерживать собственные исключения.

Представление прецедентов

Варианты использования в системе перечислены ниже. Варианты, выделенные *Курсивом,* важны для архитектуры и влияют на ее описание:

- *Сохранение в файл*
- *Загрузка из файла*
- Просмотр графа
- *Добавление компонентов*
- *Удаление компонента*
- *Обновление*

Диаграмму и описание прецедентов можно найти в Спецификации прецедентов.

Логическое представление

Описание

Логическое представление системы Post Delivery Logistics состоит из 2-х основных пакетов:

- **MainWindow**

Предоставляет интерфейс и реализует основные алгоритмы программы.

- **Graph**

Предоставляет контейнер для хранения графа и возможности работы с контейнером. Является наиболее значимым пакетом.

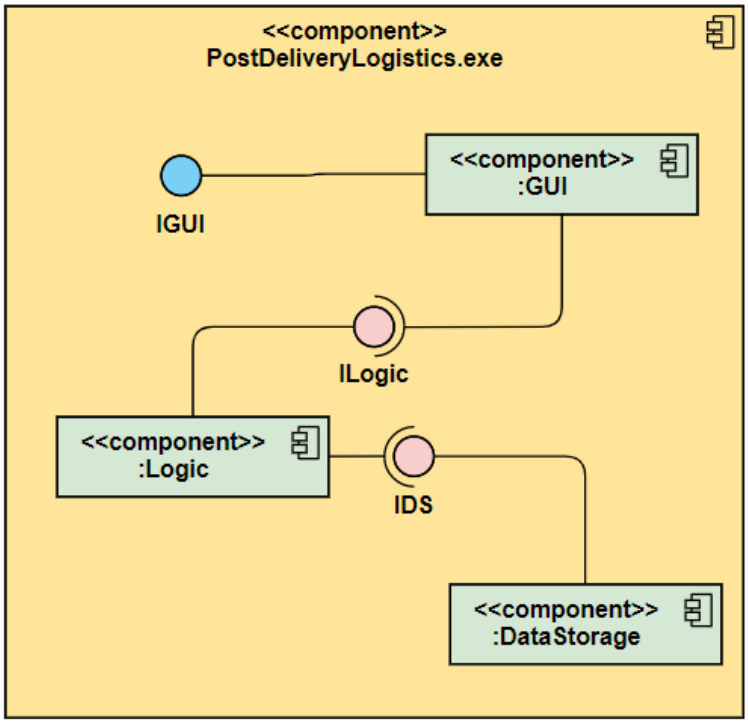


Рисунок 3 – Диаграмма компонентов

Значимые пакеты архитектуры

Graph

Пакет Graph реализует контейнер Параллельно-последовательное дерево (класс graph), который является шаблоном, содержащим в себе список вершин. Для его разработки используются такие шаблоны проектирования как Пул объектов и Итератор. Пул объектов представлен классом Allocator, предоставляющим интерфейс для выделения и освобождения памяти для объектов контейнера. Итератор представлен классом Iterator и предоставляет возможности для получения доступа к элементам контейнера. Используется для обхода графа по вершинам.

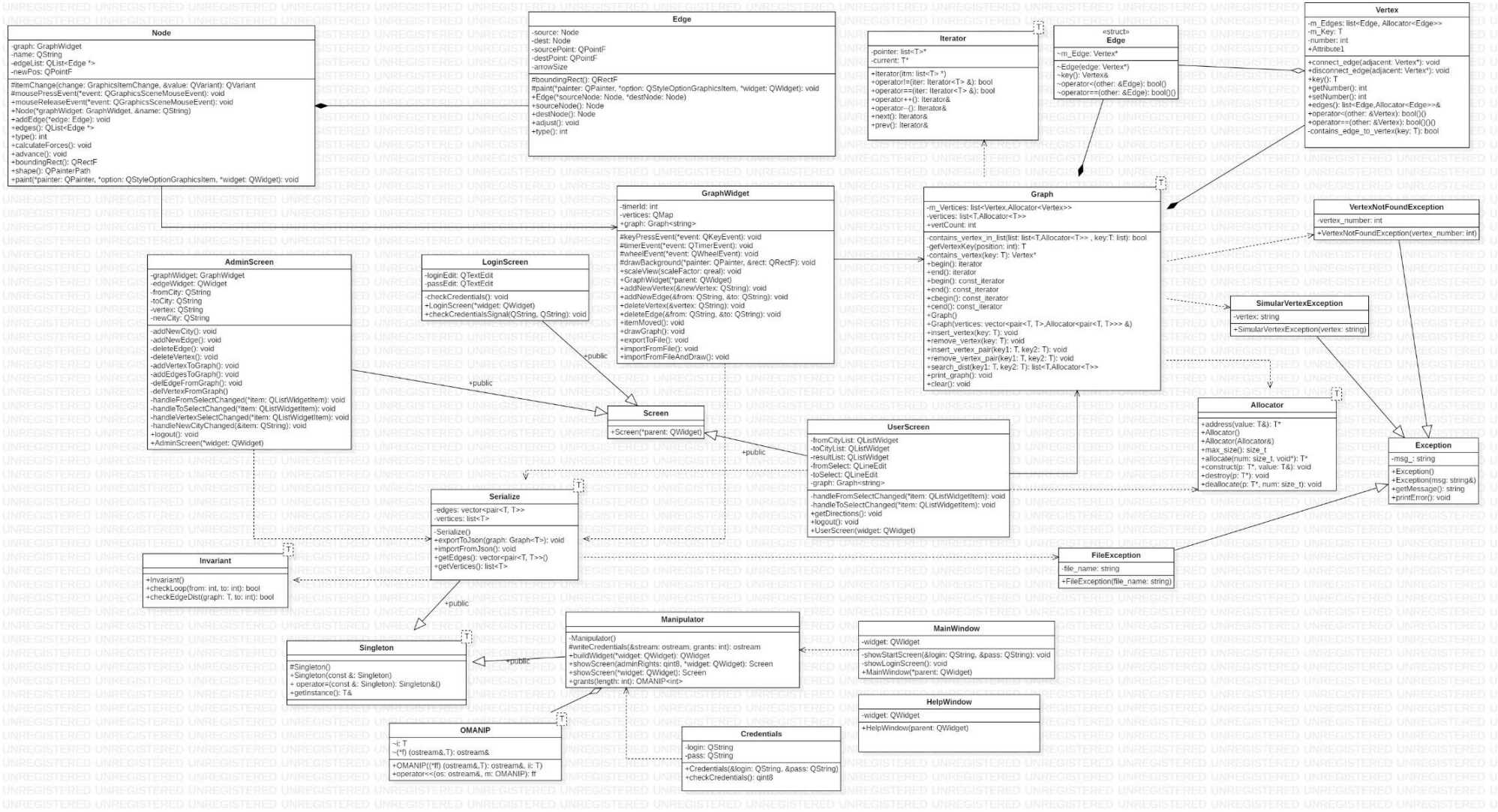


Рисунок 4 – Диаграмма классов

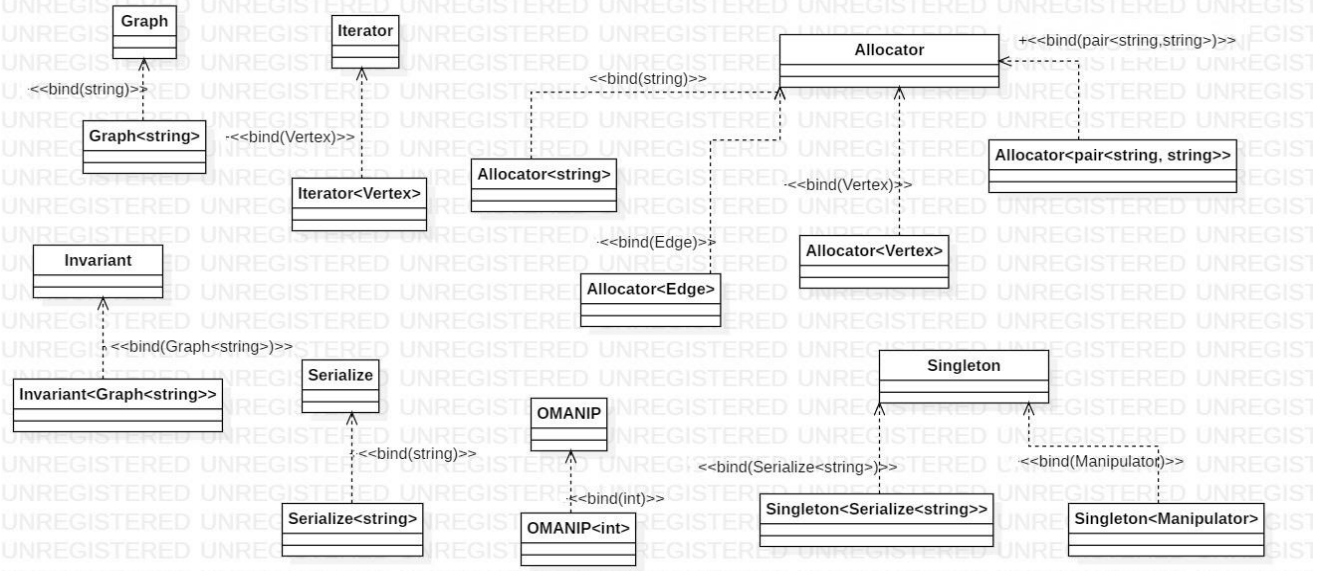


Рисунок 5 – Диаграмма классов, развернутых из шаблонных

Представление реализации

Описание

Система реализована как единая исполняемая программа. Саму систему можно разделить на 3 слоя: слой интерфейса, слой контейнера и внешний слой.

Слои

- **Слой интерфейса**

Набор классов, реализующих интерфейс и функции активных элементов**.**

- **Слой контейнера**

Набор шаблонных классов, реализующих функции контейнера graph.

- **Внешний слой**

Набор классов и возможностей, полученных из внешних библиотек.

Производительность

Качество

Система предоставляет понятный пользовательский интерфейс, согласно существующим графическим стандартам. Прототип интерфейса показан в документе Прототип интерфейса.

2. РЕАЛИЗАЦИЯ СИСТЕМЫ КОНФИГУРАЦИИ КОМПОНЕНТОВ ПК

Система реализована как единая исполняемая программа. Саму систему можно разделить на 3 слоя: слой интерфейса, слой контейнера и внешний слой.

Слой интерфейса

● Набор классов, реализующих интерфейс и функции активных элементов.

Слой контейнера

● Набор шаблонных классов, реализующих функции контейнера Graph.

Внешний слой

● Набор классов и возможностей, полученных из внешних библиотек.

Требования к реализации:

1. Кроссплатформенность. Система должна работать на различных машинах и операционных системах.

2. Все функции должны быть доступны и понятны пользователю.

3. Обязательное наличие контейнера и/или дерево с использованием библиотеки STL.

4. Контейнер должен использовать собственный аллокатор, итератор, манипулятор и поддерживать собственные исключения.

2.1. Использованные технологии

Git – распределенная система управления версиями;

GitHab – веб-сервис для хостинга IT-проектов и их совместной разработки;

Graphviz – пакет утилит по автоматической визуализации графов, заданных в виде описания на языке DOT.

Google Disk – облачный сервис хранения данных;

STL – библиотека согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++;

Qt – кроссплатформенный фреймворк для разработки ПО на языке программирования C++. Далее подробнее об использованных частях, входящих в Qt:

QtCreator – кроссплатформенный свободный IDE для разработки на C, C++ и QML;

QMake – утилита облегчающая процесс сборки кроссплатформенных приложений;

Использованные библиотеки:

QtCore – классы ядра библиотеки, используемые другими модулями;

QtGui – компоненты графического интерфейса;

QtWidgets – содержит классы для классических приложений на основе виджетов;

2.2. Шаблоны проектирования

В программном продукте для отделения графического интерфейса от бизнеса логики, и бизнес логику от данных был использован шаблон проектирования Model-View-Controller (MVC). Схема шаблона проектирования MVC на примере приложения представлена на рис. 6:

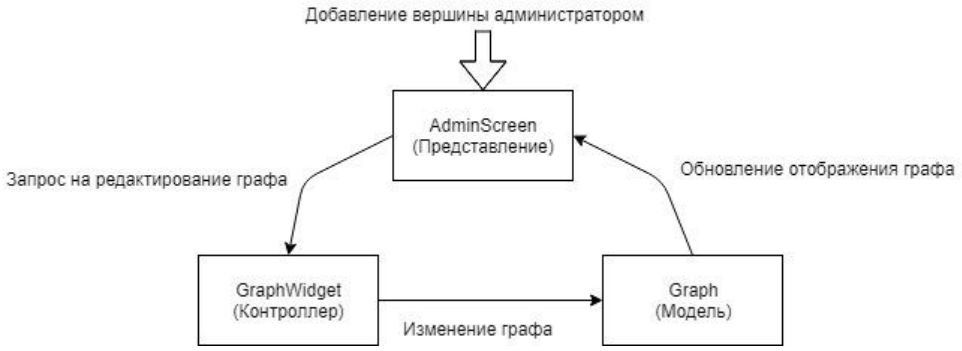


Рисунок 6 – Схема шаблона проектирования MVC

Для поддержки некоторых сущностей в единственном экземпляре (сериалайзер, манипулятор) был использован шаблон проектирования Singleton.

Схема шаблона проектирования MVC на примере приложения представлена на рис. 7:

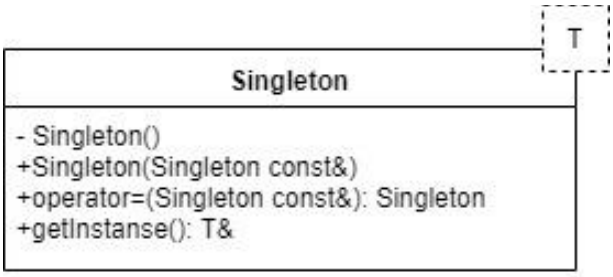


Рисунок 7 – Схема шаблона проектирования Singleton

2.3. Реализация требований

2.3.1. Реализация шаблона полиморфного контейнера (параллельно-последовательный граф), построенного на основе STL-контейнера

В теории графов параллельно-последовательные графы — это графы с двумя различными вершинами, которые называются терминальными, образованные рекурсивно с помощью двух простых операций.

2.3.2. Реализация системы классов исключительных ситуаций

В разработанной системе реализована обработка следующих исключительных ситуаций:

- Exception - базовый класс в иерархии исключений.
- FileException - исключение, порождаемое в случае ошибки открытия файла для чтения или записи, содержит в себе название файла. Наследник класса Exception.
- SimularVertexException - исключение, порождаемое в случае добавления уже существующей вершины в граф, содержит в себе название вершины. Наследник класса Exception.

• VertexNotFoundException - исключение, порождаемое в случае, когда не удается найти определенную вершину в графе, содержит в себе индекс вершины. Наследник класса Exception.

2.3.3. Реализация сериализации и манипулятора

Сериализация происходит с помощью QJsonObject, очередь сериализации:

- Идентификаторы вершин,
- вершины,
- дуги.

Сериализация происходит в специальный файл с расширением “.json”.

2.3.4. Реализация аллокатора

Разработан класс Allocator с использованием порождающего шаблона проектирования "объектный пул", хранит указатели на всю выделенную память, при очищении или разрешении контейнера, безопасно возвращает всю зарезервированную память. Схема шаблона проектирования "объектный пул" на примере приложения представлена на рис. 9:

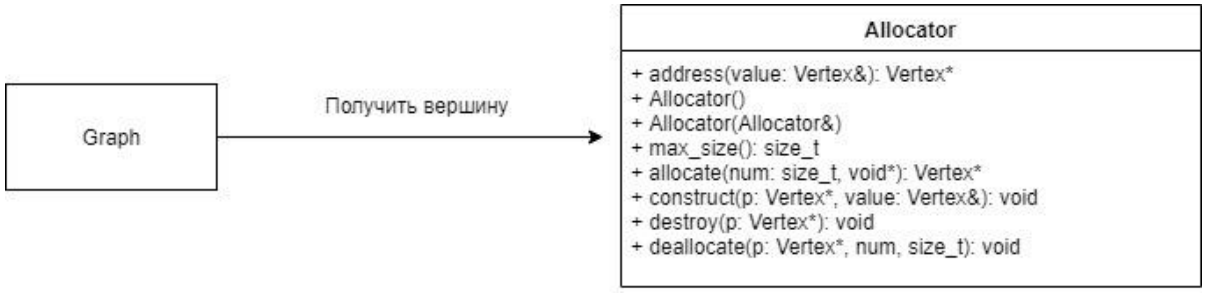


Рисунок 8 – Схема шаблона проектирования Object Pool

2.3.5. Реализация проверки инвариантов контейнера предусловий и постусловий для контейнера

Для контейнера проверяются два инварианта:

1. Для всех рёбер графа проверяется, что вершина куда входит ребро не является исходящей вершиной.

2. Для каждой вершины в графе, просматриваются все исходящие дуги, проверяется, чтобы вторая вершина на конце дуги, принадлежала графу.

ЗАКЛЮЧЕНИЕ

В ходе работы над проектом достигнуты следующие результаты:

- Проведено проектирование и реализована программная система на C++ с использованием RUP-методики и шаблонов проектирования.
- Приобретен опыт работы в команде.
- Изучены и закреплены навыки объектно-ориентированной разработки, разработки человеко-машинного интерфейса, конструирования и тестирования ПО.

Результаты работы полностью соответствуют указанным целям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика использования C++.: Пер. с англ. – М. : ООО «И.Д. Вильямс», 2011. – 1248 с.

2. Кратчен Ф. Введение в Rational Untified Process. 2-е изд. – М. : Вильямс, 2002. – 240 с.

3. Буч Г., Рамбо Дж., Якобсон А. Язык UML. Руководство пользователя – 2-е изд. – М., СПб. : ДМК Пресс, Питер, 2004. – 432 с. 6. Фаулер М. Рефакторинг: улучшение существующего кода. – пер. с англ. – СПб: Символ-Плюс, 2003. – 432 с.

4. Мейерс С. Эффективное использование STL // Библиотека программиста. СПб.: Питер. – 2002.

5. Фаулер М. UML Основы. СПб.: Символ-Плюс, 2004. 192 с.

6. Кролл П. Кратчен Ф. Rational Untified Process – это легко. Руководство по RUP. Пер. с англ. – М. : КУДИЦ-ОБРАЗ, 2004. – 432 с.

7. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес; [пер. с англ.: А. Слинкин науч. Ред.: Н. Шалаев]. Санкт-Петербург [и др.]: Питер, 2014. 366 с.

8. Седжвик Роберт. Фундаментальные алгоритмы на C++. Части 1-4. Анализ. Структуры данных. Сортировка. Поиск - 3-я редакция. — Пер. с англ. — Киев: ДиаСофт, 2001. — 688 с. — ISBN 966-7393-89-5.

ПРИЛОЖЕНИЕ А. АРТЕФАКТЫ НАЧАЛЬНОГО ЭТАПА

Post Delivery Logistics

Глоссарий

История изменений

Дата	Версия	Описание	Автор
01.03.19	1.0	Создание документа	Козловский А.О.
04.03.19	1.1	Добавление новых сокращений	Гордеева Т.В Дронников И.М. Полушина Ю.И. Козловский А.О.

Введение

Данный документ содержит определения терминов в системе подбора персонала. Документ будет расширяться на протяжении всего жизненного цикла проекта. Любые определения, не включенные в данный документ, могут быть включены.

Глоссарий

GUI

Графический пользовательский интерфейс - разновидность интерфейса, в котором элементы интерфейса исполнены в виде графических изображений.

RUP

Rational Unified Process - процесс, направленный на коллективную разработку ПС.

Use-cases

Сценарии использования - описание поведения ПС при взаимодействии с пользователем.

Адаптация

Приспособление функций и классов для работы в ПС.

Алгоритм

Набор функций, реализующих взаимодействие между контейнером и приложением.

Артефакт

Практический результат, который производится во время выполнения мероприятий, связанных с проектом. В качестве артефактов выступают: документы, диаграммы, исходный код.

Версия

Состояние рабочего артефакта, которое может быть восстановлено в любой момент времени независимо от истории изменения.

Граф

Абстрактный математический объект, представляющий из себя множество вершин графа и набор рёбер, то есть соединений между парами вершин.

Интерфейс

Совокупность методов и правил взаимодействия программ между собой или с пользователем, а также средств, реализующих это взаимодействие.

Компонент (в приложении)

Графический элемент пользовательского интерфейса, предоставляющий функциональность для работы с остальными частями ПС.

Контейнер

Структура, позволяющая инкапсулировать в себя объекты разных типов.

Компонент

Элемент ПК, в приложении каждый компонент определён как вершина графа.

Конфигурация

Совокупность всех компонентов, одного набора элементов.

Модель (в приложении)

Текущее состояние приложения, а именно выбранные и найденные пользователем специальности и специалисты.

Пользователь ПС

Работодатель, использующий приложение для взаимодействия с программной системой. Целью взаимодействия является поиск специалистов.

Поток событий

Череда действий и ответов приложения на действия пользователя.

Приложение

Часть ПС, предоставляющая графический интерфейс пользователя для работы с другими частями ПС.

Проект

Экземпляр выполнения ПС со своим набором специальностей и выбранных специалистов.

ПС

Программная система - совокупность приложений и программ, конструктивно объединенных в единое изделие для выполнения определенной задачи.

Рефакторинг

Процесс изменения внутренней структуры программы, не затрагивая ее внешнего поведения и имеющий целью облегчить понимание работы программы.

Система

Совокупность элементов, находящихся в отношениях и связях между собой и образующих определенную целостность, единство.

Тест-кейс

Тестовый набор - ситуация, проверяющая конкретно взятое условие из требований.

Файл

Именованная область данных на носителе информации.

История изменений

Дата	Версия	Описание	Автор
12.03.19	1.0	Создание документа	Гордеева Т.В.
15.03.19	1.1	Изменение разделов и обзор продукта	Дронников И.М. Полушина Ю.И. Козловский А.О.

Введение

Требуется разработать ПО для наглядного отображения пути для доставки заказов курьерской службы. Разработка ведется в соответствии с методологией RUP. Процесс разработки на каждой итерации сопровождается соответствующим набором документов. Требуется проведение разрабатываемого ПО с написанием документации.

Позиционирование

Решаемая проблема

Таблица 1 – Решаемая проблема

Проблема	Имея множество, заказов сложно в них ориентироваться, а также сложно оптимально составить порядок доставки заказов.
Затрагивает	Работники службы доставки
Влияет	На время, затрачиваемое на составления порядка доставки товаров
Успешное решение	Упростит контроль заказов на доставку, позволит оптимизировать порядок выполнения доставки.

Положение продукта на рынке

Продукт подходит для служб доставки товаров, в которых имеется множество заказов на доставку при ограниченном количестве адресов доставки.

Описание заинтересованных сторон

Заинтересованные лица

Таблица 2 – Заинтересованные лица

Name	**Description**	**Responsibilities**
Студенты	Разработчики ПО	Создание конечного программного продукта, создание документации.
Пользователь	Сотрудники службы доставки	Использование ПО

Рабочая среда пользователя

Работа проводится на персональном компьютере с ОС Windows или Linux. Пользователю предоставляется графический интерфейс. В один момент времени приложением может пользоваться только один пользователь. Все необходимые расчеты выполняются автоматически.

Обзор продукта

Перспективы продукта

Данный продукт не имеет зависимостей от стороннего ПО, не требует установки сторонних библиотек, поддерживает операционные системы семейства Windows и Linux.

Допущения и зависимости

Изменения следующих факторов повлечет за собой изменение текущего документа:

1. Изменения списка поддерживаемых ОС.

2. Изменение перечня ключевых потребностей пользователей.

3. Расширения списка возможностей ПО.

Потребности и возможности

Таблица 3 – Потребности

Необходимость	Приоритет	Проблема	Текущее решение	Предполагаемое решение
Учитывать и сокращать порядок доставки товаров	Высокий	Да	Запись порядка доставки на бумаге, сокращение порядка доставки вручную	Введение порядка доставки в приложении, автоматическая проверка посещаемости объекта

Альтернативы и конкуренция

На рынке не имеется аналогичных продуктов.

Другие требования к продукту

1. Язык программирования – C++.

2. Реализовать шаблон контейнера параллельно последовательного графа, построенного на основе STL-контейнера.

3. Реализовать систему классов исключительных ситуаций и продуманную стратегию контроля аномального поведения программы.

4. Реализовать сериализацию и манипулятор (программный инструмент, который позволяется изменять состояние потока).

5. Реализовать аллокатор (программный инструмент, позволяющий управлять распределением памяти под контейнер).

6. Описать и реализовать проверку инвариантов контейнера предусловий и постусловий для контейнера.

7. Создать, описать и использовать пространства имен.

8. Реализовать графический интерфейс, визуализацию приложения.

9. Разработать полный набор диаграмм UML.

10. Контролировать и фиксировать рефакторинг.

11. Использовать шаблоны проектирования (паттерны) в проекте.

12. Разработать документацию по тестированию.

13. Разработать требуемые артефакты RUP.

14. Описать архитектуру программы.

История изменений

Дата	Версия	Описание	Автор
15.03.19	1.0	Создание документа	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Введение

Данный документ представляет собой описание того, как RUP будет использоваться в проекте “ Post Delivery Logistics ”.

Определения и сокращения

Смотрите артефакт Глоссарий.

Обзор

В оставшейся части документа, будут описаны адаптации RUP к данному проекту, артефакты, появляющиеся в процессе прохождения итераций.

Обзор процесса разработки

Данный проект будет состоять из начальной стадии, стадии уточнения и трех этапов стадии конструирования (анализ и проектирование, реализация, тестирование). Обзор дизайна и кода будет проводится на ключевых этапах итерации, а проверка на соответствие требованиям в конце каждого этапа.

Этапы

Начальный этап

Для описания требований мы будем использовать use-cases. Следующие артефакты будут производится в течение данного процесса:

Таблица 1 - Описание артефактов, создающихся на начальном этапе

Артефакт	**Используемый инструмент**
Видение (Vision)	Microsoft Word
Глоссарий (Glossary)	Microsoft Word
Спецификация прецедентов (Use-cases specificaion)	Microsoft Word, Draw.io
План проекта (Project plan)	Microsoft Word, Microsoft Project
Экономическое обоснование (Business-case)	Microsoft Word

Этап уточнения

На основе разработанных требований формируется основа для последующей реализации программного продукта. Используя use-cases производится создание архитектуры программного продукта, создание прототипа пользовательского интерфейса. Следующие артефакты буду производится в течение данного процесса:

Таблица 2 - Описание артефактов, создающихся на этапе уточнения

Артефакт	**Используемый инструмент**
Документ архитектуры ПО (Soft Architecture Document)	Microsoft Word
Пользовательский интерфейс (прототип)	Microsoft Word, Balsamiq mockups 3

Этап конструирования

Конструирование будет происходить на основе артефактов, полученных в процессе Анализа и Проектирования. Следующие артефакты буду созданы в течение данного процесса:

Таблица 3 - Описание артефактов, создающихся на этапе конструирования

Артефакт	**Используемый инструмент**
Компиляция (Build)	Microsoft Word

Этап тестирование

От тестирования программного продукта будет зависеть необходимость рефакторинга ПО. Следующие артефакты буду производится в течение данного процесса:

Таблица 4 - Описание артефактов, создающихся на этапе тестирования

Артефакт	**Используемый инструмент**
План тестирования (Master Test Plan)	Microsoft Word
Тестирование шаблона (Container test cases)	Microsoft Word
Тестирование пользовательского интерфейса (GUI Test cases)	Microsoft Word

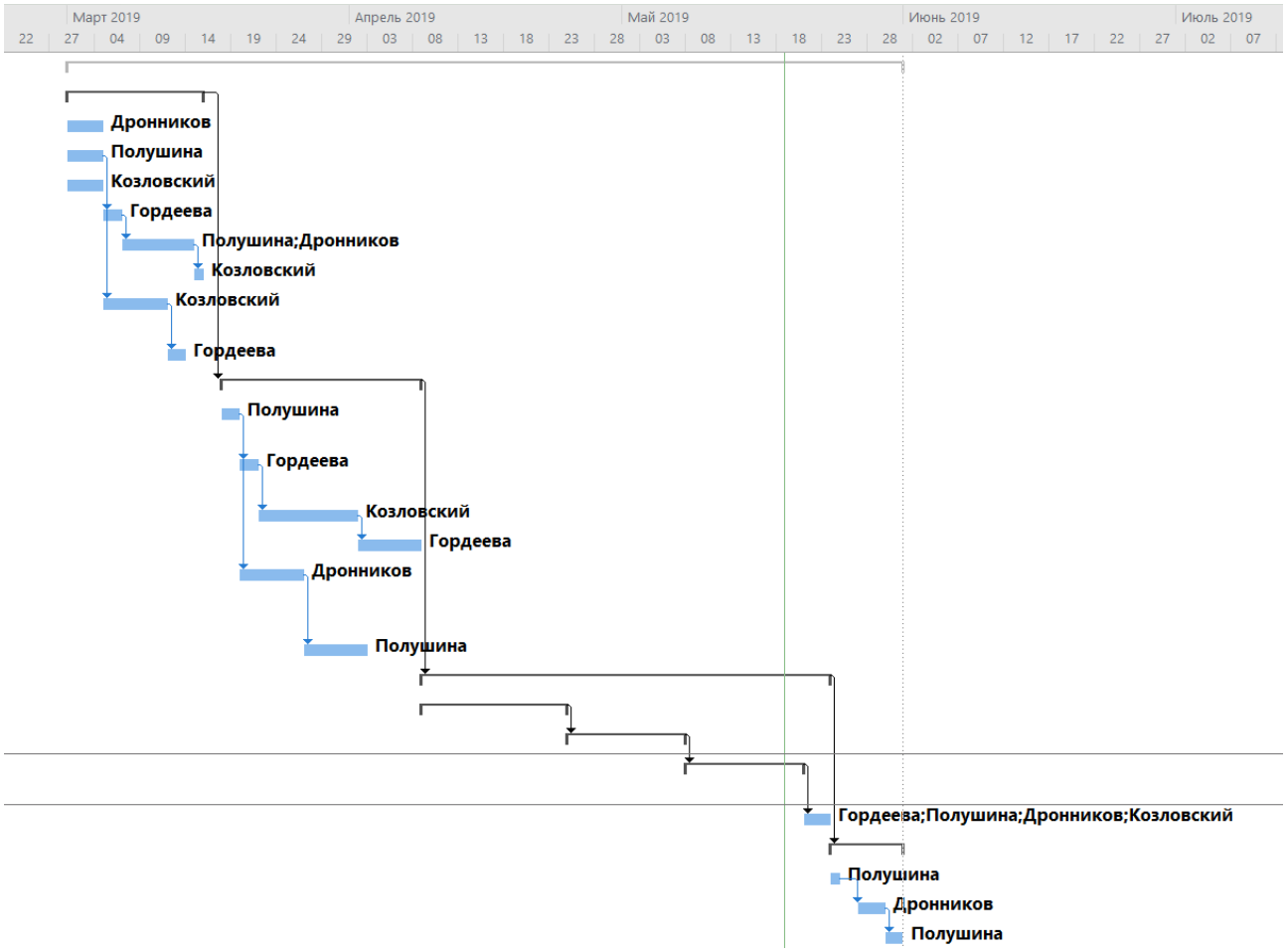
Распределение ролей

Таблица 5 – Распределение ролей

Member	Role	Allocation	Phone	Notes
Гордеева Т.В.	Technical Writer		-	
	Implementer		-	
	Design Reviewer		-	
	System Analyst		-	
	Deployment Manager		-	
Дронников И.М.	Configuration Manager		-	
	Implementer		-	
	Designer		-	
	Test Analyst		-	
	Tool Specialist		-	
Козловский А.О.	Implementer		-	
	Tester		-	
	System Analyst		-	
	Design Reviewer		-	
Полушина Ю.И.	Software Architect		-	
	Architecture Reviewer		-	
	Implementer		-	
	Tester		-	
	Integrator		-	

Post Delivery Logistics

Диаграмма Ганта



Название задачи	Длительнос	Начало	Окончание	Предшественни	Названия ресурсов
Разработка проекта	**67 дней**	**Пт 01.03.19**	**Пт 31.05.19**		
Начальный этап	**12 дней**	**Пт 01.03.19**	**Пт 15.03.19**		
Распределение ролей	3 дней	Пт 01.03.19	Пн 04.03.19		Дронников
Видение	3 дней	Пт 01.03.19	Пн 04.03.19		Полушина
Глоссарий	3 дней	Пт 01.03.19	Пн 04.03.19		Козловский
Варианты использования	2 дней	Вт 05.03.19	Ср 06.03.19	3	Гордеева
План разработки ПО	6 дней	Чт 07.03.19	Чт 14.03.19	5	Полушина;Дронник
Выбор средств разработки ПО	1 день	Пт 15.03.19	Пт 15.03.19	6	Козловский
Экономичское обосноснование	5 дней	Вт 05.03.19	Пн 11.03.19	3	Козловский
Риски	2 дней	Вт 12.03.19	Ср 13.03.19	8	Гордеева
Этап уточнения	**16 дней**	**Пн 18.03.19**	**Пн 08.04.19**	**1**	
Определение потребностей заинтересованных лиц	2 дней	Пн 18.03.19	Вт 19.03.19		Полушина
Определение требований к ПО	2 дней	Ср 20.03.19	Чт 21.03.19	11	Гордеева
Архитектура ПО	7 дней	Пт 22.03.19	Пн 01.04.19	12	Козловский
Модель данных	5 дней	Вт 02.04.19	Пн 08.04.19	13	Гордеева
Прототип пользовательского интерфейса	5 дней	Ср 20.03.19	Вт 26.03.19	11	Дронников
Схема навигации	5 дней	Ср 27.03.19	Вт 02.04.19	15	Полушина
Этап разработки	**33 дней**	**Вт 09.04.19**	**Чт 23.05.19**	**10**	
Разработка контейнера	**12 дней**	**Вт 09.04.19**	**Ср 24.04.19**		
Реализация шаблона	6 дней	Вт 09.04.19	Вт 16.04.19		Дронников
Реализация итератора	2 дней	Ср 17.04.19	Чт 18.04.19	19	Дронников
Реализация аллокатора	2 дней	Вт 23.04.19	Ср 24.04.19	19	Козловский
Реализация манипулятора	3 дней	Пт 19.04.19	Вт 23.04.19	19	Полушина
Реализация проверки инварианта	4 дней	Ср 17.04.19	Пн 22.04.19	19	Козловский
Реализация пространства имён	3 дней	Ср 17.04.19	Пт 19.04.19	19	Гордеева
Реализация системы классов исключительных ситуаций, применимых к шаблону контейнера	2 дней	Ср 17.04.19	Чт 18.04.19	19	Полушина
Бизнес-логика	**9 дней**	**Чт 25.04.19**	**Вт 07.05.19**	**18**	
Реализация алгоритма поиска	7 дней	Чт 25.04.19	Пт 03.05.19		Гордеева
Реализация сериализации	6 дней	Чт 25.04.19	Чт 02.05.19		Дронников
Реализация системы классов исключительных ситуаций, применимых к приложению	2 дней	Пн 06.05.19	Вт 07.05.19	27	Полушина
Функционал пользовательского интерфейса	7 дней	Чт 25.04.19	Пт 03.05.19		Козловский
Пользовательский интерфейс	**9 дней**	**Ср 08.05.19**	**Пн 20.05.19**	**26**	
Реализация интерфейса приложения	3 дней	Ср 08.05.19	Пт 10.05.19		Дронников
Визуализация работы контейнера	6 дней	Пн 13.05.19	Пн 20.05.19	32	Гордеева;Полушина
Проведение рефакторинга	3 дней	Вт 21.05.19	Чт 23.05.19	31	Гордеева;Полушина

Этап тестирования	**6 дней**	**Пт 24.05.19**	**Пт 31.05.19**	**17**	
План тестирования	1 день	Пт 24.05.19	Пт 24.05.19		Полушина
Тестирование	3 дней	Пн 27.05.19	Ср 29.05.19	36	Дронников
Подготовка документов по тестированию	2 дней	Чт 30.05.19	Пт 31.05.19	37	Полушина

История изменений

Дата	Версия	Описание	Автор
07.03.19	1.0	Создание документа	Полушина Ю.И.
09.03.19	1.1	Добавление этапов разработки	Гордеева Т.В. Дронников И.М. Полушина Ю.И. Козловский А.О.

Начальный этап

1. Распределить роли.
2. Разработать артефакт Видение.
3. Разработать артефакт Глоссарий.
4. Разработать артефакт Варианты использования.
5. Разработать план разработки ПО.
6. Выбрать средства разработки ПО.
7. Разработать артефакт Экономическое обоснование.
8. Разработать артефакт Риски.

Этап уточнения

1. Разработать артефакт Определение потребностей заинтересованных лиц.
2. Разработать артефакт Определение требований к ПО.
3. Разработать артефакт Определение требований к ПО.
4. Разработать артефакт Архитектура ПО.
5. Разработать артефакт Модель данных.
6. Разработать артефакт Прототип пользовательского интерфейса.
7. Разработать артефакт Схема навигации.

Этап разработки

Разработка контейнера

1. Реализовать шаблон «Параллельно-последовательный граф».
2. Реализовать итератор.
3. Реализовать аллокатор.
4. Реализовать манипулятор.
5. Реализовать инварианты контейнера.
6. Реализовать свои пространства имен.
7. Реализовать систему классов исключительных ситуаций, применимых к шаблону контейнера.

Бизнес-логика

8. Реализовать алгоритм поиска кратчайшего пути.
9. Реализовать сериализацию.
10. Реализовать систему классов исключительных ситуаций, применимых к приложению.
11. Реализовать функционал пользовательского интерфейса.

Бизнес-логика

12. Реализовать интерфейс приложения.
13. Реализовать визуализацию работы контейнера

Рефакторинг

14. Провести рефакторинги кода.

Этап тестирования

1. Разработать артефакт План тестирования.
2. Разработать артефакт Тестовый набор.
3. Разработать артефакт Результаты тестирования.

Post Delivery Logistics

Спецификация прецедентов

История изменений

Дата	Версия	Описание	Автор
05.03.19	1.0	Создание документа	Гордеева Т.В. Дронников И.М. Полушина Ю.И. Козловский А.О.

Содержание

Цель

В этом документе представлен подробный обзор вариантов использования, существенно влияющих на архитектуру разрабатываемой системы, а также UML диаграмма вариантов использования.

Определение, акронимы и аббревиатуры

Определения, акронимы и аббревиатуры определены в разделе Глоссарий.

Действующие лица

Единственным действующим лицом на все прецеденты является Пользователь.

Прецедент - Сохранение в файл

Описание

Данный прецедент имеет место, когда пользователь хочет сериализовать текущий параллельно-последовательный граф и сохранить его.

Поток событий

Основной поток

- Пользователь выбирает Export cities to file.
- Система создает либо перезаписывает файл с расширением .json в папке с приложением.

Альтернативные потоки

- Система выводит сообщение об ошибке открытия файла.

Предусловие

Граф должен быть не пустым.

Постусловие

По заданному пути существует файл с расширением .json и сериализованным графом.

Прецедент – Загрузка из файла

Описание

Данный прецедент имеет место, когда пользователь хочет десериализовать текущий параллельно-последовательный граф и загрузить его.

Поток событий

Основной поток

- Пользователь выбирает Import cities from file.
- Система загружает граф из файла расширением .json, который находится в папке с приложением.

Альтернативные потоки

- Система выводит сообщение об ошибке открытия файла.

Предусловие

По заданному пути существует файл выбранного системой формата с сериализованным графом.

Постусловие

Граф должен быть не пустым.

Прецедент – Добавление города

Описание

Данный прецедент имеет место, когда пользователь хочет добавить новый город(вершину) в граф.

Поток событий

Основной поток

- Пользователь выбирает Add city.
- Пользователь вводит название города.
- Пользователь выбирает Add.
- Система добавляет город в список городов.

Альтернативные потоки

- Пользователь вводит название уже существующего города.
- Система выводит сообщение о наличии одинаковых вершин.

Прецедент – Добавление пути между городами

Описание

Данный прецедент имеет место, когда пользователь хочет добавить новый город(вершину) в граф.

Поток событий

Основной поток

- Пользователь выбирает Add edge.
- Пользователь выбирает начальный и конечный город.
- Пользователь выбирает Add.
- Система добавляет путь в граф.
- Система создает либо перезаписывает файл с расширением .json в папке с приложением.

Альтернативные потоки

- Система выводит сообщение об ошибке открытия файла.

Прецедент – Удаление города из графа

Описание

Данный прецедент имеет место, когда пользователь хочет убрать один из возможных городов из графа.

Поток событий

Основной поток

- Пользователь выбирает Delete city.
- Пользователь выбирает название города.
- Пользователь выбирает Delete.
- Система создает либо перезаписывает файл с расширением .json в папке с приложением.

Альтернативные потоки

- Система выводит сообщение об ошибке открытия файла.

Прецедент – Удаление пути из графа

Описание

Данный прецедент имеет место, когда пользователь хочет убрать один из возможных городов из графа.

Поток событий

Основной поток

- Пользователь выбирает Delete edge.
- Пользователь выбирает название городов, соединенных удаляемым путем.
- Пользователь выбирает Delete.
- Система создает либо перезаписывает файл с расширением .json в папке с приложением.

Альтернативные потоки

- Пользователь вводит названия несуществующих городов.
- Система выводит сообщение об отсутствии заданных вершин(городов).
- Система выводит сообщение об ошибке открытия файла.

Прецедент – Просмотр кратчайшего маршрута

Описание

Данный прецедент имеет место, когда пользователь хочет найти минимальное расстояние между вершинами графа(городами).

Поток событий

Основной поток

- Пользователь выбирает начальный и конечный город.
- Пользователь выбирает Get directions.
- Система показывает список городов, которые входят в кратчайший путь.

Альтернативные потоки

- Система выводит сообщение об отсутствии пути.

Другие прецеденты

Просмотр графа городов

Пользователь получает изображение графа при каждом его обновлении.

Диаграмма вариантов использования

Рисунок 1 – Диаграмма прецедентов

История изменений

Дата	Версия	Описание	Автор
12.03.19	1.0	Создание документа	Гордеева Т.В.
13.03.19	1.1	Добавление рисков	Гордеева Т.В Дронников И.М. Полушина Ю.И. Козловский А.О.

Цель

Целью данного документа является определение возможных рисков проекта, а также планирование стратегии управления рисками. Своевременное определение рисков позволяет предотвратить срыв сроков проекта, уменьшение его стоимости, а также обеспечить максимальное соответствие продукта требованиям и необходимый уровень его качества.

Определения, акронимы и сокращения

Смотрите артефакт Глоссарий.

Риски

- Персонал

Степень риска: Средняя

Описание: Члены проектной команды могут не выполнить возложенные на них задачи вследствие болезни или других причин.

Влияние:

- ❖ Невыполнение сроков проекта.
- ❖ Увеличение нагрузки на других членов проектной команды.

Показатели появления:

Выявление проявления данного риска следует вести путем периодического опроса участников команды о статусе их работы, а также слежением за ходом выполнения задач в системе управления проектом.

Стратегия предотвращения:

Составление плана выполнения проекта должно вестись с учетом возможности перераспределения задач между участниками проектной команды.

План реагирования:

1. При временной невозможности выполнения задач участником проектной команды необходимо проанализировать задачи, решаемые им на текущий момент, и оценить возможность переноса сроков этих задач.

2. В случае полной невозможности выполнения задач участником проекта и при возникновении трудностей изменения сроков задач необходимо выполнить перераспределение ресурсов между проблемными задачами.

- Профессиональная квалификация

Степень риска: Средняя

Описание: Члены проектной команды могут частично или полностью не обладать навыками работы с используемыми технологиями для решения поставленных задач, а также не иметь необходимых знаний в предметной области.

Влияние:

- ❖ Невыполнение поставленных задач и сроков проекта.
- ❖ Увеличение затрат на изучение соответствующих технологий и материала.

Показатели появления:

Выявление проявления данного риска проводится путем предварительного опроса участников проекта на предмет владения используемыми технологиями и знаниями в предметной области.

Стратегия предотвращения:

При составлении плана выполнения проекта и планировании распределения задач между участниками необходимо выполнить оценку текущей квалификации каждого участника. В том случае, если будут выявлены проблемы, связанные с незнанием используемых технологий, запланировать задачи по их изучению.

План реагирования:

1. Оценить возможность выполнения определенных задач другими участниками проекта.

2. Обеспечить возможности (время, материалы, помощь более квалифицированных участников проекта) для обучения нужным технологиям.

- Ошибки проектирования

Степень риска: Высокая

Описание: На этапе проектирования системы могут быть допущены ошибки при определении подхода проектирования системы, а также выборе вариантов реализации частей системы.

Влияние:

- ❖ Изменение сроков выполнения проекта из-за возможных проблем с добавлением нового функционала в проект.
- ❖ Частичное или полное несоответствие поставленным требованиям.
- ❖ Частично или полностью нереализованная функциональность.

Показатели появления:

Внесение изменений в проект и добавление нового функционала проходит с высокими трудозатратами.

Стратегия предотвращения:

1. Прототипирование системы и предоставление прототипа Заказчику.

2. Запланировать задачи на возможную доработку архитектуры системы.

План реагирования:

Осуществить поиск решений, позволяющих решить возникшие проблемы в рамках существующей архитектуры.

- Ошибки реализации

Степень риска: Высокая

Описание: При реализации проекта участники могут использовать сложные и неоптимальные алгоритмы. Код системы может стать сложным для понимания и доработки.

Влияние:

- ❖ Несоответствие продукта необходимому качеству.
- ❖ Сложности с внесением изменений.
- ❖ Сложности с исправлением ошибок и отладкой.

Показатели появления:

Исправление ошибок, внесение изменений в проект и добавление нового функционала проходит с высокими трудозатратами.

Стратегия предотвращения:

1. Выработать единый стиль написания кода.
2. Проводить периодический просмотр кода.
3. При реализации использовать шаблоны проектирования.

План реагирования:

1. Выполнить пересмотр и рефакторинг кода.
2. Если необходимо, провести оптимизацию алгоритмов.

- Ошибки тестирования

Степень риска: Средняя

Описание: На этапе тестирования могут выявиться критические ошибки, которые не позволят полноценно работать с системой. Кроме того, качество тестирования может не выявить некоторые ошибки, которые повлекут за собой несоответствие продукта поставленным требованиям.

Влияние:

- ❖ Увеличение затрат на исправление критических ошибок.
- ❖ Проявление в конечном продукте не выявленных ошибок.

Показатели появления:

1. Выявлены ошибки в работе разрабатываемого продукта.

2. Выявлено несоответствие продукта поставленным требованиям.

Стратегия предотвращения:

1. На каждом этапе тестирования формулировать описание функциональности, предоставляемой на тестирование.

2. Запланировать достаточное время на выполнение тестирования.

План реагирования:

1. Своевременно исправить найденные ошибки.

2. Уделить время на изучение влияния внесенных изменений на поведение системы.

- Выбор средств разработки

Степень риска: Низкая

Описание: Неверный выбор средств разработки (среда разработки, инструментарий) влияет на качество и время реализации продукта.

Влияние:

- ❖ Невыполнение сроков проекта.

Показатели появления:

1. Участники проекта тратят много времени на задачи, не связанные с написанием кода (написание скриптов сборки, контроль версий и т.д.).

2. Сроки выполнения задач увеличиваются.

Стратегия предотвращения:

1. Определить требования к средствам разработки.

2. Провести анализ решений для разработки и выбрать необходимые средства.

План реагирования:

1. Выполнить анализ возможности смены средств разработки.

2. В случае невозможности изменения средств разработки определить методы решения проблемы в текущей среде, с помощью дополнительных инструментальных средств.

История изменений

Дата	Версия	Описание	Автор
05.03.19	1.0	Создание документа	Гордеева Т.В.
10.03.19	1.1	Добавление расчётов	Гордеева Т.В. Дронников И.М. Полушина Ю.И. Козловский А.О.

Расчет заработной платы разработчика

Разработкой приложения постоянно занимаются четыре человека, месячная заработная плата принимается 60000 руб., они исполняют следующие роли: Configuration Manager, Implementer, Test Analyst, Tool Specialist, Software Architect, Designer, Architecture Reviewer, Tester, Integrator, Design Reviewer. Для определения часовой ставки заработной платы необходимо разделить заработную плату за месяц на количество рабочих часов в месяце (168 часов). Таким образом, получаем часовую ставку 357 руб./час.

На основе данных о трудоемкости выполняемых работ и часовой ставки определим расходы на заработную плату и отчислений на страховые взносы на обязательное социальное, пенсионное и медицинское страхование.

Расходы на основную заработную плату определяются по формуле:

$$З_{осн.з/пл} = \sum_{i=1}^{k} T_i * C_i,$$

где $З_{осн.з/пл}$ - расходы на основную заработную плату исполнителей (руб.);

k – количество исполнителей (в данном случае разработкой занимается 3 человека);

T_i – время, затраченное i-м исполнителем на проведение работы (в данном случае в часах);

C_i – ставка i-го исполнителя (руб./час).

Расходы на дополнительную заработную плату исполнителей определяются по формуле:

$$З_{доп.з/пл} = З_{осн.з/пл} * \frac{Н_{доп}}{100},$$

где $З_{доп.з/пл}$ – расходы на дополнительную заработную плату исполнителей (руб.);

$Н_{доп}$ – норматив дополнительной заработной платы (%). Этот норматив принимаем равным 14%.

Отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование с основной и дополнительной заработной платы исполнителей определяются по формуле:

$$З_{соц} = (З_{осн.з/пл} + З_{доп.з/пл}) * \frac{Н_{соц}}{100},$$

где $З_{соц}$ - отчисления на социальные нужды с заработной платы (руб.);

$Н_{соц}$ – норматив отчислений на страховые взносы на обязательное социальное, пенсионное и медицинское страхование (%). Этот норматив принимаем равным 30%.

Расчет амортизационных отчислений основных средств

Стоимость оборудования:

Наименование оборудования	Описание	Стоимость (с НДС), руб. за единицу
4*Офисный компьютер	Ноутбук Мышь	65000

Стоимость программного обеспечения:

Наименование ПО	Описание	Стоимость, руб. за единицу

4*Операционная система Windows 10 (8.1)		Предустановлена
4*MicrosoftOffice		Предустановлен
4*QTCreator	Среда разработки	Распространяется бесплатно
2*StarUML	UML - редактор	Распространяется бесплатно

Для разработки использовались основные средства, амортизационные отчисления по которым определяются как:

$$А_i = Ц_{п.н.i} * \frac{Н_{ai}}{100},$$

где $А_i$ – амортизационные отчисления за год по i-му основному средству (руб.);

$Ц_{п.н.i}$ – первоначальная стоимость i-го основного средства (руб.);

$Н_{ai}$ - годовая норма амортизации i-го основного средства (%).

Эксплуатироваться компьютеры начали в месяц приобретения. Указанное основное средство входит во вторую амортизационную группу, как имущество со сроком полезного использования свыше 2 лет до 3 лет включительно. Соответственно, срок полезного использования персонального компьютера устанавливаем в количестве 3 лет, т.е. получаем $^1/_3 * 100\% = 33{,}3\ \%$

Величина амортизационных отчислений по i-му основному средству, используемом при работе, определяется по формуле:

$$А_{iпр} = A_i * \frac{Т_i}{12},$$

где $А_{iпр}$ – амортизационные отчисления по i-му основному средству, используемому в работе (руб.);

$Т_i$ - время, в течение которого использовалось i-ое основное средство (мес.) (1 месяц).

Поскольку платные программные продукты отсутствуют, то затраты $З_м$ считаем нулевыми.

Результаты вычислений, по указанным формулам:

$З_{осн.з/пл}$, руб.	$З_{доп.з/пл}$, руб.	$З_{соц}$, руб.	$А_i$, руб.	$А_{iпр}$, руб.	$З_м$, руб.
179.9 тыс.	25.19 тыс.	61.5 тыс.	21.645 тыс.	1.803 тыс.	0

Смета затрат создание продукта:

№	Наименование статьи	Сумма, руб.
1	Расходы на оплату труда	205.09 тыс.
2	Отчисления на социальные нужды	61.5 тыс.
3	Амортизационные отчисления	86.58 тыс.
4	Сырье, материалы, ПО	0
ИТОГО затрат		353.17 тыс.

ПРИЛОЖЕНИЕ Б. АРТЕФАКТЫ ЭТАПА УТОЧНЕНИЯ

Post Delivery Logistics

Схема навигации

История изменений

Дата	Версия	Описание	Автор
01.05.19	1.0	Создание документа	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Введение

Цель

В этом документе представлено графическое изображения того, как навигация внутри программного продукта будет перемещать пользователя в части пользовательского интерфейса внутри приложения.

Объём

Представлены все необходимые пути для того, чтобы пользователь смог выбрать все необходимые составляющие и сохранить составленный им набор компонентов.

Определения, сокращения

Смотрите артефакт Глоссарий.

Обзор

Навигационная карта представляет собой графическое представление того, как пользователь может перемещаться между различными экранами, доступными в системе.

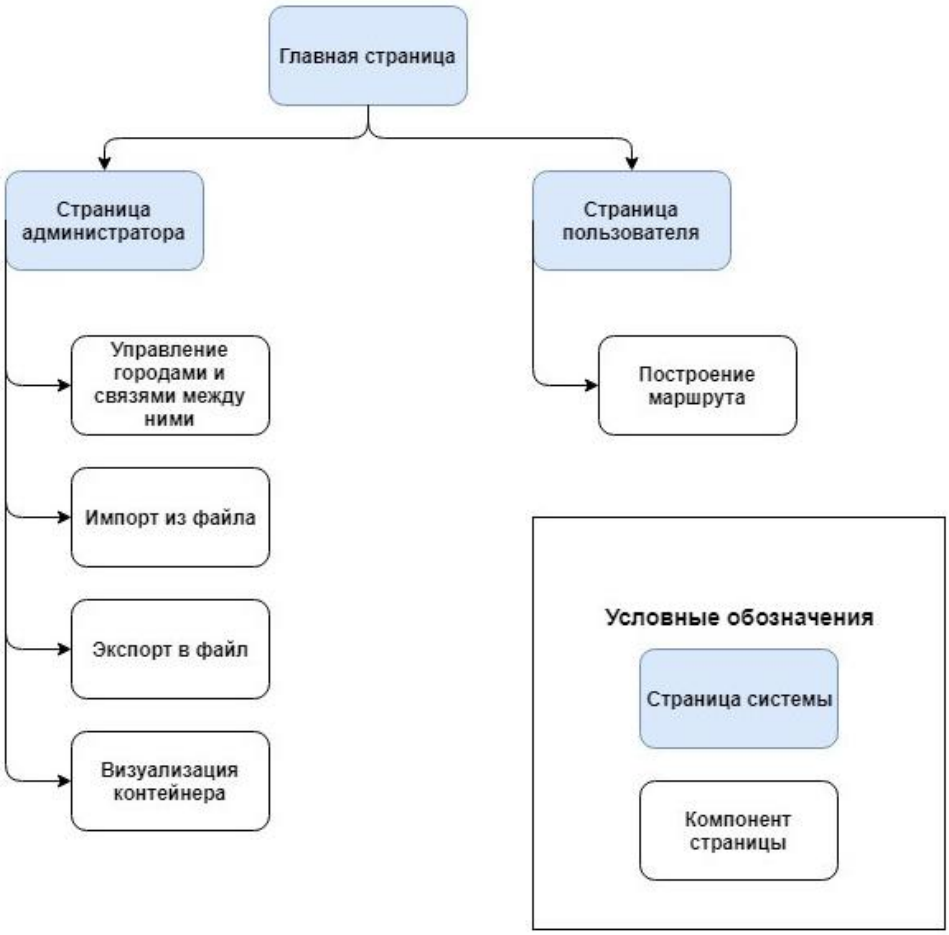


Рисунок 1 - Карта навигации

Эта простая иерархия навигации показывает ход с одного экрана на другой.

Главная страница предоставляет доступ к странице администратора или странице пользователя.

Post Delivery Logistics

Запросы заинтересованных сторон

История изменений

Дата	Версия	Описание	Автор
01.04.19	1.0	Создание документа	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Цель

Целью этого артефакта является захват всех запросов, сделанных в проекте, а также то, как эти запросы были устранены. Хотя системный аналитик несет ответственность за этот артефакт, многие люди будут вносить свой вклад в это: маркетинг людей, конечных пользователей, клиентов - всех, кто считается заинтересованным лицом в результате проекта. И соответствующие запросы должны отслеживаться и сообщаться в соответствии с утвержденным процессом управления запросами на изменение (CRM).

Примерами источников запросов заинтересованных сторон являются:

Результаты собеседований с заинтересованными сторонами

Запрос на изменение (CR)

Техническое задание

Запрос предложения

Постановка задачи

Законы и правила

Устаревшие системы

Бизнес-модели

Сроки

Запросы заинтересованных сторон собираются в основном на этапах создания и разработки, однако вы должны продолжать собирать их на протяжении всего жизненного цикла проекта для планирования усовершенствований и обновлений продукта. Инструмент отслеживания запросов на изменение полезен для сбора и определения приоритетности этих запросов.

Ответственность

Системный аналитик отвечает за целостность артефакта запросов заинтересованных сторон, гарантируя, что:

- Все заинтересованные стороны получили возможность добавлять запросы.
- Все элементы в этом артефакте учитываются при разработке подробных требований в модели использования и дополнительных спецификациях.

Портфолио

Запросы заинтересованных сторон лучше всего управляются в базе данных, например, Rational ClearQuest и / или Rational RequisitePro, для отслеживания состояния, определения приоритетов, создания отчетов и установления прослеживаемости. См. «Наставник инструментов»: управление запросами заинтересованных сторон с использованием Rational ClearQuest® и Rational RequisitePro®. Включает в себя определение информации (атрибутов), подлежащей документированию для каждого заинтересованного лица и каждого запроса заинтересованных сторон.

Post Delivery Logistics

Прототип интерфейса

История изменений

Дата	Версия	Описание	Автор
21.03.2019	1.0	Первоначальная версия	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.
08.05.2019	1.1	Добавление прототипа пользовательского интерфейса	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Содержание

Определения и сокращения

Смотрите артефакт Глоссарий.

Введение

Данный документ представляет собой описание того, как должен выглядеть пользовательский интерфейс в приложении.

На рис. 1-5 представлен разработанный интерфейс приложения.

Обзор прототипа пользовательского интерфейса

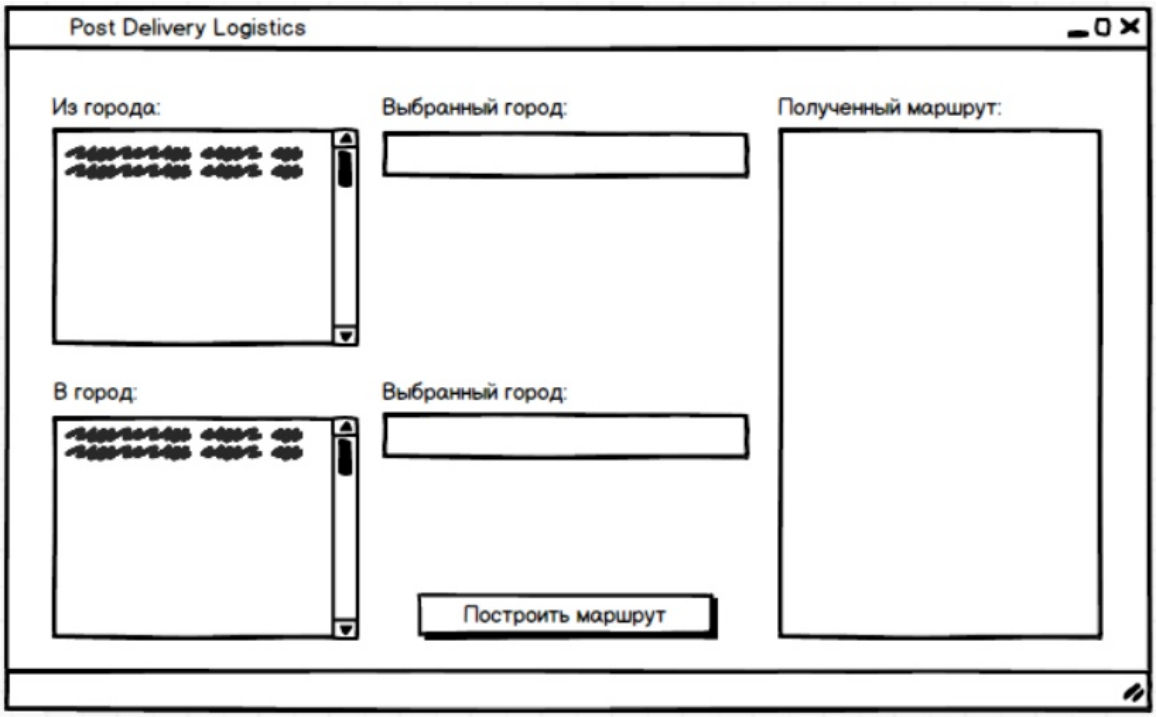


Рисунок 1 - Прототип пользовательского интерфейса

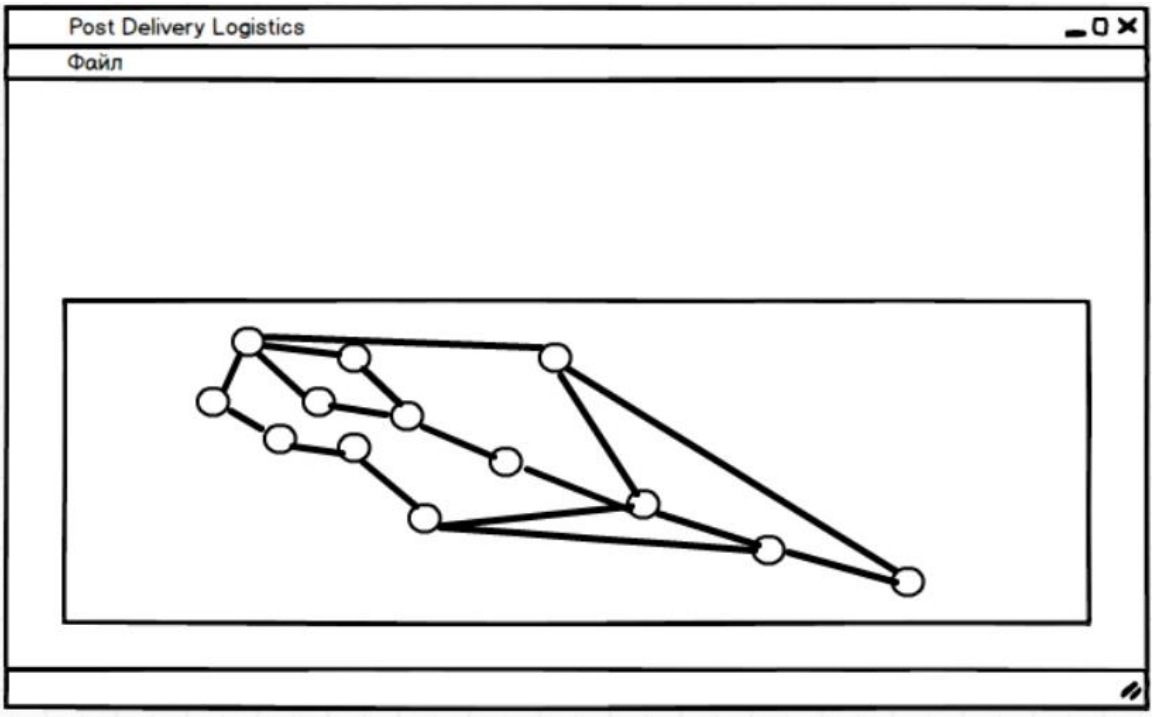


Рисунок 2 - Прототип интерфейса администратора

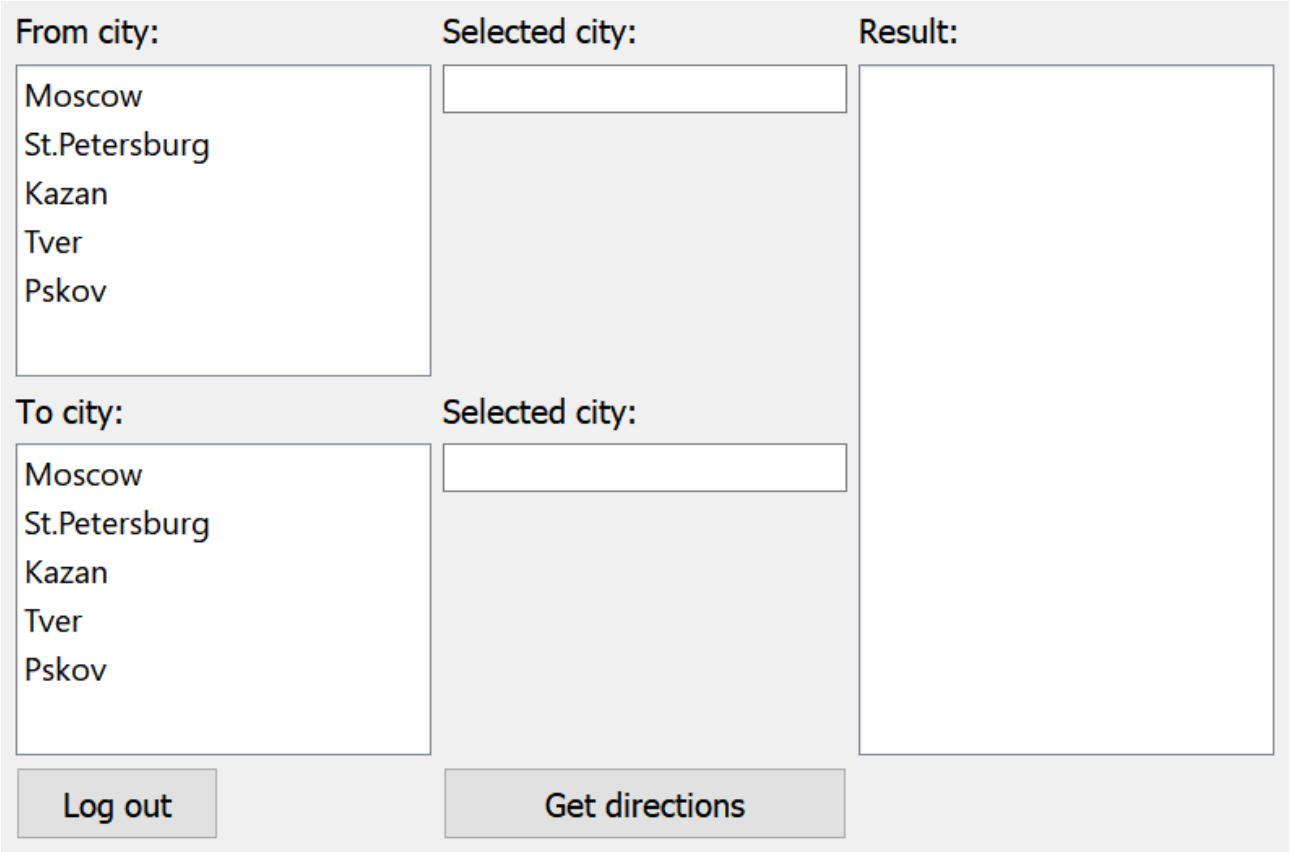


Рисунок 3 - Пользовательский интерфейс

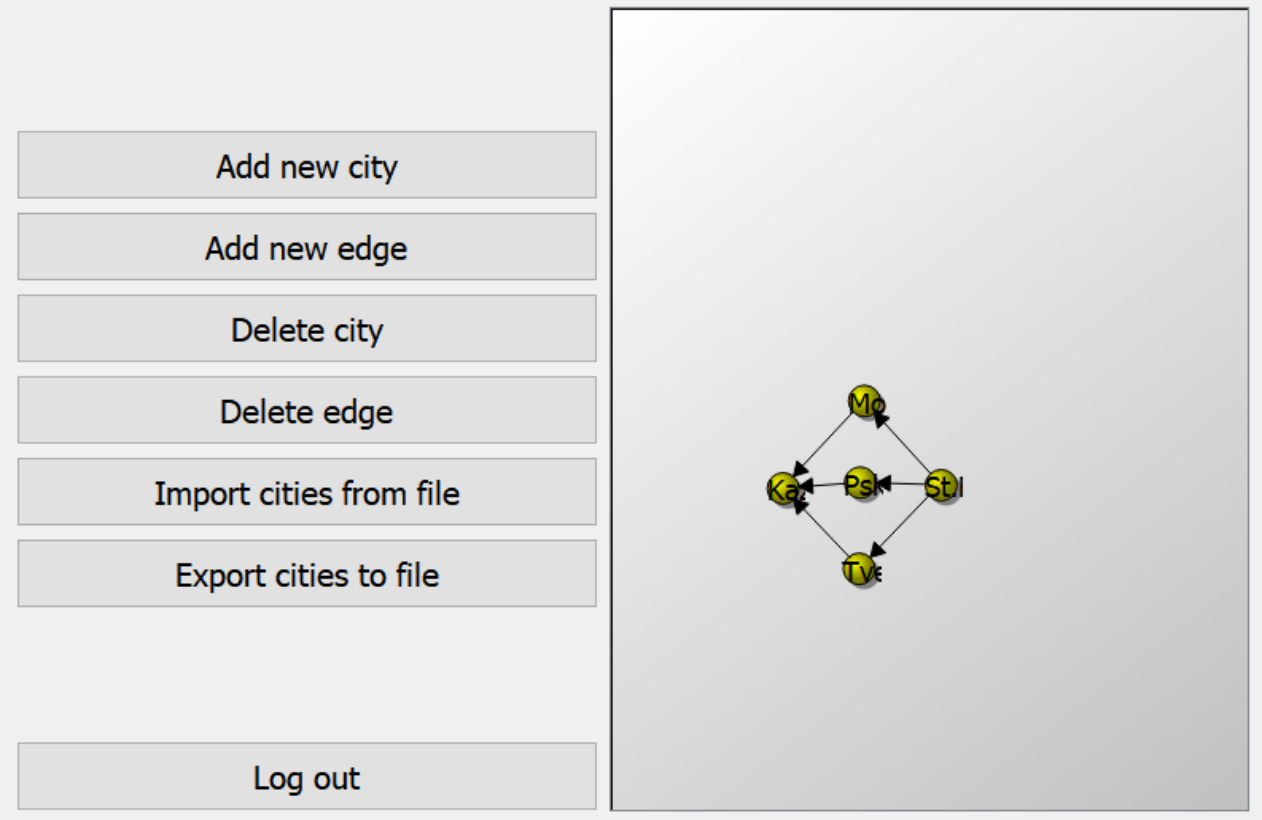


Рисунок 4 - Интерфейс администратора

Рисунок 5 - Интерфейс формы входа

ПРИЛОЖЕНИЕ В. АРТЕФАКТЫ ЭТАПА РАЗРАБОТКИ

Post Delivery Logistics

Компиляция

История изменений

Дата	Версия	Описание	Автор
15.05.2019	1.0	Первоначальная версия	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Введение

Данный документ описывает рабочую версию программной системы, демонстрирующей подмножество функций, предоставляемых программной системой. Компоновка (обычно в процессе компиляции и компоновки исходного кода) включает один или несколько элементов реализации (часто исполняемых), состоящих из других элементов.

Обзор

Программная система состоит из нескольких взаимосвязанных частей: Контейнер, Графический Пользовательский Интерфейс, Алгоритм.

На этапе компиляции используются:

1) Контейнер, обеспечивающий интерфейс необходимой структуры данных для Алгоритма.

2) Алгоритм, реализующий функциональную часть программной системы. Для корректной работы Алгоритма необходим Контейнер, а также класс, обеспечивающий передачу данных в графический пользовательский интерфейс.

3) Графический пользовательский интерфейс, реализующий отображение итогов работы Алгоритма в текущей модели, а также взаимодействующий с базой данных.

План компоновки

При изменении частей программы предлагается следующий план компоновки:

Изменение Контейнера – соответствующее изменение Алгоритма.

Изменение Алгоритма – соответствующее изменение Графического Пользовательского Интерфейса.

Изменение Графического Пользовательского Интерфейса - соответствующее изменение Алгоритма.

Post Delivery Logistics

План итераций

История изменений

Дата	Версия	Описание	Автор
20.05.2019	1.0	Первоначальная версия	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Введение

Цель

Этот документ описывает подробный план начальной итерации проекта. Целью этой итерации является определение содержания проекта, синтез потенциальной структуры и подготовка среды проекта.

Область

Настоящий документ описывает план начальной итерации проекта. Этот документ предназначен для использования Руководителем проекта и командой разработчиков.

Определения и сокращения

Смотрите в разделе Глоссарий

План

Задачи

В таблице ниже представлены общие задачи на текущую итерацию с их датами начала и окончания.

Задача	Дата начала	Дата окончания
Определение содержания проекта	14.02.19	20.02.19

Синтез потенциальной структуры	01.03.19	15.03.19
Подготовка среды проекта	20.03.19	31.03.19

Описание задач

Задача	Дата начала	Дата окончания	Продолжи-тельность	Исполнитель
Определение содержания проекта	14.02.19	20.02.19	6 дней	
Выявление требований	14.02.19	20.02.19	6 дней	Разработчик требований
Выявление запросов заинтересованных лиц	14.02.19	20.02.19	6 дней	Аналитик
Видение проекта	14.02.19	20.02.19	6 дней	Аналитик
Выявление рисков проекта	14.02.19	20.02.19	6 дней	Руководитель проекта
Синтез потенциальной структуры	01.03.19	15.03.19	15 дней	
Составление первоначального архитектурного плана	01.03.19	15.03.19	15 дней	Разработчик архитектуры

Составление первоначального плана разработки ПО	01.03.19	15.03.19	15 дней	Руководитель проекта
Подготовка среды проекта	20.03.19	31.03.19	10 дней	
Установка необходимого ПО	20.03.19	31.03.19	10 дней	

Результаты итерации

Задача	Артефакт
Определение содержания проекта	● Запросы заинтересованных лиц ● Видение ● Требования к программному обеспечению ● Список рисков
Синтез потенциальной структуры	● Документ архитектуры программного обеспечения ● План разработки программного обеспечения
Подготовка среды проекта	

Ресурсы

1. Для выполнения задач плана предварительной итерации необходимы следующие трудовые ресурсы:

2. Разработчик требований: Козловский А.О.

3. Аналитик: Гордеева Т.В.

4. Руководитель проекта: Дронников И.М.

5. Разработчик архитектуры: Полушина Ю.И.

6. Оборудование и программное обеспечение:

7. 4 ПК (Windows 10, Microsoft Office 2019, Microsoft Project 2019, Qt Creator)

8. Доступ к интернету

Критерии оценки

Результаты итерации должны быть просмотрены и одобрены каждым членом команды.

Post Delivery Logistics

План тестирования

История изменений

Дата	Версия	Описание	Автор
25.04.2019	1.0	Создание документа	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Содержание

Введение

Цель

Целью данного документа является описание подхода к тестированию готового продукта.

Включает следующие цели:

- Определить компоненты для тестирования;
- Перечислить рекомендуемые требования к проведению тестирования;
- Описать рекомендуемые стратегии тестирования;
- Определить требуемые ресурсы.

Обзор

В следующих разделах описаны предпосылки, ресурсы, основные группы тестирования, цели и технологии.

Оценка задачи

Предпосылки

В процессе разработки ПО необходимо сформулировать план тестирования для дальнейшего использования. Данный документ поможет проверить соответствие требованиям при помощи тестов.

Оценка задачи

Задачей данного документа является определения ключевых моментов тестирования с целью:

- Выявления ошибок при работе ПО;
- Выявление важных проблем и рисков;
- Формирование представления и оценки предполагаемых рисков;
- Проверки спецификации;
- Формирование представления о качестве продукта.

Мотивация тестирования

Тестирование мотивировано рядом вещей, необходимых для проверки, а именно: функциональными и нефункциональными требованиями, вариантами использования, элементами дизайна, а также запросами на изменение.

Цель тестирования элементов

Целью тестирования элементов ПО является выявления и исправления ошибок для получения более качественного продукта, удовлетворяющего требованиям заказчика.

Описание планируемых тестов

В ходе тестирования ПО планируется использовать следующие группы тестов:

● Функциональное тестирование, которое позволит проверить на соответствие требованиям, описанным в документации;

● Модульное тестирование, необходимое для тестирования шаблона контейнера и алгоритма;

● Тестирование пользовательского интерфейса для тестирования приложения.

Типы тестирования

Функциональное тестирование

Цель тестирования: определить, соответствует ли продукт требованиям, указанным в документации.

Технология тестирования: на каждой итерации проверяется соответствие реализованных функциональных возможностей ПО в соответствии с документацией.

Критерий завершенности: соответствие всех требований, заявленных в спецификации к ПО, заявленные на данную итерацию.

Примечания: при необходимости произвести переосмысление требований.

Модульное тестирование

Цель тестирования: определить соответствие функциональным требованиям отдельных модулей ПС.

Технология тестирования: рассматриваются два модуля ПО - шаблон контейнера и работа алгоритма Параллельно-последовательного графа на примере выборки. Для каждого модуля определяется из спецификации набор требований, определяющих тесты.

Критерий завершенности: Выполнение всех тестовых случаев, определенных для конкретного модуля.

Примечания: отсутствуют.

Тестирование пользовательского интерфейса

Цель тестирования: определить наибольшее количество ошибок при работе с пользовательским интерфейсом программы и исправить их.

Технология тестирования: для тестирования необходимо использовать ручное тестирование графических форм. Тестовые случаи определяются в зависимости от спецификации требований и вариантов использования.

Критерий завершенности: Выполнение всех тестовых случаев.

Примечания: переработка форм при неудачном тестировании.

Ресурсы

Сотрудники

За разработку тестирования шаблона, алгоритма выборки Параллельно-последовательного графа, приложения и пользовательского интерфейса отвечает отдельный член команды – Дронников И.М.

Система

Тестирование проводится на персональном компьютере, соответствующий необходимым требованиям ПО.

Post Delivery Logistics

Тестирование шаблона «Параллельно-последовательный граф»

Версия 1.0

История изменений

Дата	Версия	Описание	Автор
19.05.2019	1.0	Первоначальная версия	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Содержание

Введение

Цель

Целью данного документа является описание тестирования функциональных возможностей шаблона «Параллельно-последовательный граф», который был реализован в ходе разработки ПС.

Описание функциональных особенностей

Список функциональных возможностей контейнера:

- Создать вершину - создает экземпляр класса Vertex и добавляет в объект класса контейнера.
- Удалить вершину - удаляет экземпляр класса Vertex.
- Очистка контейнера - очищает контейнер от всех вершин и ребер.
- Сериализация контейнера - вывод контейнера в поток.

Post Delivery Logistics

Тестовый набор

История изменений

Дата	Версия	Описание	Автор
19.05.2018	1.0	Первоначальная версия	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Содержание

Введение

Цель

Целью данного документа является описание подхода к тестированию функциональных возможностей приложения.

Описание функциональных возможностей

Вкладка меню

Вкладка меню состоит из следующих компонентов:

- Сохранение – предоставляет пользователю сохранить созданный граф для дальнейшего использования;

- Открытие – предоставляет возможность открытие ранее сохраненного файла с графом.

Панель пользователя

Панель пользователя состоит из следующих компонентов:

- Рабочая область – предоставляет пользователю список всевозможных компонентов, хранимых Параллельно-последовательным графом;

Панель администратора

- Контрольная панель состоит из следующих компонентов:
- Кнопка добавления вершины – кнопка, при нажатии которой вершина добавляется в контейнер и отображается на рабочей области;
- Кнопка удаления вершины - кнопка, при нажатии которой вершина удаляется из контейнера и исчезает с рабочей области;
- Кнопка добавления связи– кнопка, при нажатии которой связь добавляется в контейнер и отображается на рабочей области;
- Кнопка удаления связи - кнопка, при нажатии которой связь удаляется из контейнера и исчезает с рабочей области;
- Кнопка сохранения графа - кнопка, при нажатии которой граф сохраняется в файл;
- Кнопка загрузки графа - кнопка, при нажатии которой граф импортируется из файла.

Тест-кейсы функциональных возможностей

Добавление вершины

Шаги:

1. Открыть программу.
2. Переход в панель администратора.
3. Нажать кнопку добавления вершины.
4. Ввод вершины графа.
5. Нажатие на кнопку добавления вершины.

Ожидаемый результат: вершина отобразилась на рабочей области.

Удаление вершины

Шаги:

1. Открыть программу.

2. Переход в панель администратора.

3. Нажать кнопку удаления вершины.

4. Выбор вершины из выпадающего меню вершин.

5. Нажатие на кнопку удаления вершины.

Ожидаемый результат: вершина исчезла с рабочей области.

Выполнение алгоритма

Шаги:

1. Открыть программу.

2. Переход в панель пользователя.

3. Выбор начальной и конечной вершины.

4. Нажатие на кнопку запуска алгоритма.

Ожидаемый результат: построенный маршрут.

Сохранение графа

Шаги:

1. Открыть программу.

2. Переход в панель администратора.

3. Нажатие на кнопку сохранения.

4. Файл сохраняется.

Ожидаемый результат: создан файл с сохраненными данными.

Открытие графа

Шаги:

1. Открыть программу.
2. Переход в панель администратора.
3. Нажатие на кнопку открыть.

Ожидаемый результат: граф, содержащийся в файле отображен на рабочей области.

ПРИЛОЖЕНИЕ Д. ДИАГРАММЫ UML

Диаграмма прецедентов

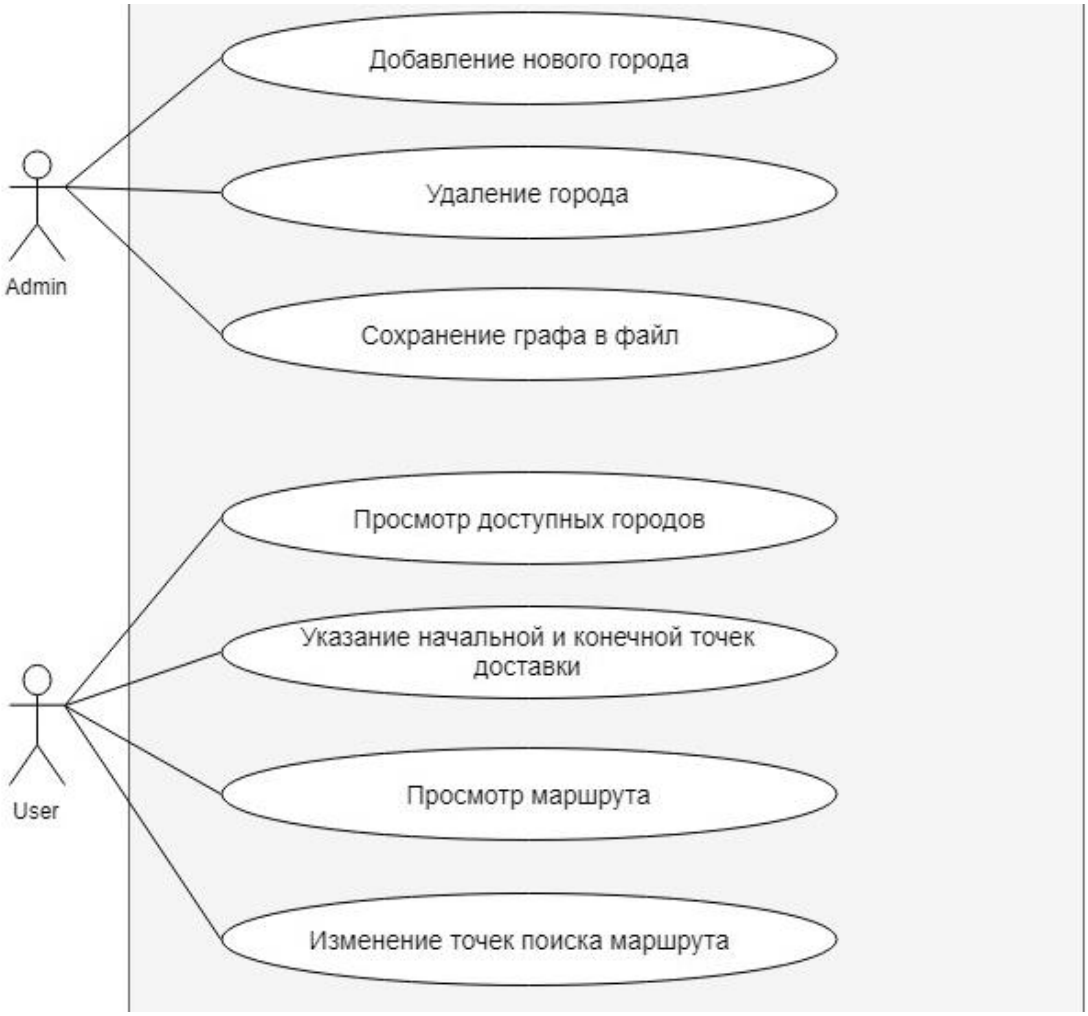


Диаграмма деятельности

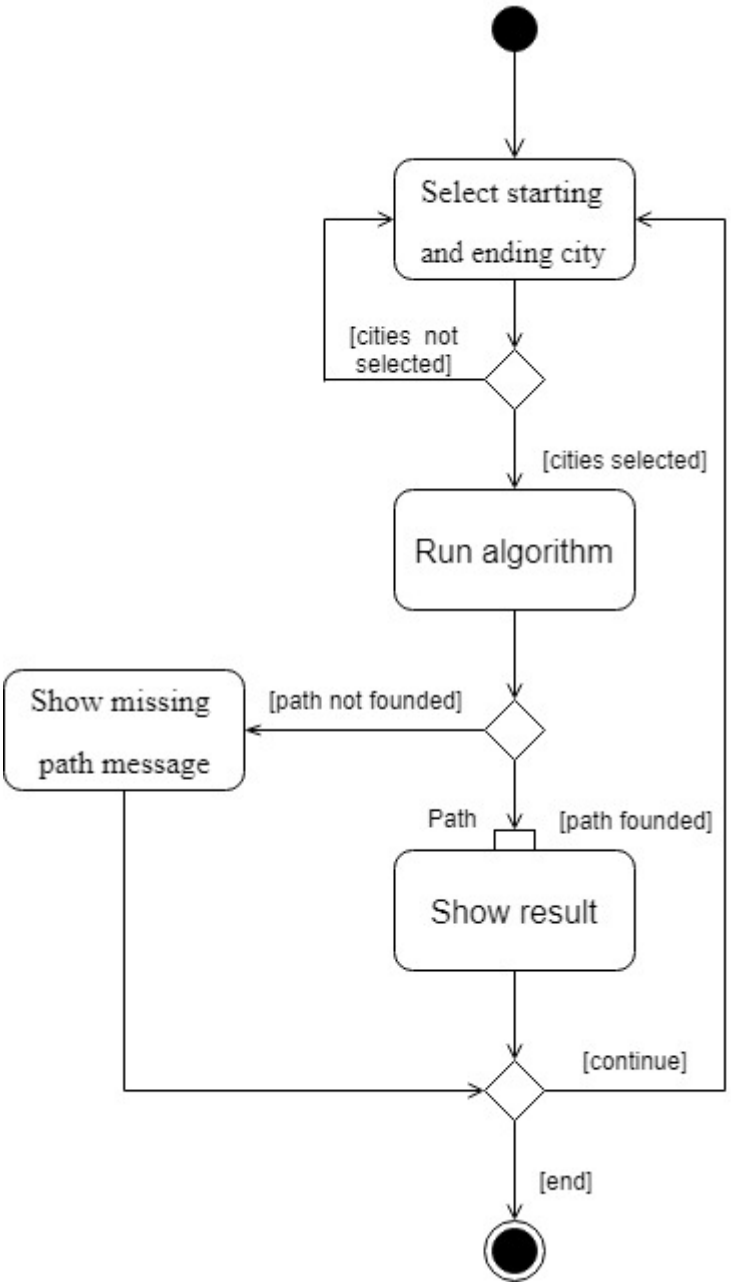


Диаграмма коммуникации

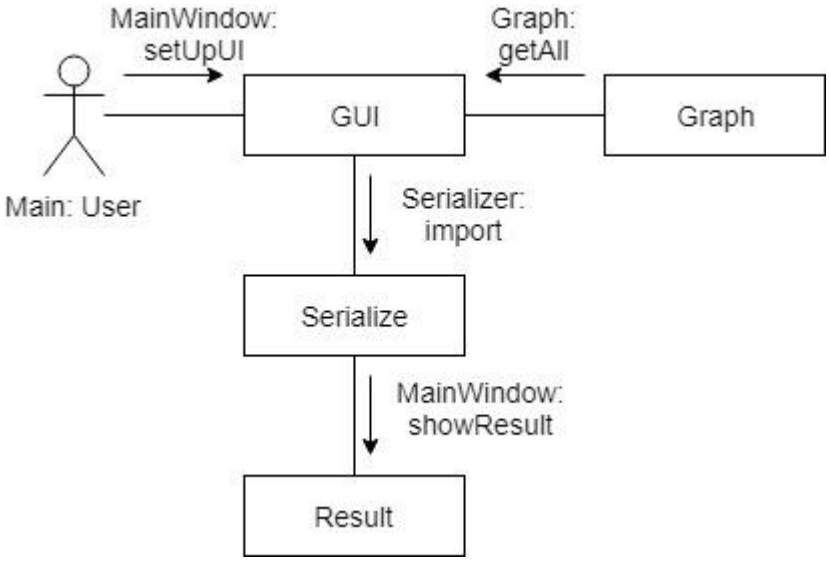


Диаграмма компонентов

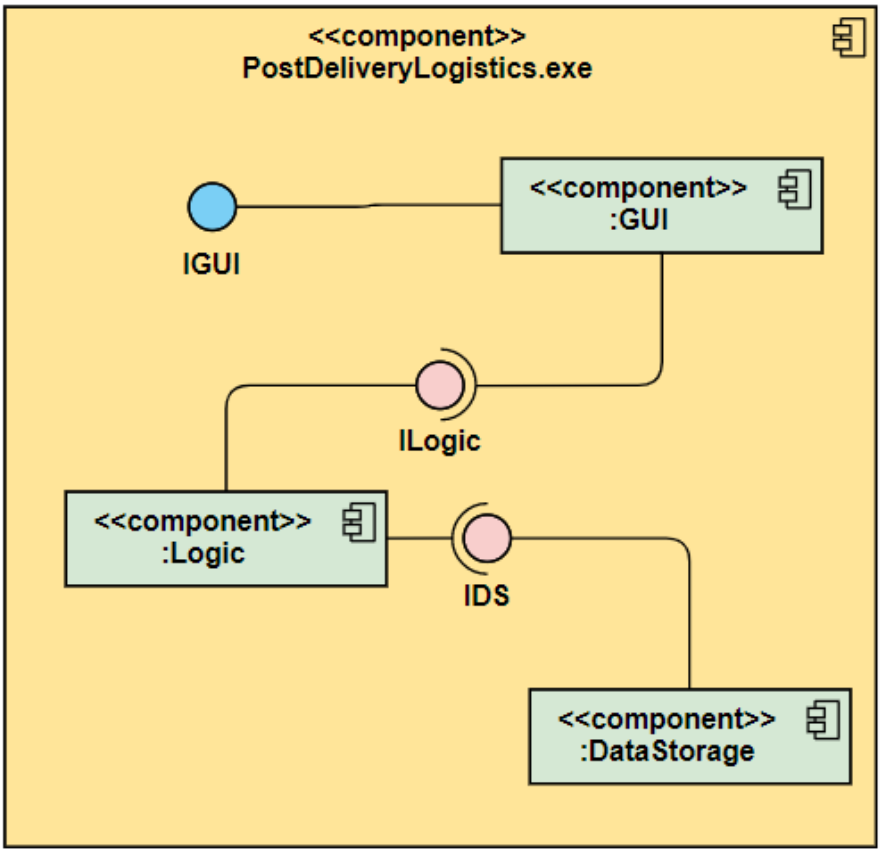


Диаграмма объектов

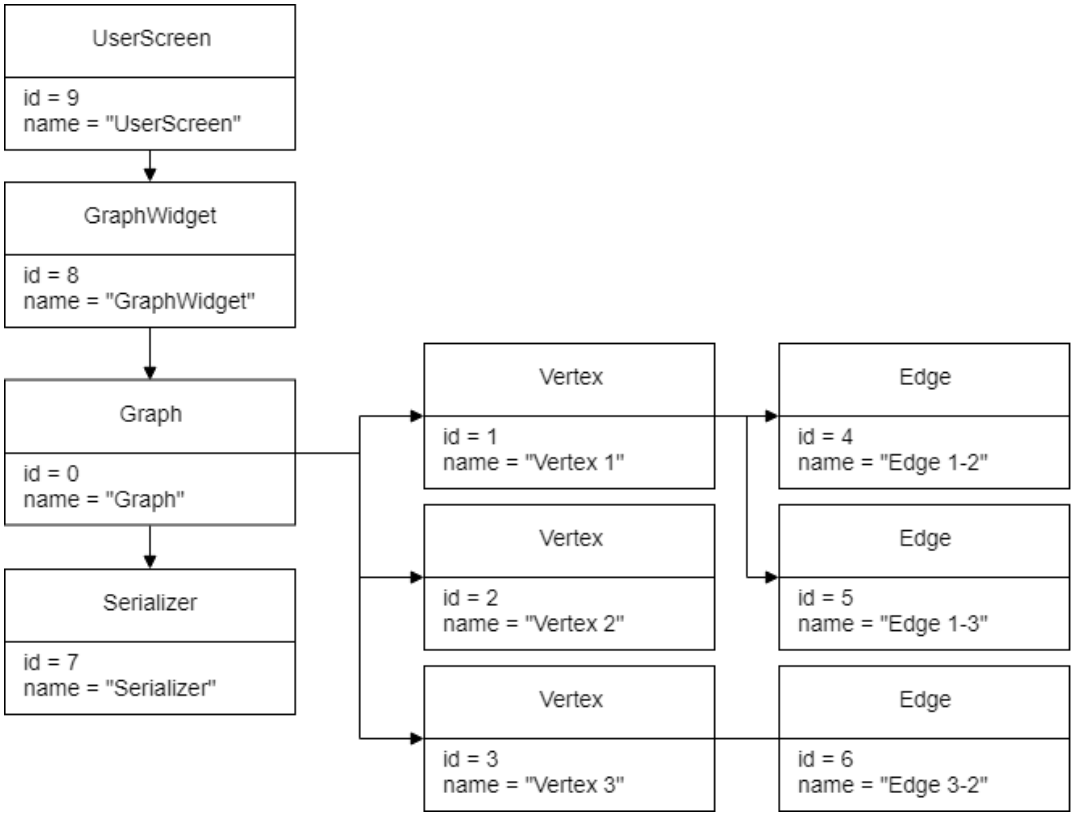


Диаграмма развертывания

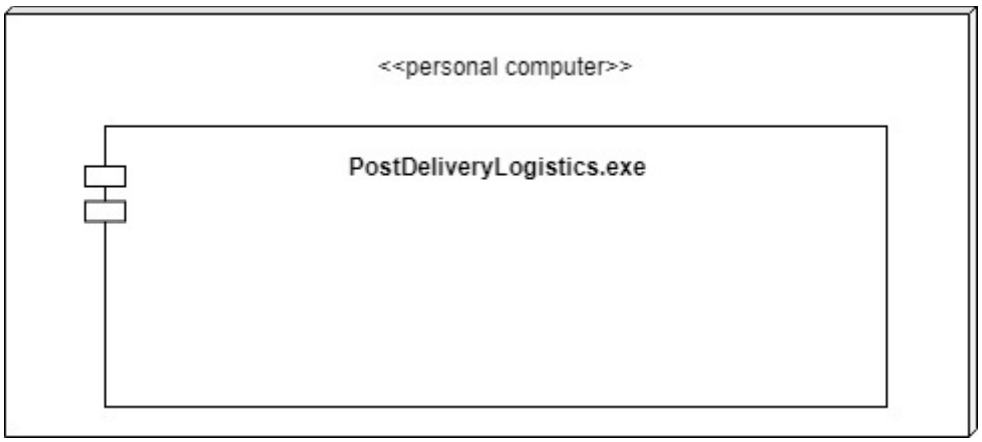


Диаграмма последовательностей

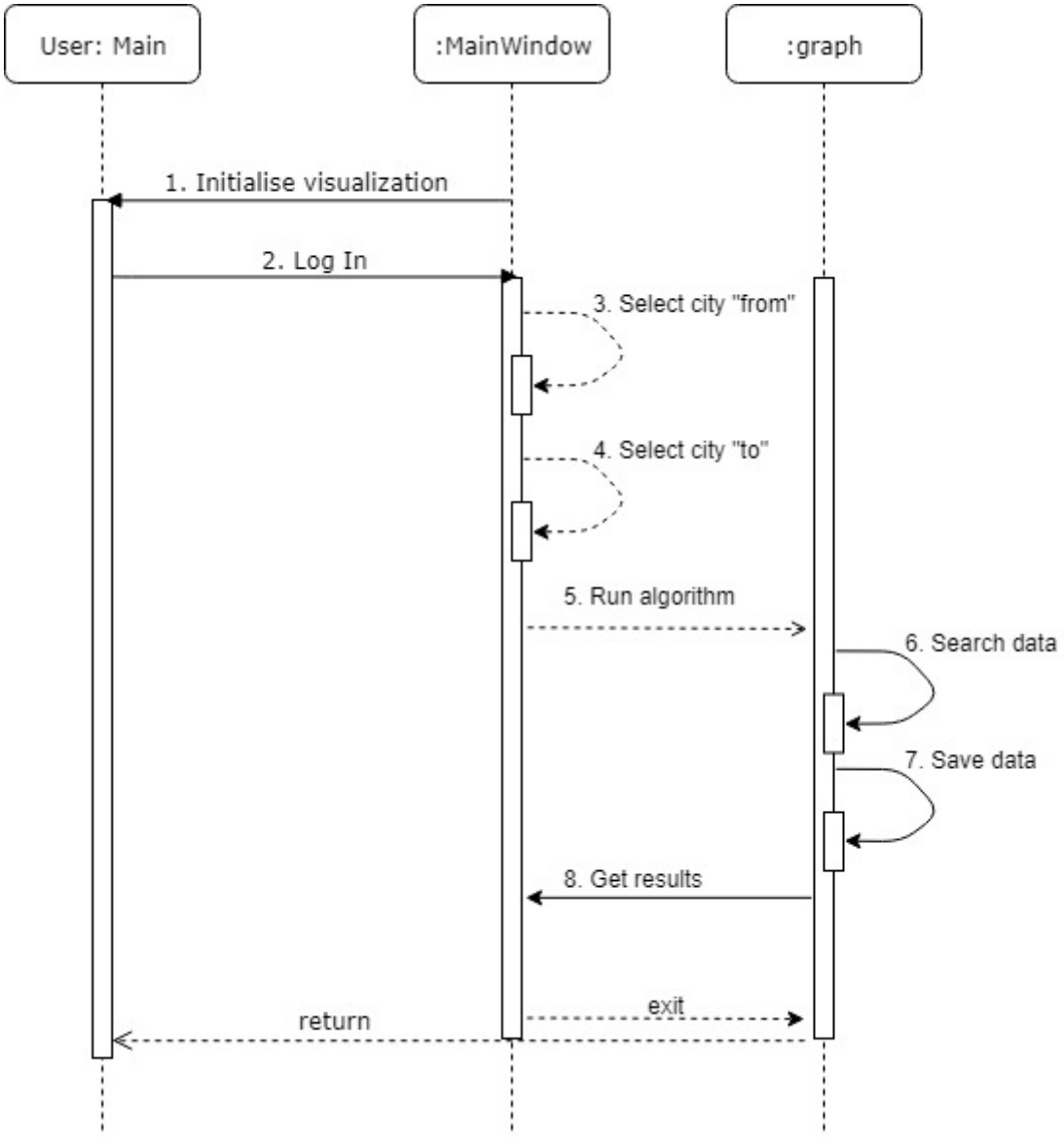


Диаграмма состояний

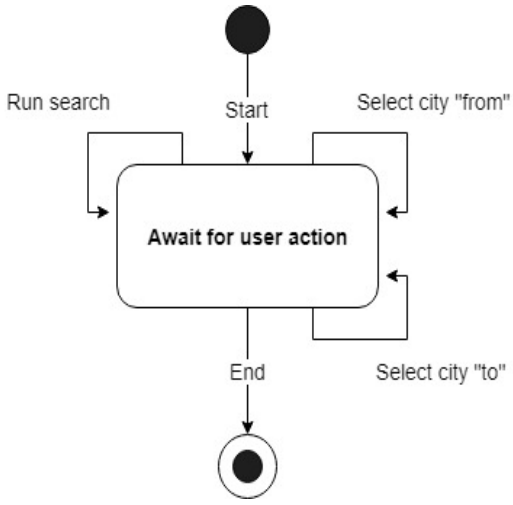


Диаграмма классов

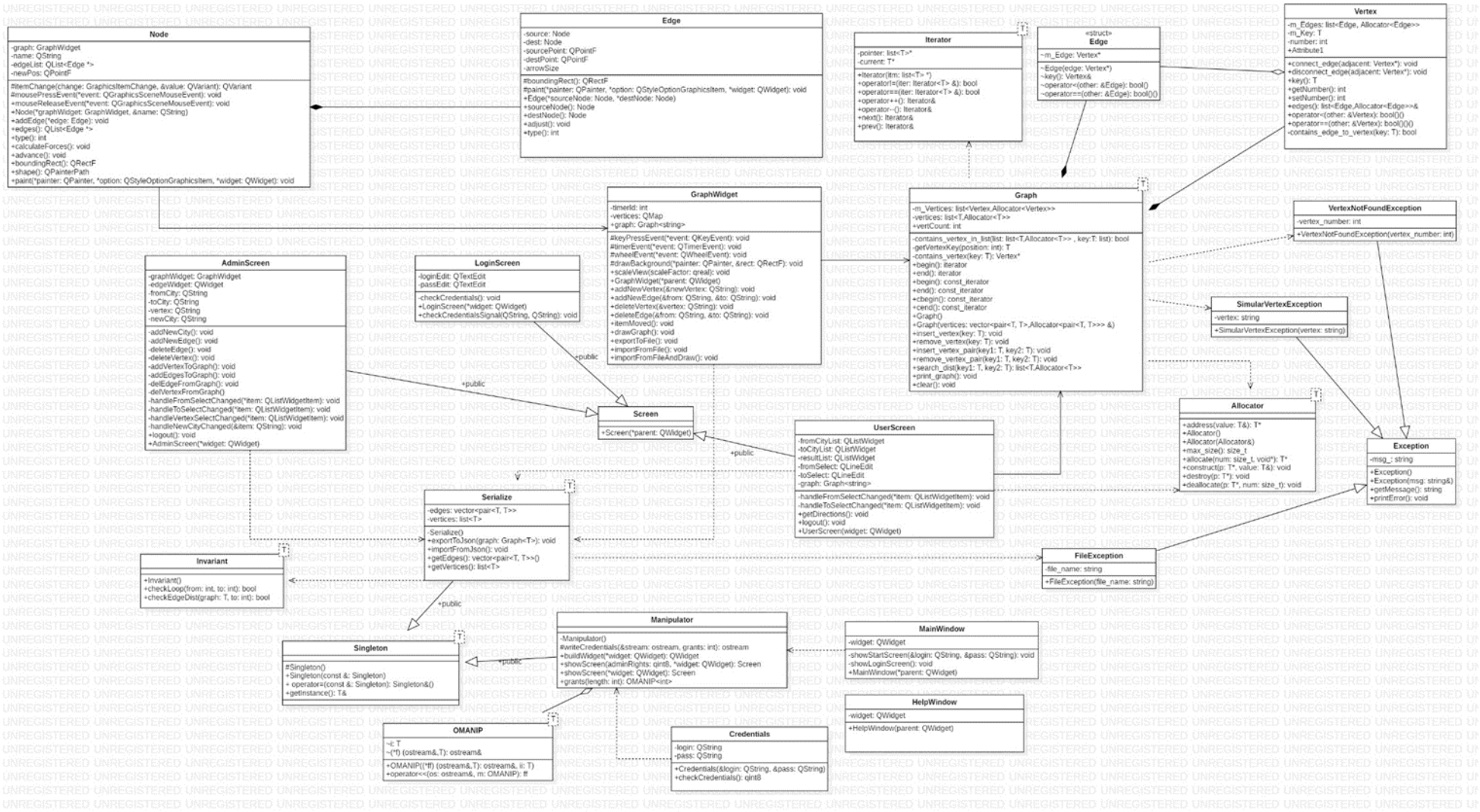
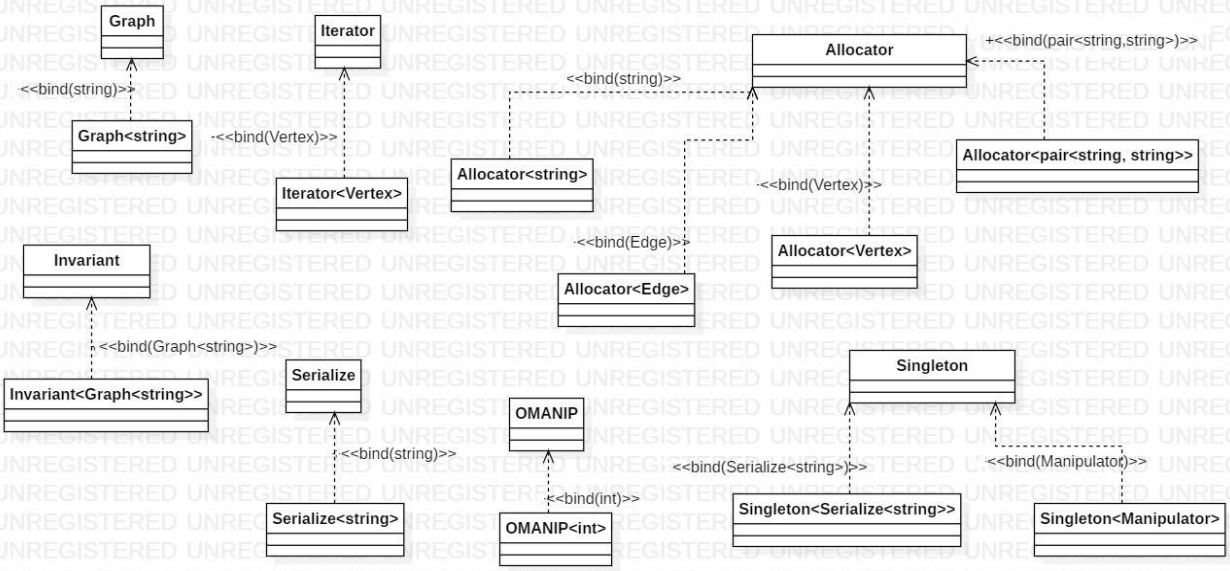


Диаграмма классов, развернутых из шаблонных



ПРИЛОЖЕНИЕ Е. РЕФАКТОРИНГ

Post Delivery Logistics

Рефакторинг контейнера

История изменений

Дата	Версия	Описание	Автор
21.05.19	1.0	Создание и заполнение документа	Дронников И.М.

Содержание

Введение

Цель

В этом документе представлен обзор изменения кода контейнера с целью оптимизации, улучшения качества и понимания кода.

Определение, акронимы и аббревиатуры

Определения, акронимы и аббревиатуры определены в артефакте Глоссарий.

Описание рефакторинга

- Удаление избыточного кода и лишних комментариев;
- Дата изменений: 28 мая 2019;
- Изменения: Удален избыточный код, затрудняющий чтение и понимание кода, удалены закомментированные функции и переменные;
- Оптимизация вызовов функций, обращения к классам;
- Участок кода, на котором произведен рефакторинг:

```
96  96          QFile file(path);
97      -       if (!file.open(QIODevice::ReadOnly)) {
98      -           throw FileException(path.toStdString());
99      -       }
        97  +   if (file.exists())
        98  +       if (!file.open(QIODevice::ReadOnly)) {
        99  +           throw FileException(path.toStdString());
        100 +       }
```

```
5       -   using namespace std;
6   5       class Exception {
7   6       public:
8   7           Exception() {printError();}
9       -       Exception(const string& msg) : msg_(msg) {printError();}
        8   +   Exception(const std::string& msg) : msg_(msg) {printError();}
10  9          ~Exception( ) {}
11      -       string getMessage( ) const {return(msg_);}
        10  +   std::string getMessage( ) const {return(msg_);}
12  11          void printError() {
13      -           cout << getMessage( ) << endl;
        12  +       std::cout << getMessage( ) << std::endl;
14  13          }
15  14      private:
16      -       string msg_;
        15  +   std::string msg_;
17  16      };
```

```
16      -
17      -       //Graph example usage
18      -       /*std::vector<std::pair<int, int> > graph_vect;
19      -       for (int i = 0; i < 100; i++) {
20      -           graph_vect.push_back(std::make_pair(rand()%20, rand()%20));
21      -       }*/
22      -       /*Graph<int> test_graph;
23      -
24      -       test_graph.insert_vertex_pair(1,200);
25      -       test_graph.insert_vertex_pair(1,5);
26      -       test_graph.insert_vertex_pair(1,6);
27      -       test_graph.insert_vertex_pair(10,1);
28      -       test_graph.insert_vertex_pair(11,1);
29      -       test_graph.insert_vertex_pair(40,200);
30      -       test_graph.insert_vertex_pair(50,200);
31      -
32      -       test_graph.print_graph();
33      -       Serializer::Serialize<int> *serialize = new Serializer::Serialize<int>();
34      -       serialize->exportToJson(test_graph);
35      -       serialize->importFromJson();
36      -       Graph<int> new_graph(serialize->getEdges());
37      -       new_graph.print_graph();
38      -       /*
39      -       test_graph.remove_vertex(1);
40      -
41      -       test_graph.print_graph();
42      -
```

```
13  12      void Ui::MainWindow::showLoginScreen() {
14          -   widget = manipulator->buildWidget(widget);
15          -   connect(manipulator->showScreen(widget), SIGNAL(checkCredentialsSignal(QString, QString)), this, SLOT(showStartScreen(QStr
        13  +   widget = Manip::Manipulator::getInstance().buildWidget(widget);
        14  +   connect(Manip::Manipulator::getInstance().showScreen(widget), SIGNAL(checkCredentialsSignal(QString, QString)), this, SLOT
16  15          this->setCentralWidget(widget);
17  16      }
18  17
19  18      void Ui::MainWindow::showStartScreen(const QString &login, const QString &pass) {
20  19          Credentials credentials(login, pass);
21          -   if (credentials.checkCredentials()==-1){
        20  +   qint8 grants = credentials.checkCredentials();
        21  +   if (grants == -1){
22  22              return;
23  23          }
24          -   widget = manipulator->buildWidget(widget);
25          -   Ui::Screen *screen = manipulator->showScreen(credentials.checkCredentials(), widget);
        24  +   widget = Manip::Manipulator::getInstance().buildWidget(widget);
        25  +   Ui::Screen *screen = Manip::Manipulator::getInstance().showScreen(grants, widget);
26  26          connect(screen, SIGNAL(logout()), this, SLOT(showLoginScreen()));
27  27          this->setCentralWidget(widget);
```

Оптимизация функций

- Дата изменений: 23 мая 2019
- Изменения: Исправлены ошибки алгоритма
- Участок кода, на котором произведен рефакторинг:

```
67  76      void Ui::UserScreen::getDirections()
68  77      {
69          -   //here put request for algorithm
70          -   resultList->addItem("MAGIK NOT WORKING YET! WAIT FOR IT.");
        78  +   resultList->clear();
        79  +   if (!fromSelect->text().isEmpty() && !fromSelect->text().isEmpty()) {
        80  +       std::list<std::string> result = graph.search_dist(fromSelect->text().toStdString(), toSelect->text().toStdSt
        81  +       typename std::list<std::string>::iterator print_it = result.begin();
        82  +       for(; print_it != result.end(); ++print_it) {
        83  +           resultList->addItem(QString::fromStdString(*print_it));
        84  +       }
        85  +   } else {
        86  +       resultList->addItem("Choose cities");
        87  +   }
```

История изменений

Дата	Версия	Описание	Автор
23.05.19	1.0	Создание и заполнение документа	Дронников И.М. Гордеева Т.В. Полушина Ю.И. Козловский А.О.

Содержание

Введение

Цель

В этом документе представлен обзор изменения кода интерфейса с целью оптимизации, улучшения качества и понимания кода, также повышения удобства интерфейса.

Определение, акронимы и аббревиатуры

Определения, акронимы и аббревиатуры определены в Глоссарии.

Описание рефакторинга

Удаление избыточного кода и лишних комментариев

• Дата изменений: 20 мая 2019.

• Изменения: Удален избыточный код, затрудняющий чтение и понимание кода.

• Участок кода, на котором произведен рефакторинг:

```
26  26      }
27      -   graph.print_graph();
28  27
29      -   typename std::list<Graph<int>::Vertex>::iterator add_nodes_it = graph.begin();
30      -   for(; add_nodes_it != graph.end(); ++add_nodes_it) {
31      -       Node *node = new Node(this, QString::number(add_nodes_it->key()));
32      -       vertices.insert(add_nodes_it->key(), node);
33      -       scene->addItem(node);
34      -   }
35      -
36      -   typename std::list<Graph<int>::Vertex>::iterator add_edges_it = graph.begin();
37      -   for(; add_edges_it != graph.end(); ++add_edges_it) {
38      -       Node *nodeFrom = vertices.value(add_edges_it->key());
39      -       typename std::list<Graph<int>::Edge>::const_iterator edge_it = add_edges_it->edges().begin();
40      -       for(; edge_it != add_edges_it->edges().end(); ++edge_it) {
41      -           Node *nodeTo = vertices.value(edge_it->m_Edge->key());
42      -           scene->addItem(new Edge(nodeFrom, nodeTo));
43      -           nodeFrom = nodeTo;
44      -       }
45      -   }
    28  +   graph.print_graph();
46  29
47      -   for(QMap<Graph<int>::Vertex, Node*>::iterator pos_it=vertices.begin(); pos_it!=vertices.end(); ++pos_it)
48      -   {
49      -       pos_it.value()->setPos(rand()%50, rand()%50);
50      -   }
    30  +   drawGraph();
51  31  }
```

ПРИЛОЖЕНИЕ Ж. ИСХОДНЫЙ КОД ПРОЕКТА

graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
#include <vector>
#include <list>
#include <QString>

#include "iterator.h"
#include "allocator.h"
#include "exceptions.h"

namespace GraphContainer
{

template <class T>
class Graph
{
public :
    Graph();
    Graph(const std::vector<std::pair<T, T>,Alloc::Allocator<std::pair<T, T>>>
&vertices);
    ~Graph(){}
    void insert_vertex(T key);
    void remove_vertex(T key);
    void insert_vertex_pair(T key1, T key2);
    void remove_vertex_pair(T key1, T key2);
    std::list<T,Alloc::Allocator<T>> search_dist(T key1, T key2);
    void print_graph();
    void clear();

    class Vertex;
    static int vertCount;

    struct Edge
    {
        Edge(Vertex *edge) :
            m_Edge(edge)
        {}
        Vertex *m_Edge;

        const Vertex &key() const {return *m_Edge;}

        bool operator< (const Edge &other) const {
            return key() < other.key();
        }

        bool operator== (const Edge &other) const {
            return key() == other.key();
        }
    };

    class Vertex
    {
    public:
        Vertex(T key) :
            m_Key(key)
        { setNumber(++vertCount); }
```

```
        void connect_edge(Vertex *adjacent);
        void disconnect_edge(Graph<T>::Vertex *adjacent);
        const T key() const {return m_Key;}
        int getNumber() {return number;}
        void setNumber(int n) {number = n;}
        const std::list<Edge,Alloc::Allocator<Edge>> &edges() const {return
m_Edges;}
        bool operator< (const Vertex &other) const {
            return key() < other.key();
        }

        bool operator== (const Vertex &other) const {
            return key() == other.key();
        }

    private:
        std::list<Edge,Alloc::Allocator<Edge>> m_Edges;
        T m_Key;
        int number;
        bool contains_edge_to_vertex(const T key);
    };

private:
    using vertices_type = std::list<Vertex,Alloc::Allocator<Vertex>>;

public:
    using iterator = typename vertices_type::iterator;
    using const_iterator = typename vertices_type::const_iterator;
    iterator begin() { return m_Vertices.begin(); }
    iterator end() { return m_Vertices.end(); }
    const_iterator begin() const { return m_Vertices.begin(); }
    const_iterator end() const { return m_Vertices.end(); }
    const_iterator cbegin() const { return m_Vertices.cbegin(); }
    const_iterator cend() const { return m_Vertices.cend(); }

private:
    vertices_type m_Vertices;
    Vertex *contains_vertex(const T key);
    bool contains_vertex_in_list(std::list<T,Alloc::Allocator<T>> list, const T
key);
    T getVertexKey(int position)
    {
        typename std::list<Vertex,Alloc::Allocator<Vertex>>::iterator find_it =
m_Vertices.begin();
        for(; find_it != m_Vertices.end(); ++find_it) {
            if (find_it->getNumber() == position) {
                return find_it->key();
            }
        }
        return NULL;
    }
    std::list<T,Alloc::Allocator<T>> vertices;
};
}

template <class T>
int GraphContainer::Graph<T>::vertCount = -1;

template <class T>
GraphContainer::Graph<T>::Graph()
{
}
```

```
template <class T>
GraphContainer::Graph<T>::Graph(const std::vector<std::pair<T,
T>,Alloc::Allocator<std::pair<T, T>>> &vertices_relation)
{
    typename std::vector<std::pair<T,
T>,Alloc::Allocator<std::pair<T,T>>>::const_iterator insert_it =
vertices_relation.begin();
    for(; insert_it != vertices_relation.end(); ++insert_it) {
        insert_vertex_pair(insert_it->first, insert_it->second);
    }
}

/*!
 * Takes in a value of type T as a key and
 * inserts it into graph data structure if
 * key not already present
 */
template <typename T>
void GraphContainer::Graph<T>::insert_vertex(T key)
{
    /*!
   * Check if vertex already in graph
   */
    Graph<T>::Vertex *insert = contains_vertex(key);
    /*!
   * If not in graph then insert it and get a pointer to it
   * to pass into edge. See () for information on how
   * to build graph
   */
    if (insert == NULL) {
        m_Vertices.push_back(Vertex(key));
        insert = contains_vertex(key);
    }
    /*!
   * At this point we should have a Vertex in graph
   * if not throw an error.
   */
    if (insert == NULL) {
        throw std::runtime_error("Unknown");
    }
}

/*!
 * Takes in a value of type T as a key and
 * inserts it into graph data structure if
 * key not already present
 */
template <typename T>
void GraphContainer::Graph<T>::insert_vertex_pair(T key1, T key2)
{
    /*!
   * Check if vertices are simular
   */
    if (key1 == key2) {
        throw SimularVertexException(key1);
    }
    /*!
   * Check if vertices already in graph
   */
    Graph<T>::Vertex *insert1 = contains_vertex(key1);
    Graph<T>::Vertex *insert2 = contains_vertex(key2);
```

```
    /*!
   * If not in graph then insert it and get a pointer to it
   * to pass into edge. See () for information on how
   * to build graph
   */
    if (insert1 == NULL) {
        m_Vertices.push_back(Vertex(key1));
        insert1 = contains_vertex(key1);
    }
    if (insert2 == NULL) {
        m_Vertices.push_back(Vertex(key2));
        insert2 = contains_vertex(key2);
    }

#ifndef NDEBUG
    assert(insert1 != NULL && "Failed to insert first vertex");
    assert(insert2 != NULL && "Failed to insert second vertex");
#endif
    /*!
   * At this point we should have a Vertex to insert an edge on
   * if not throw an error.
   */
    if (insert1 != NULL && insert2 != NULL) {
        insert1->connect_edge(insert2);

#ifndef NDEBUG
        std::cout <<"Inserting pair: " <<  insert1->key() << " -- > " <<
insert2->key() <<
                    std::endl;
#endif
        //insert2->connect_edge(insert1);
    } else {
        throw std::runtime_error("Unknown");
    }
}

/*!
 * Takes in a value of type T as a key and
 * removes it from graph data structure if
 * key present
 */
template <typename T>
void GraphContainer::Graph<T>::remove_vertex(T key)
{
    /*!
   * Check if vertex exist in graph
   */
    Graph<T>::Vertex *vertex = contains_vertex(key);

#ifndef NDEBUG
    assert(vertex != NULL && "Failed to find vertex");
#endif

    /*!
   * At this point we should have a Vertex to remove
   * if not throw an error.
   */
    if (vertex != NULL) {
        /*!
         * Removing edges from vertex
         */
```

```
        typename std::list<Edge,Alloc::Allocator<Edge>>::const_iterator edge_it
= vertex->edges().begin();
        for(; edge_it != vertex->edges().end(); ++edge_it) {
            remove_vertex_pair(vertex->key(), edge_it->m_Edge->key());
        }
        /*!
         * Removing edges to vertex
         */
        typename std::list<Vertex,Alloc::Allocator<Vertex>>::iterator vert_it =
m_Vertices.begin();
        for(; vert_it != m_Vertices.end(); ++vert_it) {
            typename std::list<Edge,Alloc::Allocator<Edge>>::const_iterator
edge_it = vert_it->edges().begin();
            for(; edge_it != vert_it->edges().end(); ++edge_it) {
                if (edge_it->m_Edge->key() == vertex->key()) {
                    remove_vertex_pair(vert_it->key(), vertex->key());
                }
            }
        }
        /*!
         * Removing vertex
         */
        m_Vertices.remove(vertex->key());
    } else {
        throw std::runtime_error("Unknown");
    }
}

/*!
 * Takes in a value of type T as a key and
 * removes it from graph data structure if
 * key present
 */
template <typename T>
void GraphContainer::Graph<T>::remove_vertex_pair(T key1, T key2)
{
    /*!
   * Check if vertices are simular
   */
    if (key1 == key2) {
        //throw std::runtime_error("Simular Vertices!");
        throw SimularVertexException(key1);
    }
    /*!
   * Check if vertices already in graph
   */
    Graph<T>::Vertex *vertex1 = contains_vertex(key1);
    Graph<T>::Vertex *vertex2 = contains_vertex(key2);

//#ifndef NDEBUG
//    assert(vertex1 != NULL && "Failed to find pair (first vertex not
exists)");
//    assert(vertex2 != NULL && "Failed to find pair (second vertex not
exists)");
//#endif
    if(vertex1 != NULL) {
        throw VertexNotFoundException(vertex1->getNumber());
    }
    if(vertex2 != NULL) {
        throw VertexNotFoundException(vertex2->getNumber());
    }
    /*!
```

```
    * At this point we should have a Vertex to insert an edge on
    * if not throw an error.
    */
     if (vertex1 != NULL && vertex2 != NULL) {
         vertex1->disconnect_edge(vertex2);
         //insert2->disconnect_edge(insert1);
     } else {
         throw std::runtime_error("Unknown");
     }
}

/*!
 * Search the std::list of vertices for key
 * if present return the Vertex to indicate
 * already in graph else return NULL to indicate
 * new node
 */
template <typename T>
typename GraphContainer::Graph<T>::Vertex
*GraphContainer::Graph<T>::contains_vertex(T key)
{
    typename std::list<Vertex,Alloc::Allocator<Vertex>>::iterator find_it =
m_Vertices.begin();
    for(; find_it != m_Vertices.end(); ++find_it) {
        if (find_it->key() == key) {
            return &(*find_it);
        }
    }
    return NULL;
}

/*!
 * Take the oposing Vertex from input and insert it
 * into adjacent list, you can have multiple edges
 * between vertices
 */
template <class T>
void GraphContainer::Graph<T>::Vertex::connect_edge(Graph<T>::Vertex *adjacent)
{
    if (adjacent == NULL)
        return;

    if (!contains_edge_to_vertex(adjacent->key())) {
        Graph<T>::Edge e(adjacent);
        m_Edges.push_back(e);
    }
}

/*!
 * Take the oposing Vertex from input and remove it
 * from adjacent list
 */
template <class T>
void GraphContainer::Graph<T>::Vertex::disconnect_edge(Graph<T>::Vertex
*adjacent)
{
    if (adjacent == NULL)
        return;

    if (contains_edge_to_vertex(adjacent->key())) {
        Graph<T>::Edge e(adjacent);
        m_Edges.remove(e);
```

```
    }
}

/*!
 * Private member function that check if there is already
 * an edge between the two vertices
 */
template <class T>
bool GraphContainer::Graph<T>::Vertex::contains_edge_to_vertex(const T key)
{
    typename std::list<Edge,Alloc::Allocator<Edge>>::iterator find_it =
m_Edges.begin();
    for(; find_it != edges().end(); ++find_it) {
        if (find_it->m_Edge->key() == key) {
            return true;
        }
    }
    return false;
}

template <typename T>
void GraphContainer::Graph<T>::print_graph()
{
    std::cout << "Graph: " << std::endl;
    typename std::list<Vertex,Alloc::Allocator<Vertex>>::iterator print_it =
m_Vertices.begin();
    for(; print_it != m_Vertices.end(); ++print_it) {
        std::cout << print_it->key();
        typename std::list<Edge,Alloc::Allocator<Edge>>::const_iterator edge_it
= print_it->edges().begin();
        for(; edge_it != print_it->edges().end(); ++edge_it) {
            std::cout << "-->" << edge_it->m_Edge->key();
        }
        std::cout << std::endl;
    }
}

template <typename T>
void GraphContainer::Graph<T>::clear()
{
    m_Vertices.clear();
}

template <typename T>
typename std::list<T,Alloc::Allocator<T>>
GraphContainer::Graph<T>::search_dist(T key1, T key2)
{
    Graph<T>::Vertex *from = contains_vertex(key1);
    Graph<T>::Vertex *to = contains_vertex(key2);
    vertices.clear();
    int N = m_Vertices.size();

    std::vector < std::vector <int> > b(N, std::vector <int> (N) );
    std::vector < std::vector <int> > c(N, std::vector <int> (N) );

    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++) {b[i][j]=999999;b[i][i]=999999;c[i][j]=-1;}
    typename std::list<Vertex,Alloc::Allocator<Vertex>>::iterator print_it =
m_Vertices.begin();
    for(; print_it != m_Vertices.end(); ++print_it) {
        typename std::list<Edge,Alloc::Allocator<Edge>>::const_iterator edge_it
= print_it->edges().begin();
```

```
        for(; edge_it != print_it->edges().end(); ++edge_it) {
            b[print_it->getNumber()][edge_it->m_Edge->getNumber()] = 1;
        }

    }
    for(int k=0;k<N;k++)
        for(int i=0;i<N;i++)
            for(int j=0;j<N;j++){
                int a=b[i][k]+b[k][j];
                if(b[i][j]>a)
                {
                    c[i][j]=k;
                    b[i][j]=a;
                }
            }
    int i=from->getNumber(),j=to->getNumber();
    while(c[i][j]!=-1){
        vertices.push_front(getVertexKey(c[i][j]));
        j=c[i][j];
    }
    i=from->getNumber(),j=to->getNumber();
    while(c[i][j]!=-1){
        if( !contains_vertex_in_list(vertices, getVertexKey(c[i][j]))){
            vertices.push_back(getVertexKey(c[i][j]));
        }
        i=c[i][j];
    }
    vertices.push_front(getVertexKey(from->getNumber()));
    vertices.push_back(getVertexKey(to->getNumber()));
    return vertices;
}

template <typename T>
bool
GraphContainer::Graph<T>::contains_vertex_in_list(std::list<T,Alloc::Allocator<T
>> list, T key)
{
    typename std::list<std::string,Alloc::Allocator<std::string>>::iterator
find_it = list.begin();
    for(; find_it != list.end(); ++find_it) {
        if (find_it->data() == key) {
            return true;
        }
    }
    return false;
}

#endif // GRAPH_H
```

allocator.h

```
#ifndef ALLOCATOR_H
#define ALLOCATOR_H

#include <iostream>
#include <limits>

namespace Alloc {
    template <class T>
    class Allocator {
    public:
```

```
    typedef T value_type;
    typedef T* pointer;
    typedef const T* const_pointer;
    typedef T& reference;
    typedef const T& const_reference;
    typedef std::size_t size_type;
    typedef std::ptrdiff_t difference_type;

    template <class U>
    struct rebind {
        typedef Allocator<U> other;
    };

    T* address (T& value) const {
        return &value;
    }
    T* address (const T& value) const {
        return &value;
    }

    Allocator() throw() {
    }
    Allocator(const Allocator&) throw() {
    }
    template <class U>
      Allocator (const Allocator<U>&) throw() {
    }
    ~Allocator() throw() {
    }

    std::size_t max_size () const throw() {
        return std::numeric_limits<std::size_t>::max() / sizeof(T);
    }

    T* allocate (std::size_t num, const void* = 0) {
        T* ret = (T*)(::operator new(num*sizeof(T)));
        return ret;
    }

    void construct (T* p, const T& value) {
        new((void*)p)T(value);
    }

    void destroy (T* p) {
        p->~T();
    }

    void deallocate (T* p, std::size_t num) {
        ::operator delete((void*)p);
    }
 };

template <class T1, class T2>
bool operator== (const Allocator<T1>&,
                 const Allocator<T2>&) throw() {
    return true;
}
template <class T1, class T2>
bool operator!= (const Allocator<T1>&,
                 const Allocator<T2>&) throw() {
    return false;
}
```

```
}

#endif // ALLOCATOR_H
```

manipulator.h

```
#ifndef MANIPULATOR_H
#define MANIPULATOR_H

#include <QWidget>
#include <iostream>
#include <iomanip>

#include "ui/screen.h"
#include "ui/adminscreen.h"
#include "ui/userscreen.h"
#include "ui/loginscreen.h"
#include "singleton.h"

namespace Manip {

template<class T>
class OMANIP {
    T i;
    std::ostream& (*f) (std::ostream&,T);
public:
    OMANIP(std::ostream& (*ff) (std::ostream&,T), T ii)
        : f(ff), i(ii) { }

    friend std::ostream& operator<<(std::ostream& os, OMANIP m)
    { return (*m.f)(os,m.i); }
};

class Manipulator : public Singleton<Manipulator>
{

protected:
    friend std::ostream &writeCredentials(std::ostream &stream, int grants);
public:
    QWidget *buildWidget(QWidget *widget);
    Ui::Screen *showScreen(qint8 adminRights, QWidget *widget);
    Ui::Screen *showScreen(QWidget *widget);
    Manip::OMANIP<int> grants(int length);

private:
    friend Manip::OMANIP<int>;
    friend class Singleton<Manipulator>;
    Manipulator();
};

}
#endif // MANIPULATOR_H
```

manipulator.cpp

```
#include "manipulator.h"

Manip::Manipulator::Manipulator()
{
```

```
}

Ui::Screen *Manip::Manipulator::showScreen(qint8 adminRights, QWidget *widget) {
    switch (adminRights) {
    case 0: {
        return new Ui::UserScreen(widget);
    }
    case 1: {
        return new Ui::AdminScreen(widget);
    }
    case 2: {
        return new Ui::LoginScreen(widget);
    }
    }
    return nullptr;
}

Ui::Screen *Manip::Manipulator::showScreen(QWidget *widget) {
    return Manip::Manipulator::showScreen(qint8(2),widget);
}

QWidget *Manip::Manipulator::buildWidget(QWidget *widget){
    if (widget != nullptr) {
        delete widget;
        widget = nullptr;
    }

    return new QWidget();
}

std::ostream &Manip::writeCredentials(std::ostream& stream, int grants)
{
    switch (grants) {
    case (0): {
        stream << "Welcome, user!"<<std::endl;
        return stream;
    }
    case (1): {
        stream << "Welcome, admin!"<<std::endl;
        return stream;
    }
    }
    return stream;
}

Manip::OMANIP<int> Manip::Manipulator::grants(int grants)
{
    return Manip::OMANIP<int>(writeCredentials,grants);
}
```

singleton.h

```
#ifndef SINGLETON_H
#define SINGLETON_H

#include <memory>

template <typename T>
class Singleton
```

```
{
protected:
    Singleton() {}
    ~Singleton() {}
public:
    Singleton(Singleton const &) = delete;
    Singleton& operator=(Singleton const &) = delete;

    static T& getInstance()
    {
        static T instance;
        return instance;
    }
};

#endif // SINGLETON_H
```

graphwidget.h

```
#ifndef GRAPHWIDGET_H
#define GRAPHWIDGET_H

#include <QGraphicsView>
#include <QMap>
#include <QMessageBox>

#include "graph.h"
#include "serialize.h"

using GraphContainer::Graph;
using Serializer::Serialize;

namespace GraphWidgetUi {
class Node;

class GraphWidget : public QGraphicsView
{
    Q_OBJECT

public:
    GraphWidget(QWidget *parent = 0);

    void addNewVertex(const QString &newVertex);
    void addNewEdge(const QString &from, const QString &to);
    void deleteVertex(const QString &vertex);
    void deleteEdge(const QString &from, const QString &to);
    void itemMoved();
    void drawGraph();

public slots:
    void exportToFile();
    void importFromFile();
    void importFromFileAndDraw();

protected:
    void keyPressEvent(QKeyEvent *event);
    void timerEvent(QTimerEvent *event);
    void wheelEvent(QWheelEvent *event);
    void drawBackground(QPainter *painter, const QRectF &rect);

    void scaleView(qreal scaleFactor);
```

```
private:
    int timerId;
    QMap<Graph<std::string>::Vertex, Node*> vertices;
    Graph<std::string> graph;
};
}

#endif
```

graphwidget.cpp

```
#include "graphwidget.h"

#include <QtGui>

#include <math.h>
#include "edge.h"
#include "node.h"

GraphWidgetUi::GraphWidget::GraphWidget(QWidget *parent)
    : QGraphicsView(parent), timerId(0)
{
    QGraphicsScene *scene = new QGraphicsScene(this);
    scene->setItemIndexMethod(QGraphicsScene::NoIndex);
    scene->setSceneRect(-200, -310, 400, 510);
    setScene(scene);
    setCacheMode(CacheBackground);
    setViewportUpdateMode(BoundingRectViewportUpdate);
    setRenderHint(QPainter::Antialiasing);
    setTransformationAnchor(AnchorUnderMouse);
    //scale(qreal(0.8), qreal(0.8));
    adjustSize();
    setMinimumSize(400, 400);

    importFromFileAndDraw();
}

void GraphWidgetUi::GraphWidget::addNewVertex(const QString &newVertex)
{
    graph.insert_vertex(newVertex.toStdString());
    Serialize<std::string>::getInstance().exportToJson(graph);
    drawGraph();
}

void GraphWidgetUi::GraphWidget::addNewEdge(const QString &from, const QString
&to)
{
    graph.insert_vertex_pair(from.toStdString(), to.toStdString());
    Serialize<std::string>::getInstance().exportToJson(graph);
    drawGraph();
}

void GraphWidgetUi::GraphWidget::deleteEdge(const QString &from, const QString
&to)
{
    graph.remove_vertex_pair(from.toStdString(), to.toStdString());
    Serialize<std::string>::getInstance().exportToJson(graph);
    drawGraph();
}
```

```
void GraphWidgetUi::GraphWidget::deleteVertex(const QString &vertex)
{
    graph.remove_vertex(vertex.toStdString());
    Serialize<std::string>::getInstance().exportToJson(graph);
    drawGraph();
}

void GraphWidgetUi::GraphWidget::drawGraph()
{
    scene()->clear();

    typename std::list<Graph<std::string>::Vertex>::iterator add_nodes_it =
graph.begin();
    for(; add_nodes_it != graph.end(); ++add_nodes_it) {
        Node *node = new Node(this, QString::fromStdString(add_nodes_it-
>key()));
        vertices.insert(add_nodes_it->key(), node);
        scene()->addItem(node);
    }

    typename std::list<Graph<std::string>::Vertex>::iterator add_edges_it =
graph.begin();
    for(; add_edges_it != graph.end(); ++add_edges_it) {
        Node *nodeFrom = vertices.value(add_edges_it->key());
        typename std::list<Graph<std::string>::Edge>::const_iterator edge_it =
add_edges_it->edges().begin();
        for(; edge_it != add_edges_it->edges().end(); ++edge_it) {
            Node *nodeTo = vertices.value(edge_it->m_Edge->key());
            scene()->addItem(new Edge(nodeFrom, nodeTo));
        }
    }

    for(QMap<Graph<std::string>::Vertex, Node*>::iterator
pos_it=vertices.begin(); pos_it!=vertices.end(); ++pos_it)
    {
        pos_it.value()->setPos(rand()%50, rand()%50);
    }
}

void GraphWidgetUi::GraphWidget::exportToFile()
{
    Serialize<std::string>::getInstance().exportToJson(graph);
    QMessageBox::information(this, "Done!", "Export done.", QMessageBox::Ok, 0);
}

void GraphWidgetUi::GraphWidget::importFromFile()
{
    Serialize<std::string>::getInstance().importFromJson();
    graph.clear();
    std::list<std::string> vertices =
Serialize<std::string>::getInstance().getVertices();
    std::vector<std::pair<std::string, std::string>> edges =
Serialize<std::string>::getInstance().getEdges();
    for (std::string it: vertices) {
        graph.insert_vertex(it);
    }
    for (std::pair<std::string, std::string> it: edges) {
        graph.insert_vertex_pair(it.first, it.second);
    }
}

void GraphWidgetUi::GraphWidget::importFromFileAndDraw()
```

```
{
    importFromFile();
    drawGraph();
}

void GraphWidgetUi::GraphWidget::itemMoved()
{
    if (!timerId)
        timerId = startTimer(1000 / 25);
}

void GraphWidgetUi::GraphWidget::keyPressEvent(QKeyEvent *event)
{
    switch (event->key()) {
    case Qt::Key_Up:
        //centerNode->moveBy(0, -20);
        break;
    case Qt::Key_Down:
        //centerNode->moveBy(0, 20);
        break;
    case Qt::Key_Left:
        //centerNode->moveBy(-20, 0);
        break;
    case Qt::Key_Right:
        //centerNode->moveBy(20, 0);
        break;
    case Qt::Key_Plus:
        scaleView(qreal(1.2));
        break;
    case Qt::Key_Minus:
        scaleView(1 / qreal(1.2));
        break;
    case Qt::Key_Space:
    case Qt::Key_Enter:
        foreach (QGraphicsItem *item, scene()->items()) {
            if (qgraphicsitem_cast<Node *>(item))
                item->setPos(-150 + qrand() % 300, -150 + qrand() % 300);
        }
        break;
    default:
        QGraphicsView::keyPressEvent(event);
    }
}

void GraphWidgetUi::GraphWidget::timerEvent(QTimerEvent *event)
{
    Q_UNUSED(event);

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items()) {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            nodes << node;
    }

    foreach (Node *node, nodes)
        node->calculateForces();

    bool itemsMoved = false;
    foreach (Node *node, nodes) {
        if (node->advance())
            itemsMoved = true;
    }
```

```
    if (!itemsMoved) {
        killTimer(timerId);
        timerId = 0;
    }
}

void GraphWidgetUi::GraphWidget::wheelEvent(QWheelEvent *event)
{
    scaleView(pow((double)2, -event->delta() / 240.0));
}

void GraphWidgetUi::GraphWidget::drawBackground(QPainter *painter, const QRectF
&rect)
{
    Q_UNUSED(rect);

    QRectF sceneRect = this->sceneRect();
    QLinearGradient gradient(sceneRect.topLeft(), sceneRect.bottomRight());
    gradient.setColorAt(0, Qt::white);
    gradient.setColorAt(1, Qt::lightGray);
    painter->fillRect(rect.intersected(sceneRect), gradient);
    painter->setBrush(Qt::NoBrush);
    painter->drawRect(sceneRect);
}

void GraphWidgetUi::GraphWidget::scaleView(qreal scaleFactor)
{
    qreal factor = transform().scale(scaleFactor, scaleFactor).mapRect(QRectF(0,
0, 1, 1)).width();
    if (factor < 0.07 || factor > 100)
        return;

    scale(scaleFactor, scaleFactor);
}
```