Государственное бюджетное профессиональное образовательное учреждение Ростовской области

«Таганрогский Колледж Морского Приборостроения»

|  |
| --- |
| Автоматизированная система проектирования плана города |
|  |
| Курсовой проект |
| Пояснительная записка |
| МДК.02.01.13.001 ПЗ |

|  |
| --- |
| Руководитель:  *Адардасов А.А* |
|  |

Студент П-319

*Ларионов М.Ю.*

2022

Содержание

[Введение 3](#_Toc105113364)

[1 Постановка задачи 4](#_Toc105113365)

[2 Проектирование 6](#_Toc105113366)

[3 Программирование 11](#_Toc105113367)

[4 Отладка, оптимизация и тестирование программы 13](#_Toc105113368)

[Заключение 14](#_Toc105113369)

[Список использованных источников 15](#_Toc105113370)

[ПРИЛОЖЕНИЕ А 16](#_Toc105113371)

[ПРИЛОЖЕНИЕ Б 22](#_Toc105113372)

# **Введение**

Городские системы — это не только структурно-конструкционные образования. Еще в конце XX века Д. Харвей указывал, что современные города являются сложными организмами, чем и объясняются трудности их изучения и проектирования. В России особенно остро стоит задача совершенствования функционально-планировочного и социально-территориального проектирования крупных городов и формирующихся в зонах их влияния систем расселения.

Проектирование можно определить как конструктивную деятельность по разработке целей и задач устойчивого развития городских систем. К проектированию также относится выявление и анализ проблем, связанных с развитием городских систем, и поиск решений. Проектная деятельность, по сути, инновационная, она предполагает пространственно-территориальные преобразования и основывается на использовании современных технологий, постоянно совершенствуясь по мере ее реализации.

Предметной областью курсовой работы является автоматизация проектирования плана города. Автоматизация данной предметной области заключается в создании программного обеспечения, которое позволит проектировать план города с использованием библиотеки условных обозначений, а также используя функции масштабирования, добавления и удаления элементов.

1. **Постановка задачи**

Назначение задачи:

По заданию требуется:разработать автоматизированную систему проектирования плана города для работы с данными, описанных в техническом задании.

Требования к программе:

Программа должна быть реализована на языке программирования “JavaScript”, используется программная платформа Electron для кроссплатформенной разработки (фреймворк), используется система управления базами данных “MySQL” и библиотека mxGraph для реализации функций рисования.

Функциональные характеристики:

Система должна обеспечивать следующий ряд функций ряд функций:

– добавление элементов;

– удаление элементов;

– масштабирование.

В программе необходимо предусмотреть возможность добавления и удаления элементов, масштабирование, выбор формата листа.

Через интерфейс должны быть реализованы следующие возможности:

Функции: добавление элементов, удаление элементов, масштабирование.

Надёжность и безопасность:

Разрабатываемое программное обеспечение должно иметь возможность:

– самовосстановления после сбоев программы;

– ограничение несанкционированного доступа к данным;

– возможность резервного копирования данных;

– предусмотреть контроль вводимой информации и блокировку некорректных действие пользователя при работе с системой.

Требования к составу и параметрам технических средств:

Программа должна работать на операционных системах семейства Windows; необходимый объем 2 Гб дискового пространства, минимальная мощность процессора должна составлять не менее 1.6 Гц, рекомендуемый объем оперативной памяти должен составлять не менее 4 Гб, расширение монитора 1920x1080.

# **Проектирование**

В данном разделе описывается процесс [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), включающий в себя сбор [требований к программному обеспечению](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%BC%D1%83_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8E), их систематизацию, выявление взаимосвязей. В этом разделе приводится структурная схема и функциональная схема программы, отражающие состав и взаимодействие по управлению частей разработанной программы, диаграмма вариантов использования, диаграмма последовательности.

Решение должно включать в себя модуль обработки данных и модуль интерфейса, модуль для работы с базой данных.

Модуль работы с базой данных включает в себя функции записи и чтения, а также переменные и функции, позволяющие ему обмениваться данными с модулем обработки данных.

Модуль обработки данных получает данные из базы данных, а также из модуля работы с графических интерфейсом.

Модуль для работы с графическим интерфейсом получает данные введенные пользователем и отправляет в модуль обработки данных.

Проектирование и реализация структурной схемы.

Для разработки программы была разработана структурная схема, представленная в соответствии с рисунком 2.1. Данная схема отображает основной набор функций, необходимый для корректной работы программы.

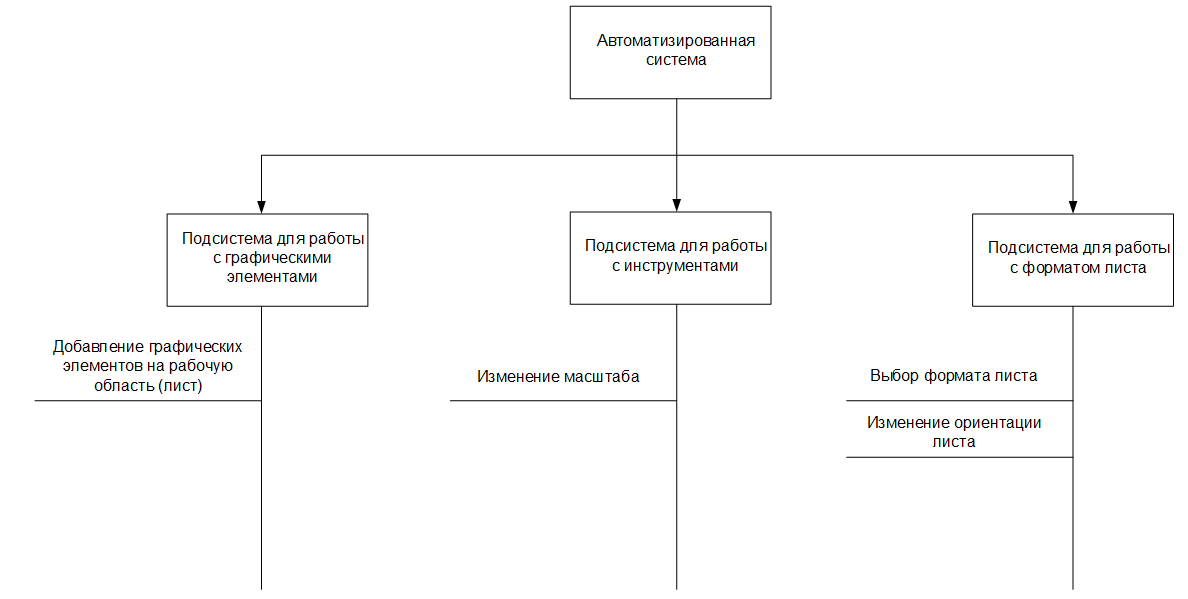


Рисунок 2.1 — Структурная схема

Проектирование и реализация функциональной схемы.

Для определения основных функций программы была разработана функциональная схема, представленная в соответствии с рисунком 2.2. Данная схема отображает процессы, происходящие на разных участках программы, иллюстрируя в целом весь процесс выполнения программы.

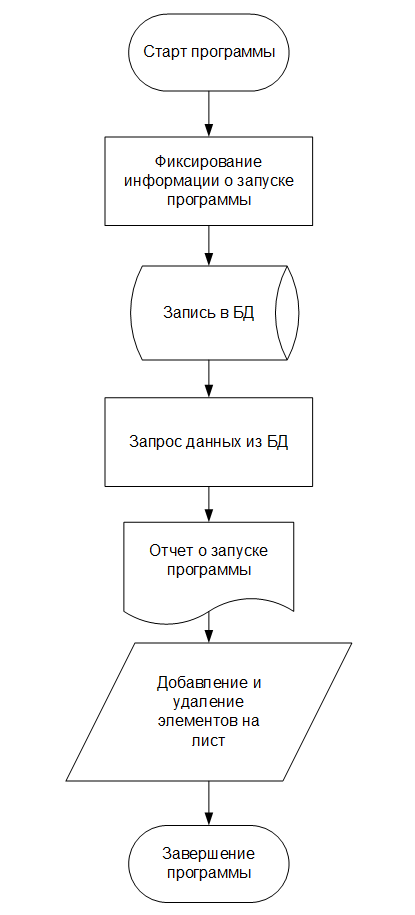


Рисунок 2.2 — Функциональная схема

Проектирование и реализация диаграммы последовательности.

Данная диаграмма, представлена в соответствии с рисунком 2.3. Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

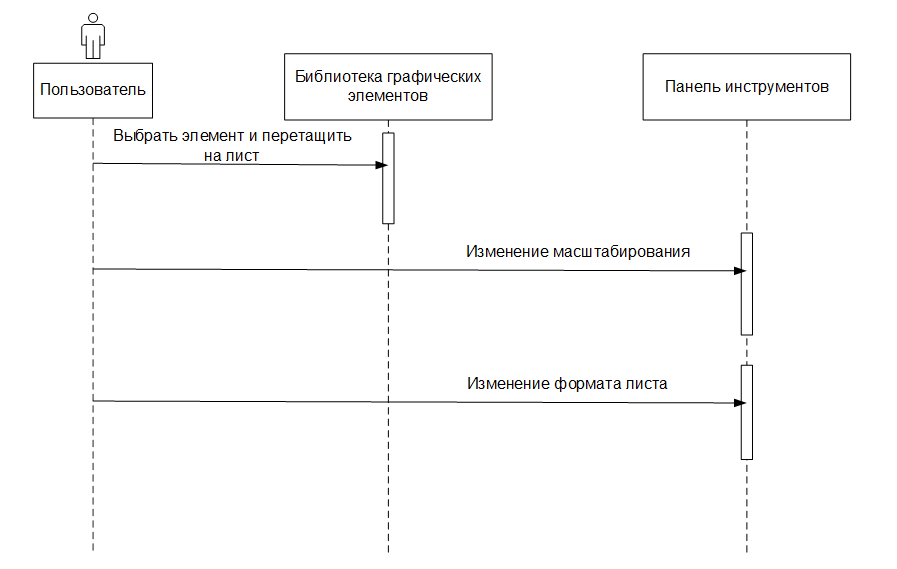


Рисунок 2.3 — Диаграмма последовательности

Проектирование и реализация диаграммы вариантов использования и диаграммы сценариев использования:

Данная диаграмма, представлена в соответствии с рисунком 2.4. Она представляет сценарий использования программы и отражает отношения между действующим лицом и прецедентами, является составной частью модели прецедентов, позволяет описать систему на логическом уровне.

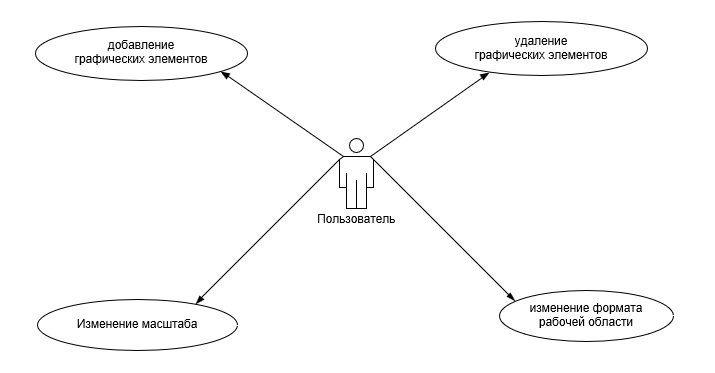


Рисунок 2.4 Диаграмма вариантов использования

1. **Программирование**

В данном разделе будут описаны функции программы по компонентам в которых они расположены.

Функции и модули компонента main .js:

1. connection() – данная функция создает подключение к базе данных mysql. Она принимает 4 параметра (host – адрес, user – имя пользователя, database – название базы данных, password – пароль от сервера);
2. createWindow – создает стартовое окно программы, принимает аргументом файл(для этого необходимо указать его путь), по которому произойдет render(процесс получения изображения по заготовленной модели);
3. connection.connect() – тестируем, удалось ли нам подключится к базе данных. Данная функция необходима для обеспечения корректного запуска программы. В качестве аргумента метод “connect” использованный из библиотеки “mysql” принимает функцию, которая в случае неудачного подключение вернет ошибку;
4. app.on('ready', main) – при готовности нашего приложения отрабатывает основная функция “main”;
5. app.on('window-all-closed') – если произошло закрытие окна логика данной функции закроет подключение к базе данных и закроет все окна приложения.

Компонент index.html:

Данный компонент возвращает верстку (структуру элементов на странице документа) нашего стартового окна, которое содержит функционал нашей автоматизированной системы проектирования плана города.

Компонент Editor.js:

Данный компонент отвечает за работу функций редактирования графических элементов, а также создание новых.

Компонент Format.js:

Данный компонент необходим для выбора формата листа.

Компонент Shapes.js:

Данный компонент необходим для создания различных объектов с использованием инструментов: линия, прямоугольник, полилиния.

1. **Отладка, оптимизация и тестирование программы**

Для отладки и тестирования использовался браузер и встроенные в него инструменты разработчика, а также локальная система управления базами данных (далее по тексту СУБД) MySQL.

Тестирование:

* необходимо запустить локальную СУБД;
* запустить выполнение основного скрипта проекта с помощью команды “npm start”;
* при закрытии приложения происходит отключения от базы данных;
* при нажатии в меню на пункт «Load statistics» в корне проекта появляется файл statistics.txt, который содержит статистику запуска приложения.

**Заключение**

В ходе выполнения курсового проекта было создано решение для автоматизированной системы проектирования плана города. Для решения были разработаны база данных, модуль графического интерфейса, модуль обработки данных, модуль для работы с базой данных, структурная схема, функциональная схема, диаграмма последовательности и диаграмма вариантов использования.

Разработанная система решает поставленные задачи, а именно:

* добавлять графические элементы на лист;
* удалять графические элементы с листа;
* менять масштаб плана;
* изменять размер листа;
* создавать собственные графические объекты;
* получать статистику запуска приложения.

# **Список использованных источников**

1. Николай Прохоренок “Учебник по Electron js. Разработка оконных приложений на JavaScript”.
2. Официальная документация по MySQL Workbench “<https://dev.mysql.com/doc/workbench/en/>”.
3. Дэн Седерхольм, Итан Маркотт “CSS ручной работы. Библиотека специалиста.”.
4. Б. Лоусон, Р. Шарп “Learn HTML 5”.
5. Документация для работы с библиотека MySql2 для JS “ https://metanit.com/web/nodejs/8.4.php ”.

ПРИЛОЖЕНИЕ А

Листинг программы

Главный файл “main.js”:

// Modules to control application life and create native browser window

const { app, BrowserWindow, Menu } = require('electron')

const path = require('path')

const mysql = require("mysql2");

const fs = require("fs");

const connection = mysql.createConnection({

host: "localhost",

user: "root",

database: "city-plan",

password: "root866705"

});

const createWindow = () => {

// Create the browser window.

const mainWindow = new BrowserWindow({

width: 1000,

height: 800,

webPreferences: {

nodeIntegration: true

}

})

// and load the index.html of the app.

mainWindow.loadFile('index.html')

// Open the DevTools.

// mainWindow.webContents.openDevTools()

const isMac = process.platform === 'darwin'

const template = [

// { role: 'appMenu' }

...(isMac ? [{

label: app.name,

submenu: [

{ role: 'about' },

{ type: 'separator' },

{ role: 'services' },

{ type: 'separator' },

{ role: 'hide' },

{ role: 'hideOthers' },

{ role: 'unhide' },

{ type: 'separator' },

{ role: 'quit' }

]

}] : []),

// { role: 'fileMenu' }

{

label: 'File',

submenu: [

{

label: 'Load statistics',

click: async () => {

const selectStatisticsSql = 'SELECT \* FROM `statistics`;';

connection.query(selectStatisticsSql, (err, results) => {

if (err) {

console.log(err);

} else {

// fs.writeFileSync("statistics.txt", results);

// let arr = [results]

// fs.writeFile('./statistics.json', results);

console.log(results);

let arrayResults =[];

for (let index = 0; index < results.length; index++) {

arrayResults.push(results[index].id, results[index].timestamp);

}

fs.writeFileSync('./statistics.txt', JSON.stringify(arrayResults));

}

});

}

},

isMac ? { role: 'close' } : { role: 'quit' }

]

},

// { role: 'editMenu' }

{

label: 'Edit',

submenu: [

{ role: 'undo' },

{ role: 'redo' },

{ type: 'separator' },

{ role: 'cut' },

{ role: 'copy' },

{ role: 'paste' },

...(isMac ? [

{ role: 'pasteAndMatchStyle' },

{ role: 'delete' },

{ role: 'selectAll' },

{ type: 'separator' },

{

label: 'Speech',

submenu: [

{ role: 'startSpeaking' },

{ role: 'stopSpeaking' }

]

}

] : [

{ role: 'delete' },

{ type: 'separator' },

{ role: 'selectAll' }

])

]

},

// { role: 'viewMenu' }

{

label: 'View',

submenu: [

{ role: 'reload' },

{ role: 'forceReload' },

{ role: 'toggleDevTools' },

{ type: 'separator' },

{ role: 'resetZoom' },

{ role: 'zoomIn' },

{ role: 'zoomOut' },

{ type: 'separator' },

{ role: 'togglefullscreen' }

]

},

// { role: 'windowMenu' }

{

label: 'Window',

submenu: [

{ role: 'minimize' },

{ role: 'zoom' },

...(isMac ? [

{ type: 'separator' },

{ role: 'front' },

{ type: 'separator' },

{ role: 'window' }

] : [

{ role: 'close' }

])

]

},

{

role: 'help',

submenu: [

{

label: 'Learn More',

click: async () => {

const { shell } = require('electron')

await shell.openExternal('https://electronjs.org')

}

}

]

}

]

const menu = Menu.buildFromTemplate(template)

Menu.setApplicationMenu(menu)

}

// This method will be called when Electron has finished

// initialization and is ready to create browser windows.

// Некоторые интерфейсы API могут использоваться только после возникновения этого события.

app.whenReady().then(() => {

createWindow()

let now = new Date();

let options = {

era: 'long',

year: 'numeric',

month: 'long',

day: 'numeric',

weekday: 'long',

timezone: 'UTC',

hour: 'numeric',

minute: 'numeric',

second: 'numeric'

};

let timestamp = now.toLocaleString("ru", options);

const insertSql = 'INSERT INTO `statistics`(timestamp) VALUES(?)';

connection.query(insertSql, timestamp, function (err, results) {

if (err) console.log(err);

else console.log("Add data");

});

app.on('activate', () => {

// On macOS it's common to re-create a window in the app when the

// dock icon is clicked and there are no other windows open.

if (BrowserWindow.getAllWindows().length === 0) createWindow()

})

})

// Quit when all windows are closed, except on macOS. There, it's common

// for applications and their menu bar to stay active until the user quits

// explicitly with Cmd + Q.

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') app.quit()

})

Файл index.js:

const { ipcRenderer } = require('electron');

document.getElementById('addMobile').addEventListener('click', () => {

ipcRenderer.send('add-mobile-window');

});

document.getElementById('addReceipts').addEventListener('click', () => {

ipcRenderer.send('add-mobile-receipts-window');

});

document.getElementById('addSales').addEventListener('click', () => {

ipcRenderer.send('add-sales-window');

});

document.getElementById('createReportMobile').addEventListener('click', () => {

ipcRenderer.send('create-Mobile-Report-file');

});

document.getElementById('createReportReceipts').addEventListener('click', () => {

ipcRenderer.send('create-Receipts-Report-file');

});

document.getElementById('createReportSales').addEventListener('click', () => {

ipcRenderer.send('create-Sales-Report-file');

});

Файл index.html:

<!--[if IE]><meta http-equiv="X-UA-Compatible" content="IE=5,IE=9" ><![endif]-->

<!DOCTYPE html>

<html>

<head>

<title>Grapheditor</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">

<link rel="stylesheet" type="text/css" href="styles/grapheditor.css">

<script type="text/javascript">

// Parses URL parameters. Supported parameters are:

// - lang=xy: Specifies the language of the user interface.

// - touch=1: Enables a touch-style user interface.

// - storage=local: Enables HTML5 local storage.

// - chrome=0: Chromeless mode.

var urlParams = (function(url)

{

var result = new Object();

var idx = url.lastIndexOf('?');

if (idx > 0)

{

var params = url.substring(idx + 1).split('&');

for (var i = 0; i < params.length; i++)

{

idx = params[i].indexOf('=');

if (idx > 0)

{

result[params[i].substring(0, idx)] = params[i].substring(idx + 1);

}

}

}

return result;

})(window.location.href);

// Default resources are included in grapheditor resources

mxLoadResources = false;

</script>

<script type="text/javascript" src="js/Init.js"></script>

<script type="text/javascript" src="deflate/pako.min.js"></script>

<script type="text/javascript" src="deflate/base64.js"></script>

<script type="text/javascript" src="jscolor/jscolor.js"></script>

<script type="text/javascript" src="sanitizer/sanitizer.min.js"></script>

<script type="text/javascript" src="src/js/mxClient.js"></script>

<script type="text/javascript" src="js/EditorUi.js"></script>

<script type="text/javascript" src="js/Editor.js"></script>

<script type="text/javascript" src="js/Sidebar.js"></script>

<script type="text/javascript" src="js/Graph.js"></script>

<script type="text/javascript" src="js/Format.js"></script>

<script type="text/javascript" src="js/Shapes.js"></script>

<script type="text/javascript" src="js/Actions.js"></script>

<script type="text/javascript" src="js/Menus.js"></script>

<script type="text/javascript" src="js/Toolbar.js"></script>

<script type="text/javascript" src="js/Dialogs.js"></script>

</head>

<body class="geEditor">

<script type="text/javascript">

// Extends EditorUi to update I/O action states based on availability of backend

(function()

{

var editorUiInit = EditorUi.prototype.init;

EditorUi.prototype.init = function()

{

editorUiInit.apply(this, arguments);

this.actions.get('export').setEnabled(false);

// Updates action states which require a backend

if (!Editor.useLocalStorage)

{

mxUtils.post(OPEN\_URL, '', mxUtils.bind(this, function(req)

{

var enabled = req.getStatus() != 404;

this.actions.get('open').setEnabled(enabled || Graph.fileSupport);

this.actions.get('import').setEnabled(enabled || Graph.fileSupport);

this.actions.get('index.html/save').setEnabled(enabled);

this.actions.get('saveAs').setEnabled(enabled);

this.actions.get('export').setEnabled(enabled);

}));

}

};

// Adds required resources (disables loading of fallback properties, this can only

// be used if we know that all keys are defined in the language specific file)

mxResources.loadDefaultBundle = false;

var bundle = mxResources.getDefaultBundle(RESOURCE\_BASE, mxLanguage) ||

mxResources.getSpecialBundle(RESOURCE\_BASE, mxLanguage);

// Fixes possible asynchronous requests

mxUtils.getAll([bundle, STYLE\_PATH + '/default.xml'], function(xhr)

{

// Adds bundle text to resources

mxResources.parse(xhr[0].getText());

// Configures the default graph theme

var themes = new Object();

themes[Graph.prototype.defaultThemeName] = xhr[1].getDocumentElement();

// Main

new EditorUi(new Editor(urlParams['chrome'] == '0', themes));

}, function()

{

document.body.innerHTML = '<center style="margin-top:10%;">Error loading resource files. Please check browser console.</center>';

});

})();

</script>

</body>

</html>

ПРИЛОЖЕНИЕ Б

Результаты выполнения программы

При запуске проекта пользователя встречает начальное окно, изображенное в соответствии с рисунком Б.1

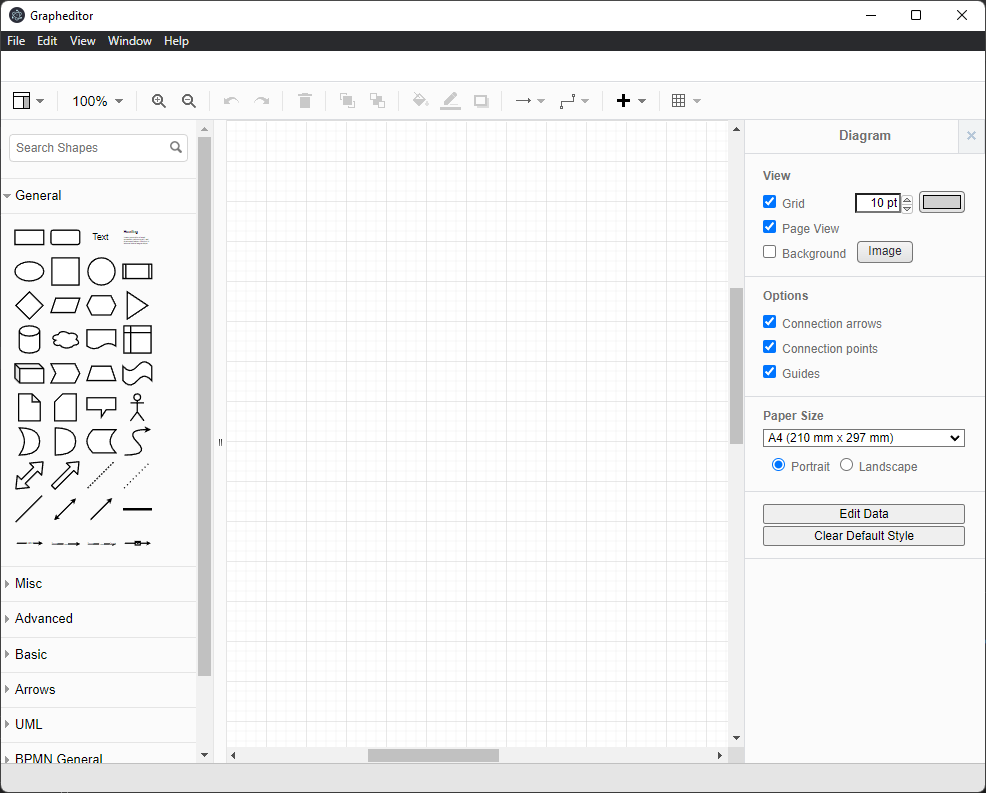


Рисунок Б.1 – экранная форма начального окна приложения

Для добавления элемента на рабочую область необходимо навести мышку и зажать ЛКМ и перетащить элемент. Элемент в рабочей области представлен в соответствии с рисунком Б.2.

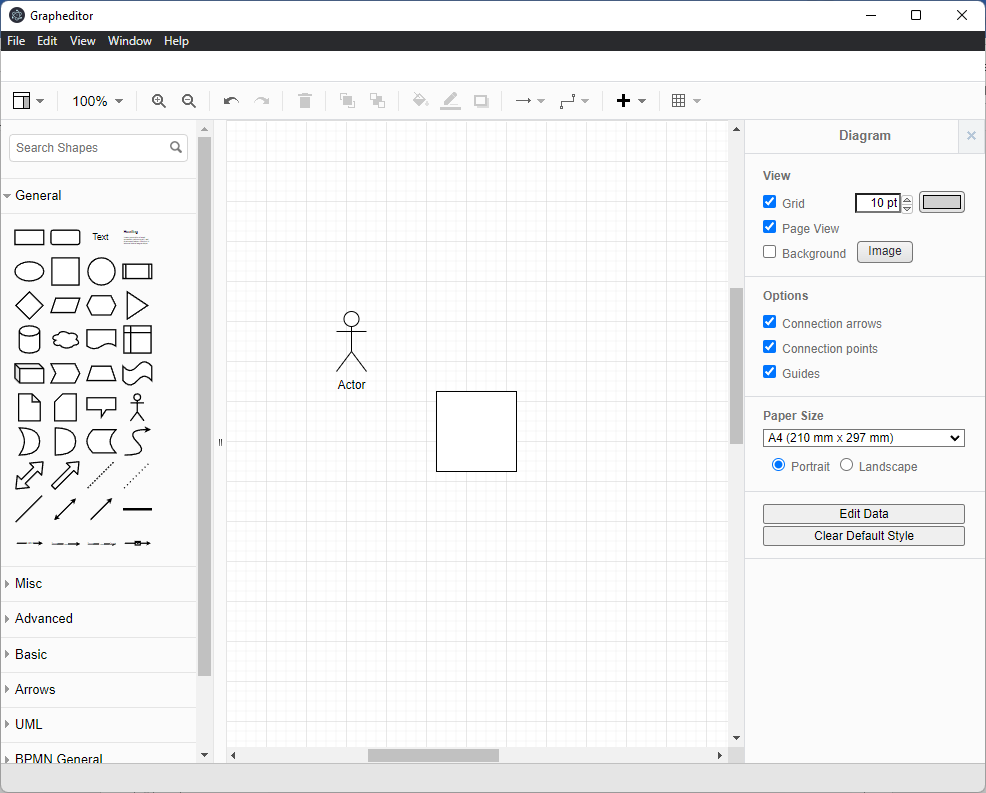


Рисунок Б.2 – элементы в рабочей области

Для формирования отчета о запусках приложения, необходимо в меню «File» выбрать пункт «Load statistics». Для ознакомления с отчетом перейти в корневой каталог проекта, где уже будет сформирован отчет, файл с названием “ statistics.txt”. Открыть его двойным нажатием по нему ЛКМ. Пример готового отчета изображен на рисунке Б.3.

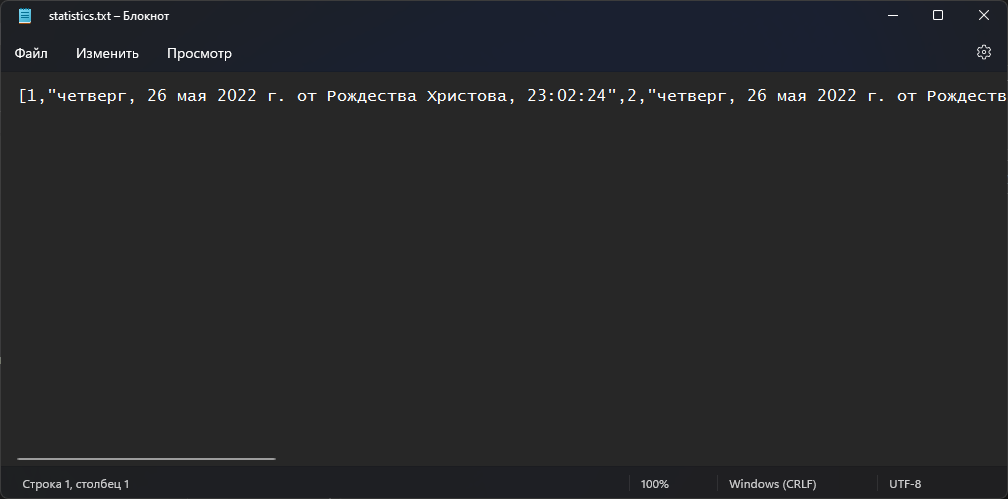


Рисунок Б.3 – иллюстрация примера отчета о запусках приложения

Закрытие приложение производится нажатием на крестик расположенном на начальном окне в правом верхнем углу, на этом этапе происходит отключение от базы данных.