**Содержание**

[Введение 3](#_Toc119578054)

[1. Анализ объекта 4](#_Toc119578055)

[1.1 Описание предметной области. 4](#_Toc119578056)

[1.2 Построение концептуальной модели предметной области. 6](#_Toc119578057)

[2 Постановка задачи 8](#_Toc119578058)

[2.1 Определение требований к программной системе. 8](#_Toc119578059)

[2.2 Описание аналогов системы. 8](#_Toc119578060)

[2.3 Обзор и обоснование выбора средств реализации 11](#_Toc119578061)

[3 Проектирование 11](#_Toc119578062)

[3.1 Разработка архитектуры программного продукта 12](#_Toc119578063)

[3.2 Проектирование структур хранения данных. 19](#_Toc119578064)

[4 Реализация 20](#_Toc119578065)

[4.1 Разработка архитектуры программного продукта 20](#_Toc119578066)

[4.2 Разработка интерфейса программного продукта 28](#_Toc119578067)

[5. Тестирование 34](#_Toc119578068)

[Литература 39](#_Toc119578069)

[Приложение А 41](#_Toc119578070)

# Введение

Рисование по координатам точек на координатной плоскости (сетке) это не только увлекательное занятие, но и поучительное как в области рисования, так и в математике. Текущее приложение предназначено для использования, как на уроках математики, так и для организации интересного досуга дома.

Существует и другой способ создать рисунок (фигуру) по координатам не в программе "Рисуем по координатам", а на клетчатом поле бумаге. Данная тема интересна тем, что в координатной плоскости на обычной клетчатой бумаге (клетчатом поле) можно строить не только различные графики функций, но и увидеть связь природной красоты и строгой математики, создавая красивые рисунки и фигуры по координатам своими руками.

Данный материал можно использовать как в урочной, так и внеурочной деятельности.

Данный материал можно использовать по разному. Можно распечатать карточки с заданиями, а ответы не выдавать. Желательно не указывать название рисунка. А можно сформулировать обратное задание: по данным рисунка запишите последовательно координаты точек. А можно использовать этот дидактический материал и по другому: сформулировать обратное задание. Для того, чтобы облегчить проверку такого задания, можно указать с какой точки начать записывать координаты и в каком направлении двигаться от этой точки.

Мой проект представляет собой веб-приложение, которое имеет возможность построения изображения по заданным координатам вершин. Данная курсовая работа выполнена на языке программирования высокого уровня Java с использованием IDE IntelijIDEA.

# Анализ объекта

## **Описание предметной области.**

Рисунки и фигуры на координатной плоскости строятся по точкам. Каждая точка на плоскости имеет две координаты и записывается в виде двух чисел X (икс) и Y (игрек) через точку с запятой в скобках, например точка A (х; у), где первая цифра обозначает расстояние, отложенное от центра системы координат по оси X (икс), вторая цифра расстояние, отложенное от центра по оси Y (игрек). Рассмотрим построение координатных точек на координатной плоскости (сетке) на клетчатой бумаге реального примера.

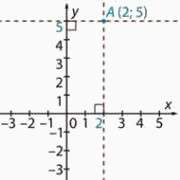


Рисунок 1.1 – Пример точек на плоскости

Для этого нужно взять либо обычный лист из тетради в клеточку, либо распечатать готовый лист А4 в клетку. Создаём прямоугольную систему координат. Для этого, рисуем координатные оси X и Y, где принимаем за единичный отрезок 1 (одна) клетка и для удобства нумеруем систему координат цифрами, как показано на рисунке. Вот теперь все готово и можно приступать к построениям.

Возьмем для примера первую координату точку A (2;5) нашего рисунка или картинки и отложим эти расстояния по координатным осям X и Y на нашем клетчатом поле. Первое число 2 (два) мы отложим по оси X, а второе значение нашей координаты число 5 (пять) по координатной оси Y. В пересечении двух мысленно проведенных перпендикулярных линий к числовым осям координат (они обозначены пунктирной линией на рисунке), мы получим нашу первую координату точку A (2;5).

Таким же методом строим координаты второй точки, третьей и так далее. После построения всех точек мы соединяем прямой линией первую точку со второй, вторую с последующей точкой в порядке их следования. После соединения всех точек мы получим заданный рисунок (фигуру) по своим координатам.

Чтобы найти координаты любой точки рисунка (фигуры) на плоскости, необходимо создать координатную числовую плоскость на этом рисунке и опустить из этой точки перпендикуляры на координатные числовые оси X и Y. Эти два значения и будут ее координаты.

Примеры таких изображений отображены ниже

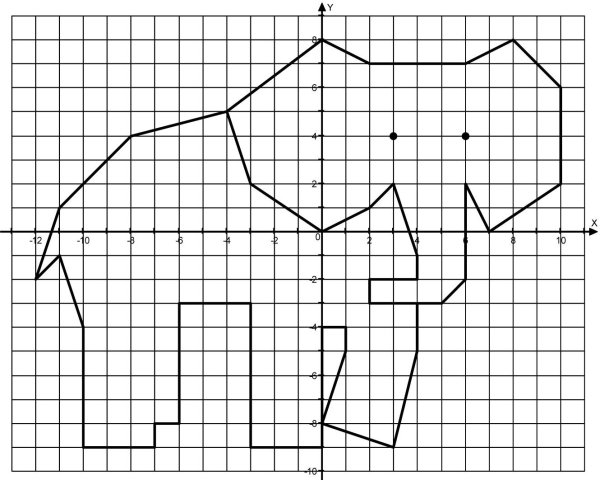


Рисунок 1.2 – Пример изображения слона на плоскости

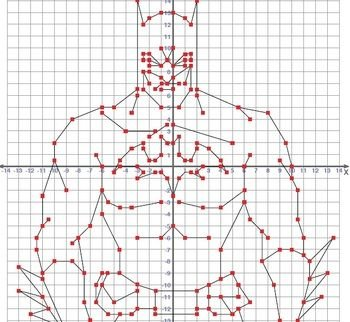


Рисунок 1.3 – Пример изображения бэтмэна на плоскости

## **Построение концептуальной модели предметной области.**

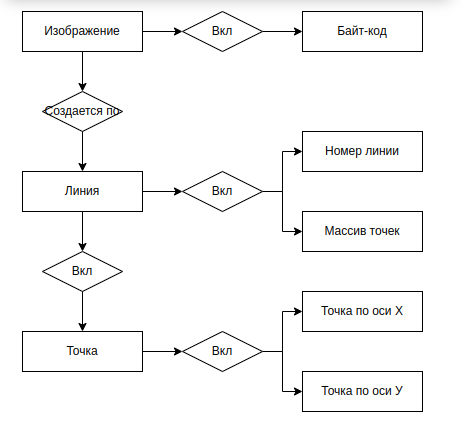


Рисунок 1.4 – Модель предметной области

Концептуальная модель приложения построения изображения по заданным точкам состоит из нескольких частей. Разбирать их легче с конца. На выходе мы получаем изображение на котором нарисовано множество линийю Каждая отдельная линия представляет из себя массив точекю Каждая же отделдьная точка имеет у себя два значения — значение по оси координат Х и значение по оси координат У.

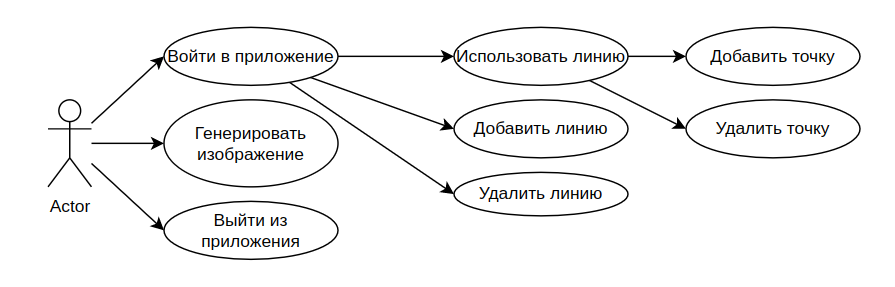
# 

# Постановка задачи

## **Определение требований к программной системе.**

Программная система должна удовлетворять нижеперечисленным требованиям и решать поставленные задачию Приложение должно:

1. Возможность работать на разных операционных системах
2. Не требовать регистрации польхователя
3. Иметь возможность рисовать более чем одну линию за раз
4. Поддерживать ввод данных с помощью клавиатуры и мыши
5. Иметь простой и дружелюбный интерфейс
6. Использовать язык программирвоания Java версии 11

Рисунок 2.1 - Диаграмма вариантов использования

## **Описание аналогов системы.**

Поискав в интернете приложения по заданной теме мной было найдено только два решения. Первое решение представляет собой попытаться купить программу на сайте зачетка.

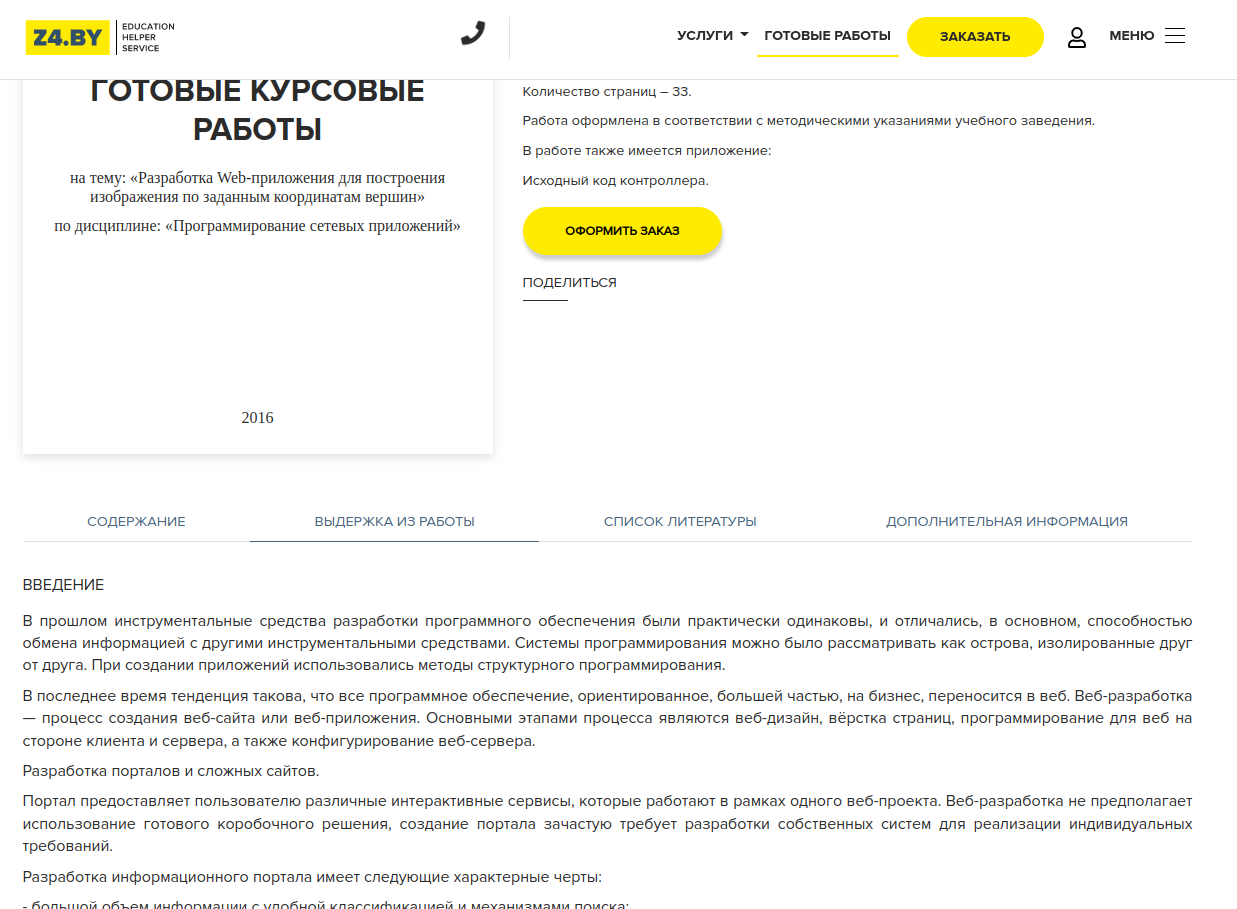


Рисунок 2.2 - Зачетка

Однако это плохое решение по нескольким причинам. Во первых, это не поможет мне развивать свои способности к программированию. Также название их работы на сайте не соответствует действиетльности, и описывает создание информационного портала.

Второе найденное решение было на сайте Веб-математики.

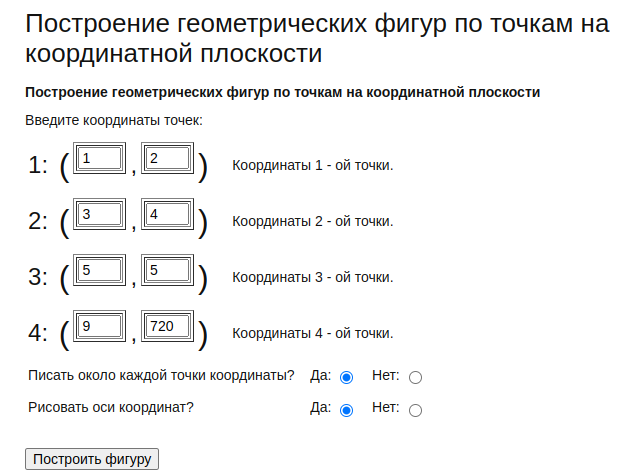


Рисунок 2.3 – Страница генерации изображения сайта веб-математики

Сайт позволяет сгенерировать изоюражения по точкам. Однако у этого решения есть два больших минуса. Во первых — оно позволяет генерацию изображения только по одной линии, так что если мне понадобистя что-то нарисовать с двумя линиями — оно мне уже не подойдет. Второй большой минус — генерация изображения ограничена, по оси У невозможно получить изображение с координатой больше 9, что также не дает возможности для генерации сложных изображений.

К тому же сайт не позволяет добавлять/удалять точки в режиме реального времени и требует обязательного заполнения их всех. Так что в том случае если вдруг ошибся с количеством точек и указал лишнюю, то придется отменять все изменения и заново вводить все координаты вершин.

Мой курсовой проект представляет собой веб приложение, которое можно легко развернуть в системе и затем использовать всеми пользователями сети без обязательного доступа к интернету. Также оно позволяет пользователю генерировать изображение по более чем одной линии и поддерживает автомасштабирование изображения в зависимости от «размера» координат пользователя.

### **2.3 Обзор и обоснование выбора средств реализации**

При разработке программного средства будут использованы следующие технологии: язык программирования Java 11, JS, CSS. В качестве фреймворка будет выбран фреймворк Spring Framework.

Для реализации Back-end части будут использована технология Spring Framework. Для реализации Front-end части будут использованы следующие технологии: Thymeleaf, HTML, CSS и JavaScript. Каждая из выбранных технологий отвечает за разные аспекты работы программы.

Существует огромное количество инструментария для программирования как на стороне сервера, так и на стороне клиента, как платных, так и бесплатных. Основные программные средства, которые будут использованы при разработке программного средства:

интегрированная среда разработки IDEA для разработки на языках программирования Java и JavaScript, языке разметки HTML и каскадных таблиц стилей CSS с возможностями анализа кода на лету, предотвращения ошибок в коде и автоматизированными средствами рефакторинга для Java;

браузеры Opera и Google Chrome для проверки на кроссбраузерность программного средства, а также проверки результатов работы программного средства при разработке.

Все перечисленные выше серверные и клиентские технологии, а также соответствующий инструментарий идеально подходят для разработки веб-приложений.

# Проектирование

## **Разработка архитектуры программного продукта**

Чаще всего веб-приложения состоят как минимум из трёх основных компонентов:

Клиентская часть веб приложения — это графический интерфейс. Это то, что вы видите на странице. Графический интерфейс отображается в браузере. Пользователь взаимодействует с веб-приложением именно через браузер, кликая по ссылкам и кнопкам.

Серверная часть веб-приложения — это программа или скрипт на сервере, обрабатывающая запросы пользователя (точнее, запросы браузера). Чаще всего серверная часть веб-приложения программируется на PHP. При каждом переходе пользователя по ссылке браузер отправляет запрос к серверу. Сервер обрабатывает этот запрос, вызывая некоторый PHP-скрипт, который формирует веб-страничку, описанную языком HTML, и отсылает клиенту по сети. Браузер тут же отображает полученный результат в виде очередной веб-страницы.

База данных (БД, или система управления баазми данных, СУБД) - программное обеспечение на сервере, занимающееся хранением данных и их выдачей в нужный момент. В случае форума или блога, хранимые в БД данные — это посты, комментарии, новости, и так далее. База данных располагается на сервере. Серверная часть веб-приложения (то есть, PHP скрипт) обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

Наше приложение также состоять из этидву частей (поскольку мы не используем базу данных) и основываться будет на Spring MVC. MVC — это не шаблон проекта, это конструкционный шаблон, который описывает способ построения структуры нашего приложения, сферы ответственности и взаимодействие каждой из частей в данной структуре.

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов [14]. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

* Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:

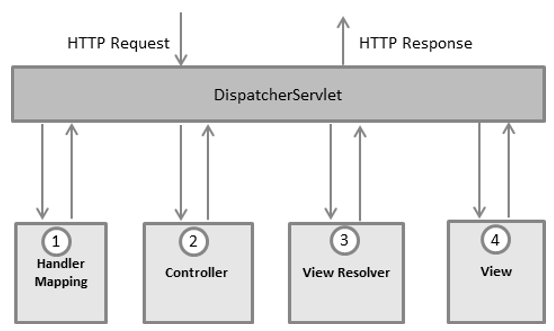


Рисунок 3.1 – Рабочий процесс обработки запроса

Ниже приведена последовательность событий, соответствующая входящему HTTP-запросу:

* После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
* Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике, и возвращает в DispatcherServlet имя Вида (View).
* При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
* После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений.

DispatcherServlet отправляет запрос контроллерам для выполнения определённых функций. Аннотация @Controllerannotation указывает, что конкретный класс является контроллером. Аннотация @RequestMapping используется для мапинга (связывания) с URL для всего класса или для конкретного метода обработчика.

Аннотация Controller определяет класс как Контроллер Spring MVC. В первом случае, @RequestMapping указывает, что все методы в данном Контроллере относятся к URL-адресу "/hello".

Следующая аннотация @RequestMapping(method = RequestMethod.GET) используется для объявления метода printHello() как дефолтного метода для обработки HTTP-запросов GET. Вы можете определить любой другой метод как обработчик всех POST-запросов по данному URL-адресу.

Вы можете написать вышеуказанный Контроллер по-другому, указав дополнительные атрибуты для аннотации @RequestMapping следующим образом

@RequestMapping(value = "/hello", method = RequestMethod.GET)  
Атрибут «value» указывает URL, с которым мы связываем данный метод (value = "/hello"), далее указывается, что этот метод будет обрабатывать GET-запросы (method = RequestMethod.GET). Также, нужно отметить важные моменты в отношении приведённого выше контроллера:

* Вы определяете бизнес-логику внутри связанного таким образом служебного метода. Из него Вы можете вызывать любые другие методы.
* Основываясь на заданной бизнес-логике, в рамках этого метода Вы создаёте Модель (Model). Вы можете добавлять аттрибуты Модели, которые будут добавлены в Вид (View). В примере выше мы создаём Модель с атрибутом «message».
* Данный служебный метод возвращает имя Вида в виде строки String. В данном случае, запрашиваемый Вид имеет имя «hello».

Spring MVC поддерживает множество типов Видов для различных технологий отображения страницы. В том числе — JSP, HTML, PDF, Excel, XML, Velocity templates, XSLT, JSON, каналы Atom и RSS, JasperReports и проч. Но чаще всего используются шаблоны JSP, написанные при помощи JSTL или HTML файлы, используя Thymeleaf.

На рисунке 3.2 представлена диаграмма последовательности, которая представляет собой один конкретный экземпляр работы программы под управлением пользователя.

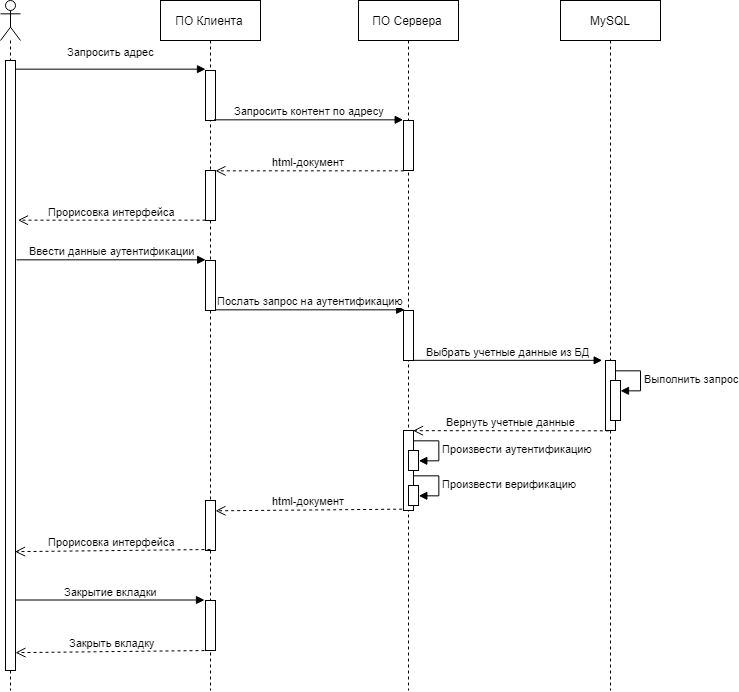


Рисунок 3.2 – Диаграмма последовательности

Диаграмма последовательности – это диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание – деятельность – уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).

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.

На диаграмме последовательности изображен один актёр – пользователь системы, а также следующие объекты:

– ПО Клиента;

– ПО Сервера;

Представлена последовательность действия в три этапа:

1. Пользователь вводит адрес сайта в браузере, а сервер отправляет клиенту html-документ.

2. Клиент вводит данные аутентификации. Браузер посылает запрос серверу на аутентификацию, а сервер посылает запрос на сервер MySQL, который выбирает данные и посылает их серверу. Сервер их преобразует и возвращает клиентскому браузеру. Браузер отображает информацию для клиента.

3. Клиент нажимает кнопку закрытия в браузере – вкладка закрывается. Работа системы завершена.

В этом подразделе произведено обоснование выбора технологий, а также описаны три основных типа диаграммы UML для отображения структуры данных.

## **3.2 Проектирование структур хранения данных.**

В нашем приложении мы не используем базу данных, поэтому этот пункт для на не актуален.

# 

# 4. Реализация

## **Разработка архитектуры программного продукта**

Так как мы используем шаблон MVC, идет отдельно разработка классов модели (сущности), отдельно вида (страниц для браузера) и контроллеров (обработки действий пользователя).

Так как данные в шаблоне будут изменяться по полученным из бд, нам нужно создать класс-сущность для хранения этих данных.

Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна. Поля сущности совпадают с полями таблицы, в которой эти данные хранятся в базе данных.

Пример класса сущности отображен в листинге ниже.

Листинг 1. Сущности линии и точки

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Line implements Comparable<Line> {

private Integer lineNumber;

private List<Point> points = new ArrayList<>();

@Override

public int compareTo(Line o) {

return lineNumber.compareTo(o.lineNumber);

}

}

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Point {

@NonNull

private Integer pointNumber;

@NonNull

private Integer pointY;

@NonNull

private Integer pointX;

}

После написания сущностей идет описание сервисов – классов, которые взаимодействуют с сущностью. В нашем случае это будет единственный сервис, который получает от пользователя данные для генерации изображения, генерирует его и возвращает ответ.

Листинг 2. Реализация сервиса изображений

@Service

public class ImageService {

private static final int MULTIPLIER = 10;

private static final int PADDING = 5;

private static final int MIN\_SIZE = 100;

private static final int MAX\_SIZE = 10000;

int size;

public byte[] generateImage(List<Line> lines) throws IOException {

defineIMageSize(lines);

int width = size \* 2 + PADDING;

int height = size \* 2 + PADDING;

BufferedImage bufferedImage = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics2D g2d = bufferedImage.createGraphics();

g2d.setColor(Color.white);

g2d.fillRect(0, 0, width, height);

g2d.setColor(Color.black);

for (Line line : lines) {

drawLine(g2d, line);

}

g2d.dispose();

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(bufferedImage, "png", baos);

return baos.toByteArray();

}

private void defineIMageSize(List<Line> lines) {

int minX = Integer.MAX\_VALUE;

int minY = Integer.MAX\_VALUE;

int maxX = Integer.MIN\_VALUE;

int maxY = Integer.MIN\_VALUE;

for (Line line : lines) {

for (Point point : line.getPoints()) {

int x = point.getPointX();

int y = point.getPointY();

if (minX > x) minX = x;

if (minY > y) minY = y;

if (maxX < x) maxX = x;

if (maxY < y) maxY = y;

}

}

int maxWidth = Integer.max(Math.abs(minX), maxX) \* MULTIPLIER;

int maxHeight = Integer.max(Math.abs(minY), maxY) \* MULTIPLIER;

size = Integer.max(maxWidth, maxHeight);

if (size < MIN\_SIZE) size = MIN\_SIZE;

if (size > MAX\_SIZE) size = MAX\_SIZE;

}

private void drawLine(Graphics2D g2d, Line line) {

List<Point> points = line.getPoints();

for (int i = 0; i < points.size() - 1; i++) {

Point a = points.get(i);

Point b = points.get(i + 1);

int x1 = a.getPointX();

int y1 = a.getPointY();

int x2 = b.getPointX();

int y2 = b.getPointY();

drawLine(g2d, x1, y1, x2, y2);

}

}

private void drawLine(Graphics2D g2d, int x1, int y1, int x2, int y2) {

g2d.drawLine(size + x1 \* MULTIPLIER,

size - y1 \* MULTIPLIER,

size + x2 \* MULTIPLIER,

size - y2 \* MULTIPLIER);

}

}

После реализации сервисов, остается только 2 шага до окончания разработки приложения – это написание контроллеров и представлений. Контроллер – это специализированный класс, который отвечает на запросы пользователей по определенным ссылкам. Обычно они именуются так: СтраницаController, где страница – это имя страница, за которую отвечает контроллер. В контроллерах опишем ответы на запросы пользователя по взаимодействию с базой данных, т.е. ответы на запросы типа /read, /create/ edit. Ниже показан контроллер главной страницы.

Листинг 3. Контроллер входа и регистрации

@Controller

@RequestMapping("/")

@RequiredArgsConstructor

public class IndexController {

private final ImageService imageService;

private final ApplicationContext context;

public static int counter = 0;

private static Map<Integer, byte[]> map = new HashMap<>();

@GetMapping

public String getIndex(Model model) {

Point pointA = new Point(1, 5, 6);

Point pointB = new Point(2, 7, 8);

Line line = new Line(1, List.of(pointA, pointB));

model.addAttribute("lines", List.of(line));

return "index.html";

}

@PostMapping("/image")

public ResponseEntity<String> makeImage(@RequestBody Lines lines) throws IOException {

counter++;

map.put(counter, imageService.generateImage(lines.getLines()));

return ResponseEntity.ok("/image/" + counter);

}

@GetMapping(path = "/image/{id}", produces = MediaType.IMAGE\_PNG\_VALUE)

@ResponseBody

public byte[] getResult(@PathVariable Integer id) throws IOException {

if (map.containsKey(id)) {

byte[] bytes = map.get(id);

map.remove(id);

return bytes;

}

InputStream in = context.getResource("classpath:/static/404.png").getInputStream();

return IOUtils.toByteArray(in);

}

}

Остался последний шаг – связывание шаблонов и контроллеров. Для этого воспользуемся шаблонизатором Thymeleaf, который работает в связке с Spring.

Суть его проста – те данные, что мы передавали в модель отображения, таймлиф можем выводить простой командой th:text=”${}”. В скобках пишется имя переменной. Обращение идет как к переменным класса. В нужных местах вставляем этот код.

Листинг 4. Форма задания координат

<section>

<form action="/" method="POST">

<input type="button" value="Submit" onclick="sendData()">

<input type="submit" value="Submit" hidden class="submit">

<input type="button" value="Add line" onclick="addLine()">

<div class="lines">

<div class="line" th:each="line:${lines}">

<div class="lineTitle">Line</div>

<input type="button" value="Add point" onclick="addPoint(this)">

<div class="points">

<div class="point" name="point" th:each="point:${line.points}">

<div>

Point X: <input type="number" class="pointX" th:value="${point.pointX}" required/>

</div>

<div>

Point Y: <input type="number" class="pointY" th:value="${point.pointY}" required/>

</div>

</div>

</div>

</div>

</div>

</form>

</section>

В итоге мы получаем следующие файлы для работы нашей программы:

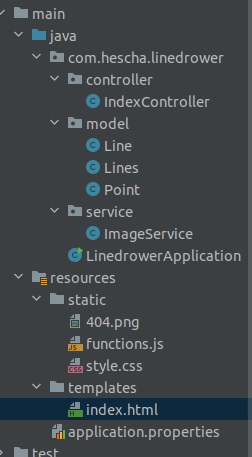
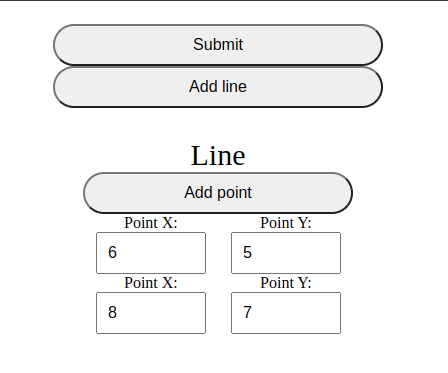


Рисунок 4.1 – Созданные файлы для работы приложения

## **Разработка интерфейса программного продукта**

Стартовое окно выглядит следующим образом:

Рисунок 4.2 – Главная страница

После ввода всех линий и координат точек получаем изображение, к примеру алмаз.

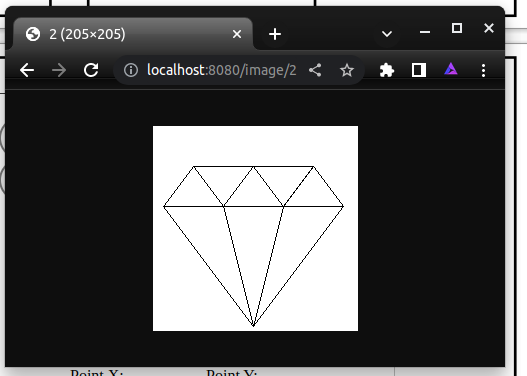


Рисунок 4.3 – Сгенерированное изображение

**Разработка алгоритмов реализации вариантов использования.**

Во время разработки приложения было решено множество задач. Решение некоторых из них представим ниже.

1. Добавление новых линий

Нужно было добавить возможность генерации изображения по более чем одной линии, чтб была возможность добавлять и удалять новые линии по мере необходимостию Поэтому пришлось написать функцию, которая по нажатию кнопки добавляла новую линию к уже существующим.

function addLine() {

var newMessage = `

<div class="line">

<div class="lineTitle">Line</div>

<input type="button" onclick="deletePoint(this)" value="Delete line" />

<input type="button" value="Add point" onclick="addPoint(this)">

<div class="points">

<div class="point">

<div>Point X: <input type="number" class="pointX" required />

</div>

<div>Point Y: <input type="number" class="pointY" required />

</div>

</div>

<div class="point" name="point">

<div>Point X: <input type="number" class="pointX" required />

</div>

<div>Point Y: <input type="number" class="pointY" required />

</div>

</div>

</div>

</div> `;

var place = document.querySelector(".lines");

place.insertAdjacentHTML("beforeend", newMessage);

};

1. Отправка данных на сервер в верном формате

Так как сервер принимает данные в формате json, обычная отправка формы вызвала бы ошибку сервера и не работала бы должным образом. Пришлось написать функцию, которая пробегается по всем линиям и координатам, выставляет им верные номера последовательности, преобразует данные к формату json и отправляет все на сервер.

function sendData () {

var $myForm = document.querySelector('form');

if (!$myForm.checkValidity()) {

$myForm.querySelector('.submit').click();

return;

}

var linesDOM = document.querySelectorAll(".line");

if(!linesDOM)return;

var lines=[];

for(let i=0; i<linesDOM.length; i++){

var points = [];

const lineDOM=linesDOM[i];

var pointsDOM = lineDOM.querySelectorAll(".point");

for(let j=0; j<pointsDOM.length; j++){

const pointDOM = pointsDOM[j];

const pointX = pointDOM.querySelector(".pointX").value;

const pointY = pointDOM.querySelector(".pointY").value;

var point = {

'pointNumber': j,

'pointX':pointX,

'pointY':pointY

};

points.push(point);

}

const line={

'lineNumber': i,

'points':points

};

lines.push(line);

}

var data = {"lines": lines};

$.ajax({

type: "POST",

contentType: "application/json",

url: "/image",

data: JSON.stringify(data),

dataType: 'json',

cache: false,

timeout: 600000,

success: function (data) {

alert("success!");

},

error: function (e) {

if(e["status"]==200){

window.location = e["responseText"];

} else{

alert(JSON.stringify(e));

}

}

});

}

# 5. Тестирование

Проведем тестирование программы, пройдя все варианты взаимодействия. Запустим само приложение. Запуск происходит без проблем.

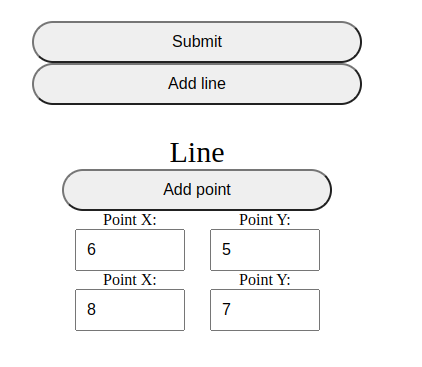


Рисунок 5.1 – Запуск приложения

Добавим несколько новых точек. Как видно, они добавляются без проблем.

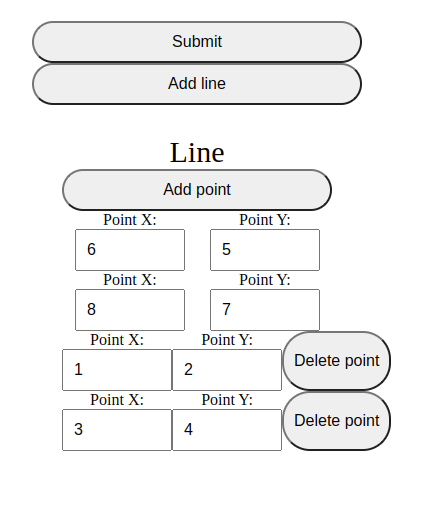


Рисунок 5.2 – Добавление точек

Попробуем удалить среднюю точку. Как видим на изображении, именно выбранная точка удалилась, остальные точки это не затронуло.

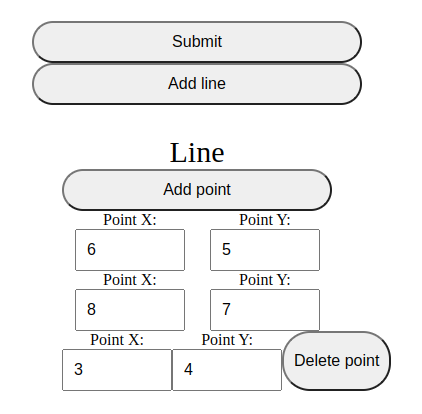


Рисунок 5.3 – Удаление точки

Попробуем добавить две линии. Они добавилась.

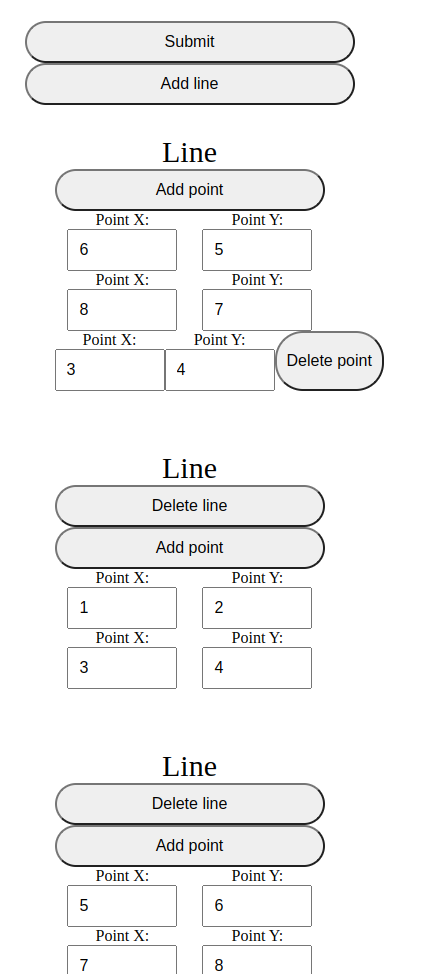


Рисунок 5.4 – Добавление линии

Теперь удалим среднюю линию. Так же как и с точками — удалилась только нужная линия

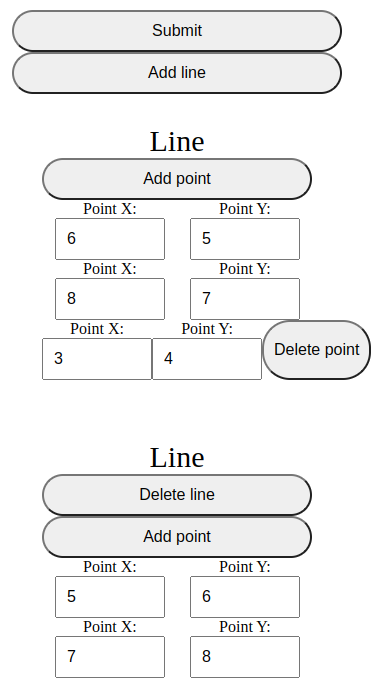


Рисунок 5.5 – Удаление линии

Попробуем отправить данные на сервер не зполняя поля — получаем ошибку.

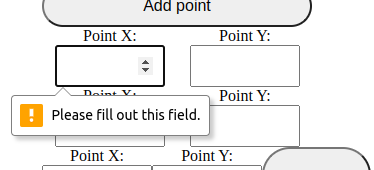


Рисунок 5.6 –Проверка заполнения полей

Попробуем сгенерировать сложное изображение используя шаюлон точек.

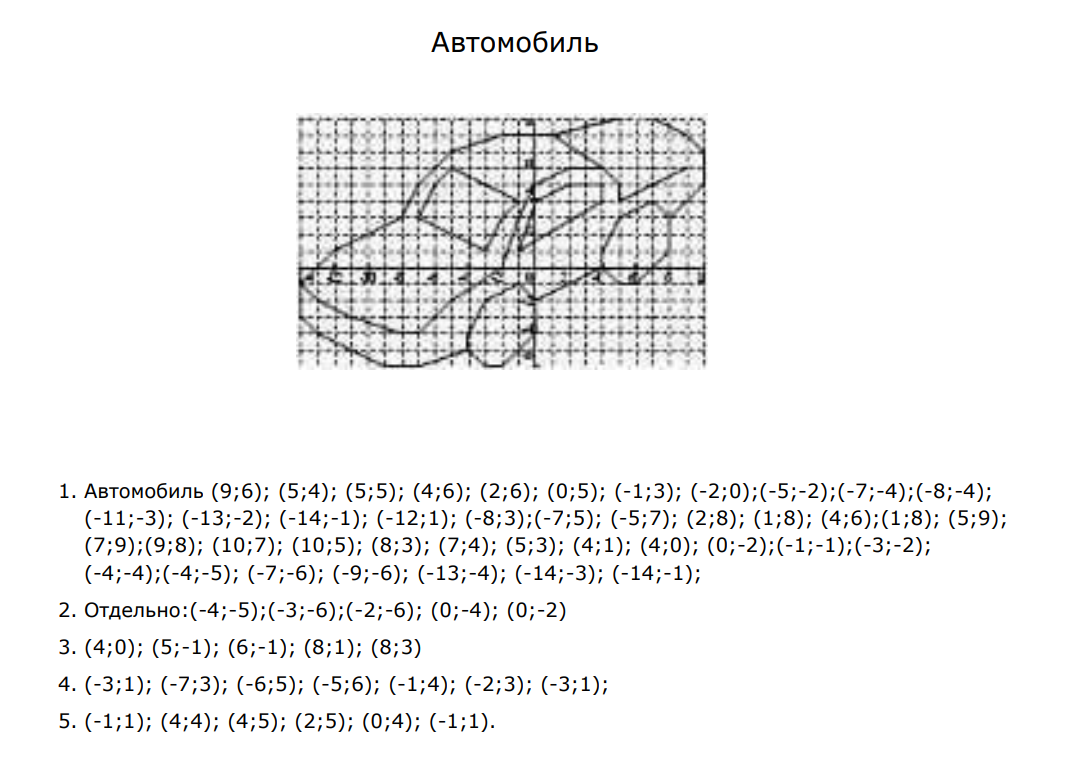


Рисунок 5.7 – Используемый шаблон

Как мы можем заметить — приложение сгенерировало точно то изображение, точки которого мы отправили.

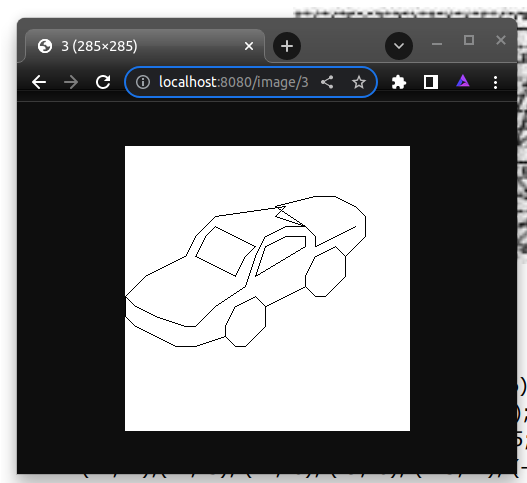


Рисунок 5.7 – Сгенерированное изображение

Работа приложения происходит без ошибок.

# 

# Литература

1. Хорстманн, К.С. Java 2. Библиотека профессионала, том1. Основы. 8-е издание, : Пер. с англ. / К. С. Хорстманн, Г. Корнелл - М.: ООО «Вильямс», 2012. - 816 с.
2. Дирк, Л. Самоучитель Java 7: Пер. с нем. / Л. Дирк, П. Мюллер. – СПб: БХВ-Петербург, 2013. – 464 с.: илл.
3. Шилдт, Г. Java. Полное руководство. 8-е издание, : Пер. с англ. / Г. Шилдт. – М.:ООО «Вильямс», 2012. – 1104 с.
4. Блинов, И.Н. Java. Промышленное программирование./ И.Н. Блинов, В.С. Романчик –Минск: «Четыре четверти», 2013. – 896 с.
5. Приложение постраение изображения по координатам [Электронный ресурс]. – Режим доступа: https://www.webmath.ru/web/prog30\_1.php. – Дата доступа: 17.11.2022.
6. Сайт для создания диаграм онлайн [Электронный ресурс]. – Режим доступа: https://app.diagrams.net/. – Дата доступа: 17.11.2022.
7. Шаблоны для рисования изображений по координатам [Электронный ресурс]. – Режим доступа: https://alekseevaee.ru/metodobespechenie/programmy/risuem-po-koordinatam. – Дата доступа: 17.11.2022.
8. Spring Boot [Электронный ресурс]. – Режим доступа: https://spring.io/projects/spring-boot. – Дата доступа: 17.11.2022.
9. HTML tutorial [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/html/. – Дата доступа: 17.11.2022.
10. CSS tutorial [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/css/. – Дата доступа: 17.11.2022.
11. Learn JS [Электронный ресурс]. – Режим доступа: https://learn.javascript.ru/. – Дата доступа: 17.11.2022.
12. IntelijIdea [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/idea/. – Дата доступа: 17.11.2022.

## Приложение А

Исходный код, откомпилированная программа и записка находятся на диске.