Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „Софтуерни и интернет технологии“

**Проект по „ООП-1“**

по дисциплината „Обектно-ориентирано програмиране 1 част проект”

на тема: „Хотел”

Тема 8

|  |  |
| --- | --- |
| Изготвил*: Християн Веселинов Керкенезов* | Проверил: |
| Специалност: СИТ |  |
| Група: 2 (б) |  |
| Факултетен номер: 22621617 |  |

2024

**Глава 1. Увод**

**1.1 Описание на проекта**

Проектът представлява разработка на информационна система за управление на хотел. Основната цел на програмата е да съхранява и обработва данни за стаите в хотела, предоставяйки удобен интерфейс за извършване на различни операции с тези данни.

Програмата работи с xml файл, в който се съхранява информацията за стаите, и ще предоставя функционалности за отваряне, записване и обработка на този файл.

**1.2 Идея на проекта**

Информационната система ще предоставя следните основни операции:

1. Отваряне на файл (***open***):

Зарежда данните за стаите от избран xml файл.

1. Затваряне на файл (***close***):

Затваря текущо отворения файл, без да запазва направените промени.

1. Записване на файл (***save***):

Запазва текущите данни в същия файл.

1. Записване на файл като (***save as***):

Запазва текущите данни в нов xml файл, избран от потребителя.

1. Помощ (***help***):

Показва списък с наличните команди и кратко описание за тяхното използване.

1. Изход (***exit***): Затваря програмата.

**1.3 Команди за управление на стаите**

1. ***checkin*** <room> <from> <to> <note> [<guests>]

Описание:

Регистриране в стая с номер <room> от дата <from> до дата <to> и се добавя бележка <note>. Незадължителният параметър <guests> задава броя на гостите, настанени в стаята. Ако той не е указан, се счита, че броят на настанените гости е равен на броя на леглата в стаята.

Пример: checkin 229 2020-03-23 2020-03-31 The Simpsons

1. ***availability*** [<date>]

Описание: Извежда списък на свободните стаи на дата <date>.

Ако датата не е зададена, се използва текущата дата.

Пример: availability 2020-03-23 или availability

1. ***checkout*** <room>

Описание: Освобождаване на заета стая с номер <room>.

Пример: checkout 229

1. ***report*** <from> <to>

Описание: Извежда справка за използването на стаи в периода от дата <from> до <to>.

Извежда се списък, в който за всяка стая, използвана в дадения период, се показва и броят на дните, в които е била използвана.

Пример: report 2020-01-01 2020-03-31

1. ***find*** <beds> <from> <to>

Описание: Намиране на подходяща свободна стая с поне <beds> на брой легла в периода от <from> до <to>.

При наличие на повече свободни стаи се предпочитат такива с по-малко на брой легла.

Пример: find 2 2020-03-23 2020-03-31

1. ***find!*** <beds> <from> <to>

Описание: Реализира алгоритъм за спешно намиране на стая за важен гост в случай на липса на свободни стаи за даден период.

Алгоритъмът предлага разместване на настанените от поне две стаи.

Пример: find! 2 2020-03-23 2020-03-31

1. ***unavailable*** <room> <from> <to> <note>

Описание: Обявява стаята с номер <room> от дата <from> до дата <to> за временно недостъпна и се добавя бележка <note>.

В стаята няма регистриран гост, но никой не може да бъде настанен в нея.

Пример: unavailable 200 2018-06-01 2019-03-01 Under construction

**1.4 Структура на проекта**

1. Основен клас (CommandFactory.java): Този клас ще съдържа основния метод, който отговаря за управлението на менютата и командите.
2. Клас за стаи (Room.java): Клас, който описва структурата на стаята с полета като номер, статус, брой легла, брой посетители, стартова дата, крайна дата, бележка и събития.
3. Клас хотел (Hotel.java): Този клас управлява стаите в хотела и предоставя удобен начин за извършване на различни операции свързани с тях.
4. Клас за обработка на файлове (FileManager.java): Клас, който ще се грижи за отварянето, записването и затварянето на файловете.

**1.5 Структура на документацията**

Документацията е структурирана в пет основни глави:

* Глава 1: Увод – кратко описание на проекта, целите и задачите на разработката, както и структурата на документацията.
* Глава 2: Преглед на предметната област – дефиниции, концепции, алгоритми, проблеми и методи за решаване.
* Глава 3: Проектиране – обща структура на проекта, диаграми и блок схеми.
* Глава 4: Реализация и тестване – детайли за реализацията на класовете, алгоритми, оптимизации и тестови сценарии.
* Глава 5: Заключение – обобщение на постигнатите цели и насоки за бъдещо развитие.

**2. Преглед на предметната област**

**2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани**

Тези дефиниции, концепции и алгоритми са от съществено значение за разработката на софтуера и ще бъдат използвани в различни части на приложението за обработка и манипулиране на xml данните:

 **Обектно ориентирано програмиране (ООП)**:

* Класове като Room, Hotel, Event, Reservation, FileManager и др., които са основните строителни блокове на вашия софтуер.
* Използването на обекти и техните методи за манипулиране на данни и изпълнение на функционалност.

 **Наследяване**:

* Вашите класове може да имат наследници или да бъдат наследени от други класове. Например, ако имате клас SingleRoom, той може да наследява от Room.

 **Полиморфизъм**:

* Използването на методи с едно и също име, но с различна имплементация, в различни класове. Например, методът Command() се използва в различните команди като SaveAsCommand, PrintEnrolledRoomsCommand, и т.н., но има различна имплементация във всяка от тях.

 **Алгоритми**:

* Вашите команди като SaveAsCommand и PrintEnrolledRoomsCommand извършват различни операции върху данните, което може да бъде разгледано като прилагане на алгоритми върху структурите от данни, които съхраняват информацията за хотела.

**2.2. Дефиниране на предизвикателства и сложност на проекта:**

1. **Управление на данни**:

Проектът изисква ефективно управление на голям обем от данни, включително информация за стаи, резервации и допълнителни бележки. Осигуряването на цялостността и сигурността на тези данни е от съществено значение за правилното функциониране на системата.

1. **Оптимизация на наличността**:

Системата трябва да може ефективно да определя подходяща стая за резервация, като отговаря на изискванията на клиента и минимизира времето за търсене и броя на заявките към базата данни. Това включва разработка на ефективни алгоритми за търсене и филтриране на данните.

1. **Интеграция с външни системи**:

Възможността за интеграция с външни системи, като онлайн резервационни платформи и финансови системи, изисква създаване на добре дефинирани интерфейси и осигуряване на сигурността на комуникацията. Също така е необходимо да се внимава за защита на данните и съответствие с регулаторните изисквания за защита на личните данни.

1. **Потребителски интерфейс и използваемост**: Проектът изисква разработка на потребителски интерфейс, който да бъде лесен за използване и навигация от потребителите. Това включва създаване на интуитивен дизайн и внимателно използване на цветове, икони и текст, за да се предостави ясна и разбираема информация.
2. **Тестване и поддръжка**: Необходимо е да се осигури задоволително ниво на тестване на системата, както и редовна поддръжка и актуализация на софтуера. Това включва изпълнение на функционални, интеграционни и удобството на потребителя тестове, както и поправки на грешки и добавяне на нови функционалности според нуждите на потребителите.

**2.3. Подходи, методи (евентуално модели и стандарти) за решаване на поставените проблемите**

1. **Бази данни и структуриране на данни**:

Използването на релационни бази данни за съхранение на информацията за стаите в хотела може да предложи ефективен начин за организация на данните. Моделирането на базата данни с подходящи таблиците, връзките и индексите може да улесни бързия достъп и манипулиране на данните.

1. **Алгоритми за търсене и оптимизация**:

Използването на алгоритми за търсене като двоично търсене или оптимизационни алгоритми като алгоритъмите за динамично програмиране може да помогне за намирането на подходяща стая за резервация и оптимизиране на наличността в хотела. Например, може да се използва двоично търсене за бързо намиране на свободни стаи в определен период или алгоритми за динамично програмиране за оптимално разпределение на резервациите във времето.

1. **Сигурност и защита на данните**:

Използването на стандарти и практики за сигурност на данните е от съществено значение за защитата на чувствителната информация. Криптирането на данни, управлението на достъпа и защита от атаки като SQL инжекции и XSS атаки са важни мерки за гарантиране на сигурността и цялостта на информацията. Така се предпазва информационната система на хотела от нарушения и компрометиране на данните.

**2.4. Потребителски (функционални) изисквания (права, роли, статуси, диаграми, ...) и качествени (нефункционални) изисквания (скалируемост, поддръжка, ...)**

1. **Скалируемост**: Системата трябва да е скалируема, за да се справи с увеличаващия се трафик и брой на потребителите.
2. **Поддръжка и разширяемост**: Лесна поддръжка и разширение с добре документирани API интерфейси.
3. **Безопасност и защита на данните**: Високо ниво на защита на данните с криптиране и управление на достъпа.

**3. Проектиране**

**3.1. Обща структура на проекта пакети който ще се реализират**

1. **Commands**: Този пакет съдържа класовете, които реализират различните команди, достъпни за потребителя. Включва следните класове:

* OpenCommand
* ExitCommand
* HelpCommand
* CloseCommand
* SaveCommand
* SaveAsCommand
* CheckInCommand
* AvailabilityCommand
* CheckOutCommand
* ReportCommand
* FindCommand
* FindNowCommand
* UnavailableCommand
* EnrollCommand
* UnrollCommand
* ShowEventsCommand
* PrintEnrolledRoomsCommanд

1. **Files**: Този пакет съдържа класове, отговарящи за обработка на файлове и управление на данни. Включва следните класове:

* DataHandler
* FileManager
* XmlStructure

1. **Model**: Този пакет съдържа класове, които представят моделите и функционалностите на системата за управление на хотела. Включва следните класове:

* Event
* Room
* Hotel
* Guest
* Reservation

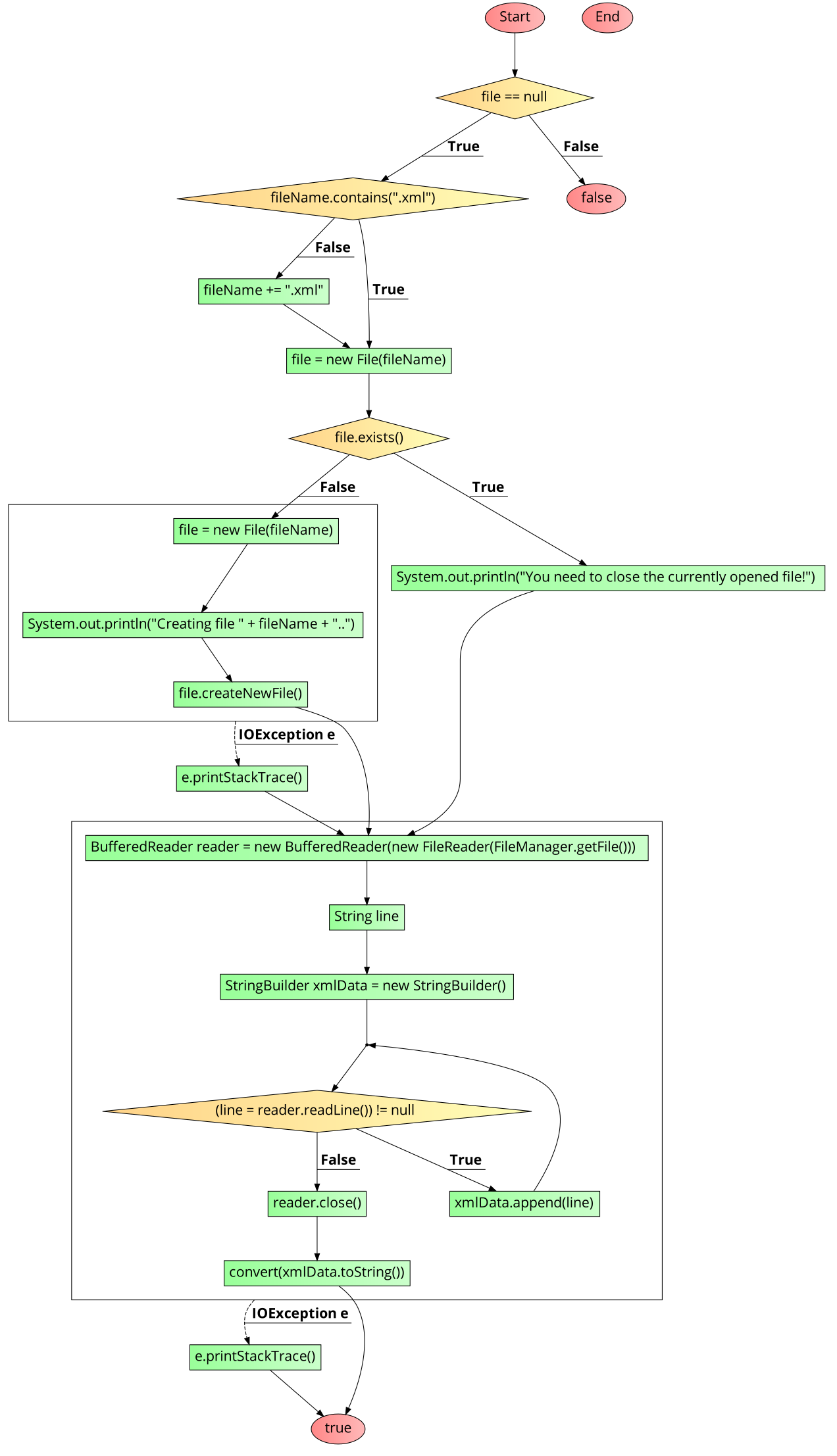
1. **UI**: Този пакет съдържа класове, които предоставят потребителски интерфейс и поддръжка на графичния интерфейс. Включва следните класове:

* ColorCode
* ProgramLauncher

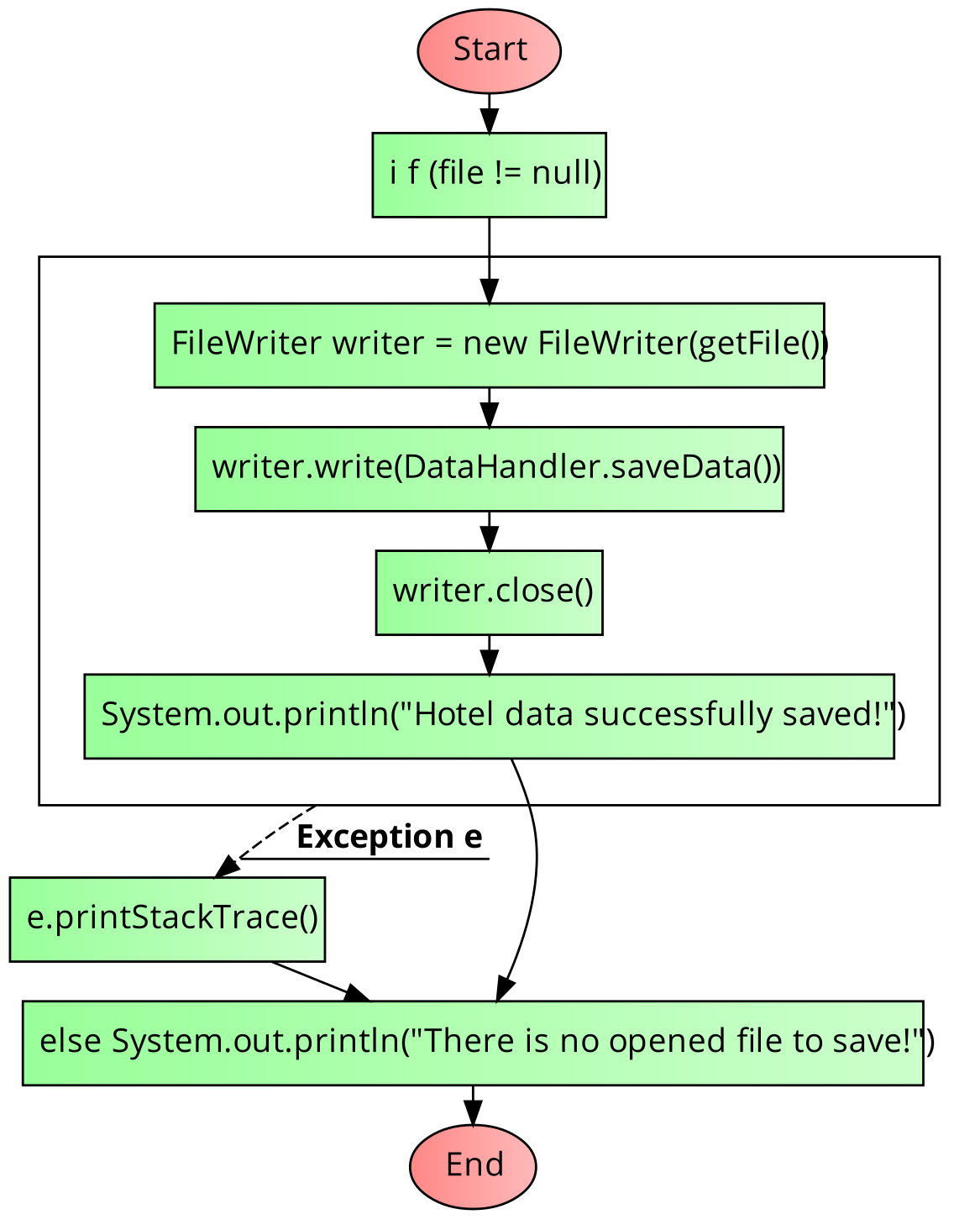
1. **Commands пакет**:
   * Този пакет съдържа класове, които представят конкретни команди, които потребителят може да използва за управление на хотелската информационна система.
   * Включва класове като **CheckInCommand**, **AvailabilityCommand**, **CheckOutCommand**, и т.н., като всеки клас представлява отделна команда.
   * Всеки от тези класове наследява абстрактния клас **Command**, който дефинира метода **execute()**, който се извиква при изпълнение на командата.
2. **CommandFactory клас**:
   * Този клас предоставя фабричен метод, който динамично създава инстанции на съответните команди в зависимост от входната команда от потребителя.
   * Командите се създават динамично, като се използва шаблона за дизайн "Factory Method", което позволява гъвкавост при добавянето на нови команди в бъдеще.
3. **Files пакет**:
   * Тук се намират класове, които са отговорни за обработката на файлове и данни в приложението.
   * Включва класове за четене, писане и манипулиране на данни във файлове, както и за обработка на информацията, получена от тях.
   * Може да включва и други вспомагателни класове, които улесняват работата с файловете и данните, като например класове за сериализация и десериализация.
4. **Model пакет**:
   * В този пакет се намират основните модели или класове на приложението, които представят основните същности и данни, с които приложението работи.
   * Включва класове като **Room**, **Hotel** и **Event**, които представят информация за стаи, хотели и събития в системата съответно.
   * Тук се съдържат и други вспомагателни класове или енумерации, които са необходими за функционирането на приложението, като например енумерация за типовете стаи или за събитията.

**3.2 Диаграми/Блок схеми(на структура и поведение -по обекти, слоеве с най-важните извадки от кода)**

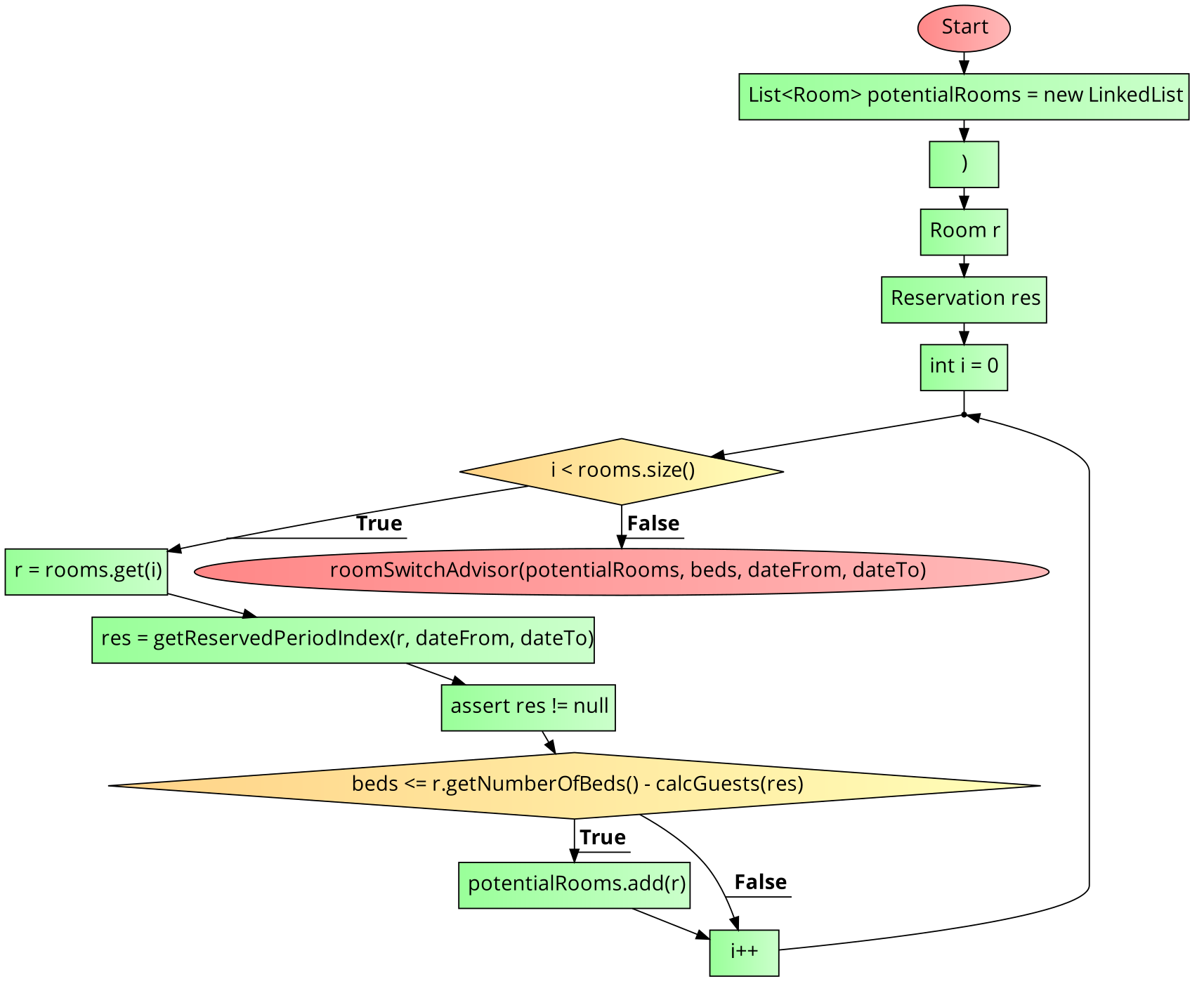
1. OpenFile



1. SaveFile



1. Find!



**4. Реализация, Тестване**

**4.1. Реализация на класове (включва важни моменти от реализацията на класовете и малки фрагменти от кода)**

1. **Hotel (Хотел):**
   * Класът Hotel представлява целия хотел и съдържа списък от всички стаи в хотела. Той осигурява функционалности за управление на тези стаи, като добавяне на нови стаи, проверка на наличността на стаи, регистриране на гости и други подобни операции. Класът Hotel играе централна роля в управлението на стаите и осигурява ефективна и удобна система за управление на хотела.

package Model;  
import UI.ColorCode;  
import java.time.LocalDate;  
import java.util.\*;  
import java.util.List;  
  
public class Hotel {  
 private static List<Room> *rooms*;  
  
 public Hotel() {  
 *rooms* = new LinkedList<>();  
 }  
  
 public static void generateRooms() {  
 int beds = 5;  
 for (int i = 1; i <= 400; i = i + 100) {  
 for (int j = 0; j <= 30; j++) {  
 if (i == 1) {  
 j = 1;  
 i = 0;  
 }  
 *rooms*.add(new Room(i + j, beds, null));  
 }  
 beds--;  
 }  
 }  
  
 public static void unavailable(int room, LocalDate dateFrom, LocalDate dateTo, String note) {  
 *checkIn*(room, dateFrom, dateTo, note, "Staff0");  
 }  
  
 public static void availability(LocalDate date) {  
 if (date == null) {  
 date = LocalDate.*now*();  
 System.*out*.println(ColorCode.*SUGGEST*.getCode() + "Using today's date for this search!");  
 }  
 Room r;  
 for (int i = 0; i < *rooms*.size(); i++) {  
 r = *rooms*.get(i);  
 if (r.checkIfAvailable(date, date)) {  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room " + r.getNumber() + " is available at " + date);  
 }  
 }  
 }  
  
 public static void report(LocalDate dateFrom, LocalDate dateTo) {  
 if (dateFrom == null || dateTo == null) {  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "Dates cannot be null!");  
 return;  
 }  
 boolean overallDataFound = false;  
  
 for (Room r : *rooms*) {  
 boolean dataFound = false;  
 int days = 0;  
 for (Reservation res : r.getReservations()) {  
 if (!(dateTo.isBefore(res.getFrom()) || dateFrom.isAfter(res.getTo()))) {  
 days += res.getTo().compareTo(res.getFrom());  
 dataFound = true;  
 overallDataFound = true;  
 }  
 }  
 if (dataFound) {  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room " + r.getNumber() + " is being used for " + days + " days in the specified period.");  
 }  
 }  
  
 if (!overallDataFound) {  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "No data of room use at that period!");  
 }  
 }  
  
 public static void checkIn(int number, LocalDate dateFrom, LocalDate dateTo, String note, String identity) {  
 int index = *findRoom*(number);  
 if (index != -1 && *getRooms*().get(index).checkIfAvailable(dateFrom, dateTo)) {  
 if (dateFrom.isBefore(LocalDate.*now*()) || dateTo.isBefore(LocalDate.*now*())) {  
 System.*out*.println(ColorCode.*SUGGEST*.getCode() + "Going back in time?\n" + ColorCode.*ERROR*.getCode() + "Reservations cannot be made for dates in the past!");  
 return;  
 }  
 int guests = *intParser*(identity);  
 if (*rooms*.get(index).getNumberOfBeds() >= guests) {  
 *rooms*.get(index).setNote(note);  
 if (guests == 0) {  
 guests = *rooms*.get(index).getNumberOfBeds();  
 }  
 *rooms*.get(index).addReservation(new Reservation(dateFrom, dateTo, identity.substring(0, identity.length() - 1), guests));  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room " + *rooms*.get(index).getNumber() + " has been checked in!");  
 } else {  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "This room only has " + *rooms*.get(index).getNumberOfBeds());  
 }  
 } else {  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "This rooms is unavailable!");  
 }  
 }  
  
 private static int intParser(String value) {  
 char ch = value.charAt(value.length() - 1);  
 try {  
 return Integer.*parseInt*(String.*valueOf*(ch));  
 } catch (NumberFormatException e) {  
 return 0;  
 }  
 }  
  
 public static int findRoom(int number) {  
 for (int i = 0; i < *rooms*.size(); i++) {  
 if (*rooms*.get(i).getNumber() == number) {  
 return i;  
 }  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "There is no such room!");  
 return -1;  
 }  
  
 public static int find(int beds, LocalDate dateFrom, LocalDate dateTo) {  
 if (beds >= 1 && beds <= 5) {  
 Room r;  
 int count = 0;  
 for (int i = 0; i < *getRooms*().size(); i++) {  
 r = *getRooms*().get(i);  
 if ((r.getNumberOfBeds() >= beds) && r.checkIfAvailable(dateFrom, dateTo)) {  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room number " + r.getNumber() + " is available!");  
 count++;  
 }  
 }  
 return count;  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "The hotel contains rooms with 1-5 beds!");  
 return -1;  
 }  
  
 public static Integer[][][] findNow(int beds, LocalDate dateFrom, LocalDate dateTo) {  
 List<Room> potentialRooms = new LinkedList<>();  
 Room r;  
 Reservation res;  
 for (int i = 0; i < *rooms*.size(); i++) {  
 r = *rooms*.get(i);  
 res = *getReservedPeriodIndex*(r, dateFrom, dateTo);  
 assert res != null;  
 if (beds <= r.getNumberOfBeds() - *calcGuests*(res)) {  
 potentialRooms.add(r);  
 }  
 }  
 return *roomSwitchAdvisor*(potentialRooms, beds, dateFrom, dateTo);  
 }  
  
 private static Reservation getReservedPeriodIndex(Room room, LocalDate from, LocalDate to) {  
 for (Reservation res : room.getReservations()) {  
 if (from.isBefore(res.getTo()) && to.isAfter(res.getFrom())) {  
 return res;  
 }  
 }  
 return null;  
 }  
  
 private static int calcGuests(Reservation reservation) {  
 int sum = 0;  
 for (Guest g : reservation.getGuests()) {  
 sum += g.getNumber();  
 }  
 return sum;  
 }  
  
 private static Integer[][][] roomSwitchAdvisor(List<Room> potentialRooms, int beds, LocalDate from, LocalDate to) {  
 if (!potentialRooms.isEmpty()) {  
 Integer[][][] indexLink = new Integer[potentialRooms.size()][*rooms*.size()][2];  
 Reservation resM, resN;  
 int m, n, counter;  
 for (int i = 0; i < indexLink.length; i++) {  
 m = *findRoom*(potentialRooms.get(i).getNumber());  
 counter = 0;  
 for (int j = 0; j < *rooms*.size(); j++) {  
 n = j;  
 resM = *getReservedPeriodIndex*(*rooms*.get(m), from, to);  
 resN = *getReservedPeriodIndex*(*rooms*.get(n), from, to);  
 if (*rooms*.get(m).getNumber() != *rooms*.get(n).getNumber() &&  
 (*rooms*.get(m).getNumberOfBeds() - *calcGuests*(resM)) >= *calcGuests*(resN) &&  
 *rooms*.get(n).getNumberOfBeds() >= beds) {  
 indexLink[i][counter][0] = m;  
 indexLink[i][counter][1] = n;  
 counter++;  
 }  
 }  
 }  
 if (indexLink.length != 0) {  
 for (int i = 0; i < indexLink.length; i++) {  
 for (int k = 0; k < indexLink[i].length; k++) {  
 if (indexLink[i][k][0] != null) {  
 System.*out*.println(ColorCode.*NORMAL*.getCode() + "Room " + *rooms*.get(indexLink[i][k][0]).getNumber() + " (" + *calcGuests*(*getReservedPeriodIndex*(*rooms*.get(indexLink[i][k][0]), from, to)) + " | " + *rooms*.get(indexLink[i][k][0]).getNumberOfBeds() + " guests) <- Room " + *rooms*.get(indexLink[i][k][1]).getNumber() + " (" + *calcGuests*(*getReservedPeriodIndex*(*rooms*.get(indexLink[i][k][1]), from, to)) + " | " + *rooms*.get(indexLink[i][k][1]).getNumberOfBeds() + " guests)");  
 }  
 }  
 }  
 return indexLink;  
 }  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "Hotel is completely full!");  
 return null;  
 }  
  
 public static void moveGuests(Integer[][][] indexLink, LocalDate from, LocalDate to) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println(ColorCode.*NORMAL*.getCode() + "Do you want move the guests to another room?\n" +  
 ColorCode.*DONE*.getCode() + "Yes" + ColorCode.*CLEAR*.getCode() + " or " + ColorCode.*ERROR*.getCode() + "No" +  
 ColorCode.*NORMAL*.getCode() + "\n");  
 System.*out*.print(">");  
 String answer = scanner.nextLine();  
 if (answer.equalsIgnoreCase("yes")) {  
 System.*out*.println(ColorCode.*NORMAL*.getCode() + "Select the room you want to move other guests to:");  
 List<Integer> temp = new ArrayList<>();  
 for (int i = 0; i < indexLink.length; i++) {  
 for (int j = 0; j < indexLink[i].length; j++) {  
 if (indexLink[i][j][0] != null) {  
 if (!temp.contains(indexLink[i][j][0])) {  
 temp.add(indexLink[i][j][0]);  
 System.*out*.print(*rooms*.get(indexLink[i][j][0]).getNumber() + ", ");  
 }  
 }  
 }  
 }  
 System.*out*.print("\n>");  
 answer = scanner.nextLine();  
 if (temp.contains(*findRoom*(Integer.*parseInt*(answer)))) {  
 int receiverId = *findRoom*(Integer.*parseInt*(answer));  
 System.*out*.println(ColorCode.*NORMAL*.getCode() + "Select the room that will be cleared:");  
 temp.clear();  
 for (int i = 0; i < indexLink.length; i++) {  
 for (int j = 0; j < indexLink[i].length; j++) {  
 if (indexLink[i][j][0] != null) {  
 temp.add(indexLink[i][j][1]);  
 System.*out*.print(*rooms*.get(indexLink[i][j][1]).getNumber() + ", ");  
 }  
 }  
 }//find! 2 2024-10-10 2024-10-20  
 System.*out*.print("\n>");  
 answer = scanner.nextLine();  
 if (temp.contains(*findRoom*(Integer.*parseInt*(answer)))) {  
 int senderId = *findRoom*(Integer.*parseInt*(answer));  
 Reservation receiver = *getReservedPeriodIndex*(*rooms*.get(receiverId), from, to);  
 Reservation sender = *getReservedPeriodIndex*(*rooms*.get(senderId), from, to);  
 assert sender != null;  
 assert receiver != null;  
 receiver.getGuests().addAll(sender.getGuests());  
 *rooms*.get(senderId).removeReservation(sender);  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room swap complete!");  
 }  
 return;  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "This room isn't listed");  
 return;  
 }  
 System.*out*.println(ColorCode.*NORMAL*.getCode() + "No changes have been made!");  
 }  
  
 public static void checkOut(int number, Integer resId) {  
 int index = *findRoom*(number);  
 int id = resId - 1;  
 if (index != -1) {  
 if (!*rooms*.get(index).getReservations().isEmpty()) {  
 if (resId == -1) {  
 *rooms*.get(index).getReservations().clear();  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room " + *rooms*.get(index).getNumber() + " has been checked out!");  
 return;  
 } else if (*rooms*.get(index).getReservations().size() >= resId) {  
 *rooms*.get(index).getReservations().remove(id);  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room " + *rooms*.get(index).getNumber() + " has been checked out!");  
 return;  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "Reservation doesn't exist!");  
 return;  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "This room is already available!");  
 return;  
 }  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "Room number " + number + " doesn't exist!");  
 }  
  
 public static List<Room> getRooms() {  
 return *rooms*;  
 }  
  
 public static void setRooms(List<Room> rs) {  
 *rooms* = rs;  
 }  
}

**4.2. Алгоритми и Оптимизации**

В процеса на разработка са приложени оптимизации към алгоритмите, осигуряващи ефективност и бързодействие на системата. Тук може да включите конкретни алгоритми и тяхната оптимизация, ако има такива.

package Commands;  
  
import Model.Event;  
import Model.Guest;  
import Model.Hotel;  
import Model.Reservation;  
import UI.ColorCode;  
  
import java.util.List;  
  
public class PrintEnrolledRoomsCommand extends Command {  
 @Override  
 public void Command(List<String> args) {  
 if (args.size() == 1) {  
 int eventId = Integer.*parseInt*(args.get(0));  
 Event event = Event.*getEvent*(eventId);  
 for (int i = 0; i < Hotel.*getRooms*().size(); i++) {  
 for (Reservation r : Hotel.*getRooms*().get(i).getReservations()) {  
 for (Guest g : r.getGuests()) {  
 if (g.getEvents().contains(event)) {  
 System.*out*.println(ColorCode.*DONE*.getCode() + "Room " + Hotel.*getRooms*().get(i).getNumber());  
 }  
 }  
 }  
 }  
 } else {  
 System.*out*.println(ColorCode.*ERROR*.getCode() + "This command requires 1 argument (event ID)!");  
 }  
 }  
}

**4.3. Планиране, описание и създаване на тестови сценарии**

<Room Number="1">  
 <NumberOfBeds>5</NumberOfBeds>  
 <Note>note</Note>  
 <Reservations>  
 <Reservation>  
 <DateFrom>2024-10-10</DateFrom>  
 <DateTo>2024-10-20</DateTo>  
 <GuestDetails>  
 <Guests>  
 <Identity>family</Identity>  
 <NumberOfGuests>5</NumberOfGuests>  
 <Events>  
 <Event>1</Event>  
 </Events>  
 </Guests>  
 </GuestDetails>  
 </Reservation>  
 </Reservations>  
</Room>

**5. Заключение**

**5.1. Обобщение на изпълнението на началните цели**

Проектът е успешно реализиран и постигна всички първоначално поставени цели. Класовете за команди предоставят функционалности за управление на хотелски стаи, включително регистрация, проверка на наличност, освобождаване на стая, генериране на отчети и други.

Класът Hotel управлява списъка със стаи и осигурява методи за тяхното създаване, маркиране като недостъпни, проверка на наличност и други. Моделът съдържа класове за стая, хотел и събитие, които се използват във функционалностите на приложението.

**5.2. Насоки за бъдещо развитие и усъвършенстване**

• Подобряване на гъвкавостта и разширяемостта: Разгледайте възможностите за добавяне на допълнителни функционалности към класовете за команди и моделите, които биха увеличили гъвкавостта и разширяемостта на приложението.

• Оптимизация на производителността: Проучете възможностите за оптимизация на алгоритмите и структурите от данни, които се използват в проекта, за да се осигури по-бърза и ефективна обработка на данните.

• Разширяване на функционалността: Разгледайте възможностите за добавяне на допълнителни функции към приложението, които биха повишили неговата полезност и удобство за потребителите.

**5.3. GitHub Repository URL:**

https://github.com/Paakk0/Project1