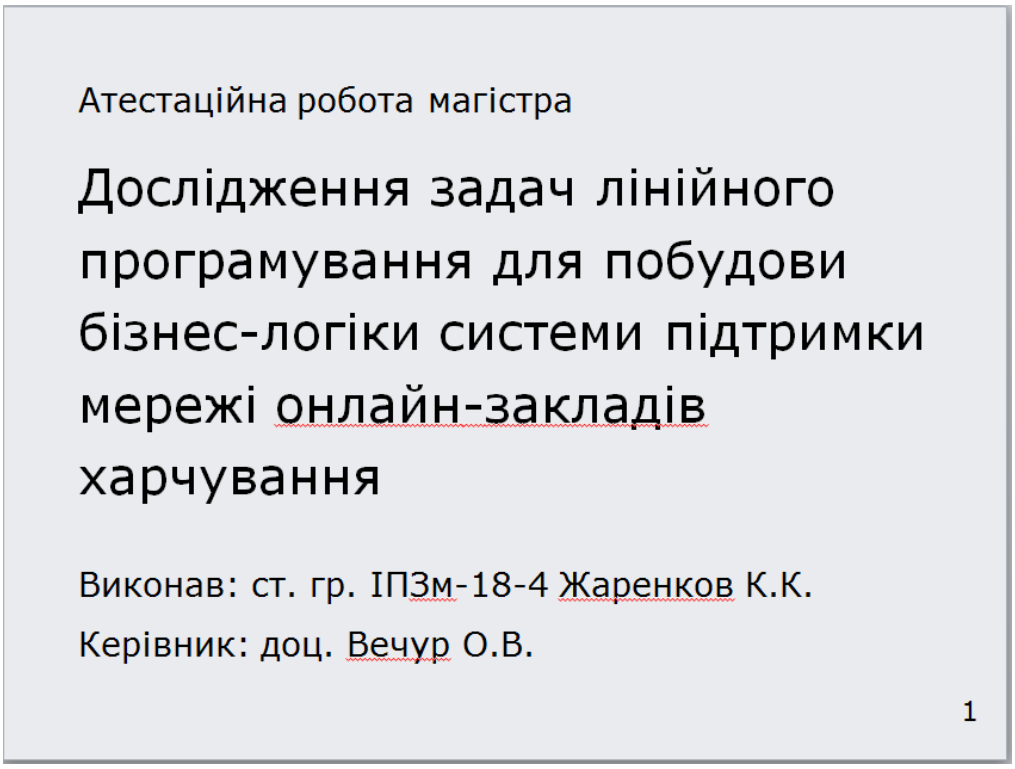
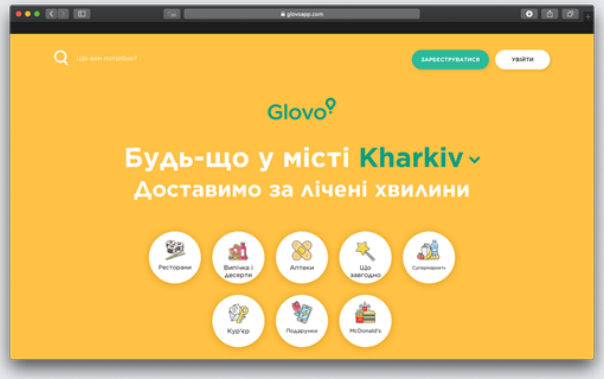
ДОДАТОК А

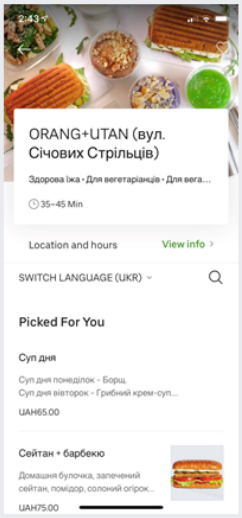
Слайди презентації



Актуальність дослідження



Інтерфейс користувача в Glovo



Інтерфейс користувача в Uber Eats

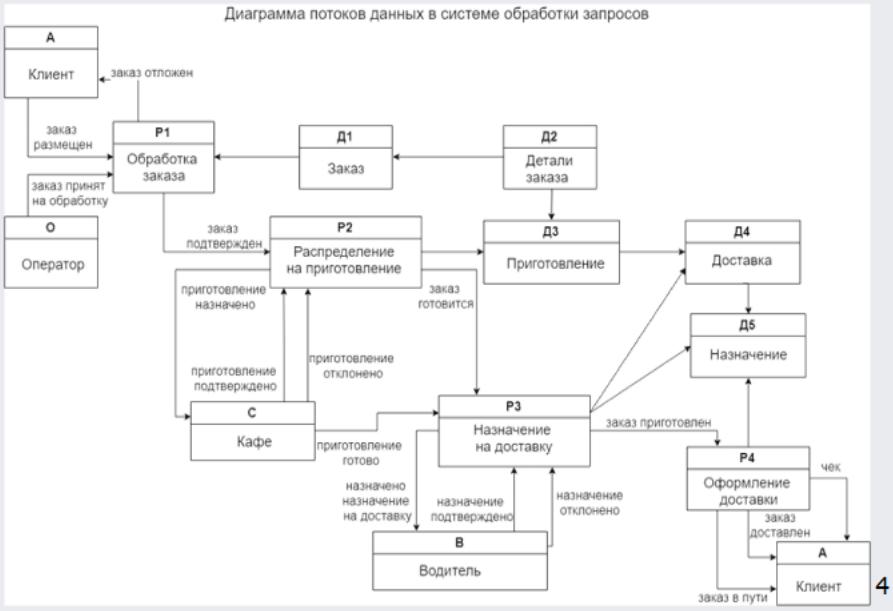
2

Постановка задачі

- провести аналіз бізнес-логіки та моделювання предметної області роботи мережі онлайн-закладів харчування;
- провести дослідження задач лінійного програмування та методів їх вирішення, обрати найкращі для моделювання основних процесів бізнес-логіки мережі онлайн-закладів харчування;
- розробити математичні моделі ЗЛП для виявлених основних процесів бізнес-логіки;
- розробити схему бази даних для збереження інформації предметної області та розроблених математичних моделей;
- розробити алгоритми вирішення промодельованих ЗЛП;
- розробити архітектуру та програмно реалізувати систему підтримки роботи онлайн-закладів харчування.

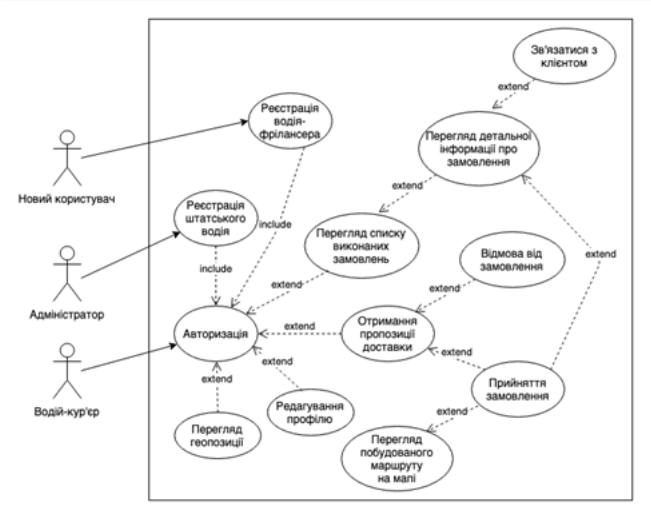
3

Аналіз предметної області



4

Моделювання предметної області



Діаграма прецедентів для бізнес-задачі призначення на доставку 5

Дослідження задач лінійного програмування

Задачі математичного програмування:

- **задачі лінійного програмування:**
  - розподільчі задачі:
  - задачі про призначення;
  - задачі про суміші;
  - задачі про розкрій або розпил;
  - транспортні здачі та інші
- задачі динамічного програмування ;
- інші;

Табличне надання оптимізаційної задачі лінійного програмування

	Роботи						Об'єм наявних ресурсів $R_i^{\square}$
Ресурси	J_1	J_2	...	J_j	...	J_n	
$R_1^{\square}$	C_{11}	C_{12}	...	C_{1j}	...	C_{1n}	b_1
...	...	...	...	...	...	...	...
$R_i^{\square}$	C_{i1}	C_{i2}	...	C_{ij}	...	C_{in}	b_i
...	...	...	...	...	...	...	...
$R_m^{\square}$	C_{m1}	C_{m2}	...	C_{mj}	...	c_{mn}	b_m
Об'єм необхідних ресурсів $R_i^{\square}$	a_1	a_2	...	a_j	...	a_n	

Похідні дані для моделювання

(задача розподілу великих замовлень для приготування)

Необхідно врахувати:
а) потужності закладів харчування (ЗХ);
б) технологію приготування (час приготування);
в) час, на який було замовлення;
г) час на доставку та інше.

	Деталі замовлення,які необхідно приготувати			
Заклади харчування	Піца з салямі	Картопля «Фрі»	Кекс	Лимонад
ЗХ № 1	Max 10 порцій	Max 10 порцій	Max 40 порцій	Max 60 порцій
ЗХ № 2	Max 50 порцій	Max 50 порцій	Max 40 порцій	Max 70 порцій
ЗХ № 3	Max 50 порцій	Max 50 порцій	Max 40 порцій	Max 60 порцій
ЗХ № 4	Max 50 порцій	Max 50 порцій	Max 20 порцій	Max 70 порцій
ЗХ № 5	Max 50 порцій	Max 30 порцій	Max 20 порцій	Max 70 порцій
Об'єм необхідних порцій	80 порцій	100 порцій	60 порцій	100 порцій

7

Математична модель задачі розподілу замовлення

$$F = \sum_{i=1}^{count\_C} (Price_{ik} * Sgn\,(\sum_{j=1}^{count\_D_k} X_{ij})) \to min$$

$$\sum_{i=1}^{count\_C} X_{ij} \geq count\_P_j, \quad \forall\, j = \overline{1, count\_D_k}$$

$$0 \leq X_{ij} \leq p_j^i, \qquad \forall\, i = \overline{1, count\_C}, \qquad \forall\, j = \overline{1, count\_D_k}$$

Промодельовані обмеження:
- кожна деталь замовлення D_j^k повинна бути приготовлена в кількості не менше ніж було замовлено ($count\_P_j$);
- кількість X_{ij}, що буде розподілена для приготування в ЗХ повинна бути позитивною та не повинна перевищувати поточну потужність p_j^i ЗХ з приготування цього типу порцій.

8

Табличне надання оптимізаційної задачі з призначення водіїв на доставку

	Роботи з доставки					Кіл-сть наявних водіїв Dr
Водії	$J_1(1,1)$	…	$J_h(i,k)$	…	$J_{count\_J}(.,.)$	
Dr_1	$Price_{11}$	…	$Price_{1h}$	…	$Price_{1count\_J}$	1
Dr_2	$Price_{21}$		$Price_{2h}$	…	$Price_{2count\_J}$	1
…	…	…	…	…	…	…
Dr_l	$Price_{l1}$	…	$Price_{lh}$	…	$Price_{lcount\_J}$	1
…	…	… .	…	…	…	…
Dr_{count_Dr}	$Price_{count\_Dr1}$	…	$Price_{count\_Drh}$	…	$Price_{Count\_Dr}$	1
Кіл-сть не обхідних водіїв Dr	1	1	1	…	1	

9

Вагові обмеження

	Роботи з доставки					Вагові обме-ження водіїв Dr
Водії	$J_1(1,1)$	…	$J_h(i,k)$	…	$J_{count\_J}(.,.)$	
Dr_1	X_{11}	…	X_{1h}	…	X_{1count_J}	Wd_1
Dr_2	X_{21}	…	X_{2h}	…	X_{2count_J}	Wd_2
…	…	…	…	…	…	…
Dr_l	X_{l1}	…	X_{lh}	…	X_{lcount_J}	Wd_l
…	…	…	…	…	…	…
Dr_{count_Dr}	$X_{count\_Dr\,1}$	…	$X_{count\_Dr\,h}$	…	$X_{count\_Dr\,count}$	Wd_{count_Dr}
Вага доставок	W_1		W_h		W_{count_J}	

$$\sum_{h=1}^{count\_J} W_h * X_{lh} \le Wd_l \quad ,\forall l = \overline{1, count\_Dr}_i$$

10

Математична модель задачі про призначення водія на доставку

$$F = \sum_{l=1}^{count\_Dr} \sum_{h=1}^{count\_J} Price_{lh} * X_{lh} \to min,$$

$$\sum_{h=1}^{count\_J} X_{lh} \le 1, \quad \forall\, l = \overline{1, count\_Dr},$$

$$\sum_{l=1}^{count\_Dr} X_{lh} = 1, \quad \forall h = \overline{1, count\_J},$$

$$\sum_{l=1}^{count\_Dr} Tprob_{lh} * X_{lh} \le Torder_h, \quad \forall h = \overline{1, count\_J}$$

$$\sum_{h=1}^{count\_J} W_h * X_{lh} \le Wd_l, \quad \forall l = \overline{1, count\_Dr}$$

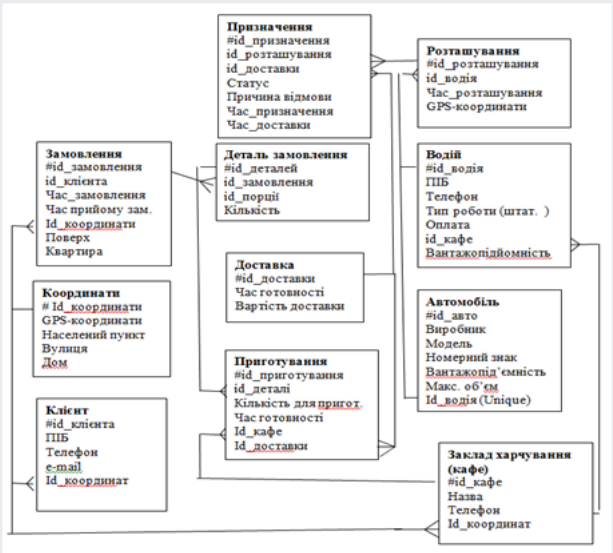
$$\sum_{h=1}^{count\_J} V_h * X_{lh} \le Vd_l, \quad \forall l = \overline{1, count\_Dr}$$

$$X_{lh} = 1 \text{ або } X_{lh} = 0$$

X_{lh} – змінна, що показує чи призначено водія Dr_l на роботу $J_h(i,k)$ з доставки з *i*-го ЗХ за *k*-ю адресою замовлення;
$Price_{lh}$ – вартість доставки *l*-тим водієм роботи $J_h(i,k)$ до її клієнта;
$Tprob_{lh}$- ймовірний час доставки *l*-тим водієм роботи $J_h(i,k)$;
$Torder_h$ - час, на який було замовлення;
W_h - вантажопідйомність доставки;
Wd_l - вантажопідйомність авто водія;
Vd_l - максимальний об'єм авто водія;
V_h - об'єм доставки.

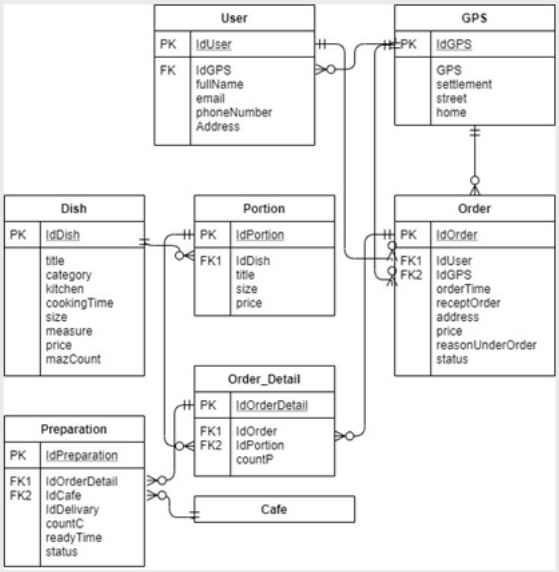
11

Фрагмент ER-діаграми (задача призначення водіїв)



12

Фрагмент схеми БД (задача розподілу замовлення)



13

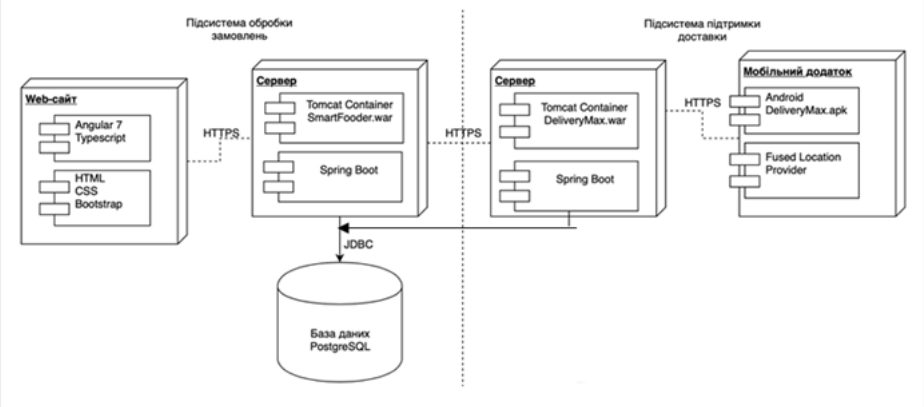
Розробка алгоритмів вирішення оптимізаційних задач

- графічний метод;
- симплекс-метод;
- «угорський» алгоритм;
- **«жадібний» алгоритм;**
- та інші.

Скорочена блок схема жадібного алгоритму призначення водія на доставку

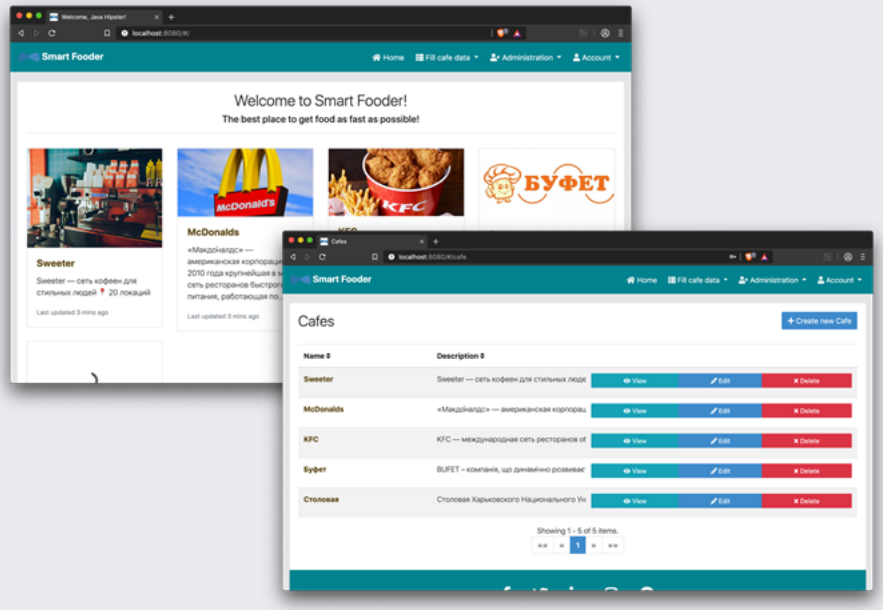
14

Архітектура програмної системи



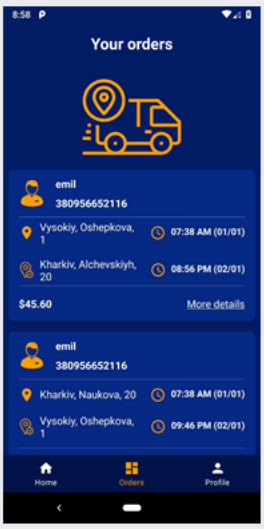
15

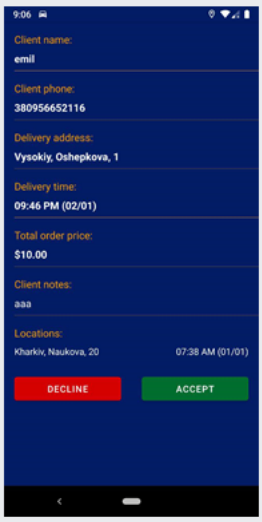
Веб-додаток клієнта



16

Мобільний додаток водія





17

Висновки

- проведено аналіз бізнес-логіки та моделювання предметної області роботи мережі онлайн-закладів харчування;
- проведено досліждення задач лінійного програмування та методів їх вирішення, обрано найкращі для моделювання основних процесів бізнес-логіки мережі онлайн-закладів харчування;
- розроблено математичні моделі ЗЛП для виявлених основних процесів бізнес-логіки;
- розроблено схему бази даних для збереження інформації предметної області та розроблених математичних моделей;
- розролено алгоритми вирішення промодельованих ЗЛП;
- розроблено архітектуру та програмно реалізовано систему підтримки роботи онлайн-закладів харчування;
- подано тези доповіді на Міжнародний молодіжному форуму «Радіоелектроніка та молодь в XXI ст.».

18

ДОДАТОК Б

Тези доповіді

ДОСЛІДЖЕННЯ ЗАДАЧ ЛІНІЙНОГО ПРОГРАМУВАННЯ ДЛЯ ПОБУДОВИ БІЗНЕС-ЛОГІКИ ОНЛАЙН-ЗАКЛАДІВ ХАРЧУВАННЯ

Жаренков К.К.
Научный руководитель – к.т.н., доц. Вечур О.В.
Харьковский национальный университет радиоэлектроники
(61166, Харьков, пр. Науки, 14, каф. Программной инженерии, тел. (057) 702-14-46)
e-mail: kyrylo.zharenkov@nure.ua

The work is devoted to the study of linear programming problems with the aim of applying them to simulate the business logic of online cafes. The paper proposes model of the optimization distribution problem of distributing large orders between online cafes for cooking and model of the optimization problem of assigning drivers to deliver such orders. Based on the developed models, an application server has been developed for the online cafe network support system.

В сучасному динамічному світі все більш популярним стає замовлення їжі через Інтернет в мережах онлайн-закладів харчування. Отже, бізнес-логіка таких закладів орієнтована на обробку великих обсягів замовлень, на обробку замовлень різних об’ємів та своєчасну доставку таких замовлень в будь-який куточок світу. Існуючі на сьогодні системи автоматизації ресторанного бізнесу не завжди в змозі підтримати все різноманіття виникаючих бізнес-задач, що може стати причиною затримок в обслуговування клієнтів.

Достатньо цікавими та складними для підтримки є такі бізнесові задачі як:

- розподілення навантаження з приготування великих замовлень між декількома закладами харчування в мережі для своєчасного їх виконання;
- розподілення завдань на доставки між водіями таких мереж, а інколи й залучення водіїв-фрілансерів.

Практичні задачі такого класу можна ефективно вирішити на базі оптимізаційних задач [1], зокрема, задач лінійного програмування. Розглянуті бізнес-задачі належать до розподільчих задач та можуть бути вирішені на базі оптимізаційних моделей задач про призначення та розподілення.

Таким чином, була поставлена задача розробити оптимізаційні моделі задач розподілення великих замовлень між закладами харчування та призначення водіїв на роботи з доставки цих замовлень, а також розробити алгоритми вирішення таких задач та програмно реалізувати їх в складі системи підтримки онлайн-закладів харчування.

Була розроблена оптимізаційна модель задачі розподілення великого замовлення між закладами харчування для приготування, Цільова функція мінімізує сумарну вартість доставки усіх частин замовлення від закладів

харчування до клієнта. Модель дозволяє врахувати обмеження з виробничих потужностей закладів харчування; з наявності в них продуктів, що необхідні для приготування частин замовлення та інше..

Була розроблена оптимізаційна модель задачі про призначення водіїв на доставку

$$F = \sum_{l=1}^{c\_Dr} \sum_{h=1}^{c\_J} P_{lh} * X_{lh} \to min$$

$$\sum_{h=1}^{c\_J} X_{lh} \leq 1, \ \ \forall\, l = \overline{1, c\_Dr},$$

$$\sum_{l=1}^{c\_Dr} X_{lh} = 1, \ \ \forall h = \overline{1, c\_J},$$

$$X_{lh} = 1 \ \text{ або } X_{lh} = 0, \text{ де}$$

P_{lh} – вартість доставки l-тим водієм частини замовлення $J_h(i,k)$ до його клієнта; X_{ij} — ознака чи потрібно призначати i-го водія на j-ту доставку. Наведені обмеження моделюють можливість призначення кожного водія не більше ніж на доставку однієї частини замовлення; призначення для доставки кожної частини замовлення тільки одного водія, а також можливість призначати або ні водія. Модель розширена часовими обмеженнями та обмеженнями стосовно співвідношення об’єму замовлення и вмисності транспортного засобу водія.

Для вирішення розроблених задач було обрано «жадібний» алгоритм [2], який максимально швидко дозволяє знайти найкраще рішення даних задач. Для організації та зберігання інформації з моделей була розроблена база даних, що містить інформацію про заклади харчування та їх продуктових запасах, клієнтах та їх замовленнях; стравах та їх складових; водіях та їх транспортних засобах.

Була розроблена програмна система підтримка мережі онлайн-закладів харчування на основані клиєнт-серверної архітектури, що складається з мобільного додатку для водіїв, вед-додатка для оператора мережі, сервера даних та прикладного сервера для реалізації бізнес-логіки (вирішення оптимізаційних задач),

Перелік посилань: 1. Васильев, Ф. П. Линейное программирование / Ф.П. Васильев, А.Ю. Иваницкий. - М.: Факториал Пресс, 2016. - 352 с.; 2. Наконечний О.Г. Методи прийняття рішень : навч. посіб. / О. Г. Наконечний, І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев ; Мін-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2016. – 132 с. : іл.

ДОДАТОК В

Лістинг коду програмної реалізації алгоритму пошуку водіїв

```
@Component("orderHandler")
public class OrderHandler {

    private static final long DELAY_PERMANENT_DRIVER_SELECTION = 180_000L;
    private static final String FIREBASE_REFERENCE_DELIVERIES = "deliveries";

    private OrderService orderService;
    private DriverService driverService;
    private DeliveryScheduler deliveryScheduler;
    private GreedyAlgorithm greedyAlgorithm;
    private MessageSender messageSender;

    @Autowired
    public OrderHandler(
            OrderService orderService,
            DriverService driverService,
            DeliveryScheduler deliveryScheduler,
            GreedyAlgorithm greedyAlgorithm,
            MessageSender messageSender
    ) {
        this.orderService = orderService;
        this.driverService = driverService;
        this.deliveryScheduler = deliveryScheduler;
        this.greedyAlgorithm = greedyAlgorithm;
        this.messageSender = messageSender;
    }

    public void processOrder(Order order) {
        processOrder(order, order.getStartPart(), null);
    }

    private void processOrder(Order order, OrderPart orderPart, Driver driver) {
        long currentTime = System.currentTimeMillis();
        long startTime = orderPart == null ? -1 : orderPart.getReadinessTime();
        if (startTime == -1L) return;

        if (TimeUnit.MILLISECONDS.toHours(startTime) - TimeUnit.MILLISECONDS.toHours(currentTime) > 1) {
            deliveryScheduler.execute(() -> findAvailableDriver(order, orderPart, driver, Collections.emptyList(), true),
                    startTime - TimeUnit.MINUTES.toMillis(DurationEvaluator.DRIVER_NOTIFYING_TIME_MINUTES));
```

```
        } else {
            findAvailableDriver(order, orderPart, driver,
Collections.emptyList(), true);
        }
    }

    private void findAvailableDriver(Order order, OrderPart orderPart,
Driver driver, List<Long> exceptionIds, boolean shouldSchedule) {
        Driver foundDriver = driver != null ? driver :
greedyAlgorithm.findAnyDriver(orderPart, driverService.getDrivers(),
exceptionIds);

        if (foundDriver != null && foundDriver.getIsFreelancer() &&
messageSender.sendMessage(foundDriver, order.getId(),
orderPart.getId())) {
            subscribeForUpdates(order, orderPart, result -> {
                if (result.getAccepted()) {
                    driverService.findDriverById(result.getDriverId())
                            .ifPresent(it ->
snapToOrderAndFindOtherIfNeeded(order, orderPart, it));
                } else {
                    exceptionIds.add(result.getDriverId());
                    findAvailableDriver(order, orderPart, null,
exceptionIds, false);
                }
            });

            if (shouldSchedule) {
                deliveryScheduler.execute(() ->
orderService.findOrderById(order.getId())
                        .ifPresent(it -> {
                            OrderPart part =
it.getOrderPartById(orderPart.getId());

                            if (part.getDriver() == null) {
                                Driver d =
greedyAlgorithm.findPermanentDriver(part, driverService.getDrivers());
                                setPermanentDriver(it, part, d);
                            }
                        }), System.currentTimeMillis() +
DELAY_PERMANENT_DRIVER_SELECTION);
            }

        } else if (foundDriver != null) {
            setPermanentDriver(order, orderPart, foundDriver);
        }
    }

    private void setPermanentDriver(Order order, OrderPart orderPart,
Driver driver) {
        if (driver == null) {
            System.err.println("There is no permanent driver!");
        } else {
            if (messageSender.sendMessage(driver, order.getId(),
orderPart.getId())) {
```

```
                snapToOrderAndFindOtherIfNeeded(order,      orderPart,
driver);
            }
        }
    }

    private    void    snapToOrderAndFindOtherIfNeeded(Order    order,
OrderPart          orderPart,          Driver          driver)          {
        orderPart.setDriver(driver);

driver.getCar().setFreeCargoVolume(driver.getCar().getFreeCargoVolume(
)                    -                    orderPart.getCargoVolume());

driver.getCar().setFreeCargoWeight(driver.getCar().getFreeCargoWeight(
)                    -                    orderPart.getCargoWeight());
        driver.setAvailable(false);
        orderService.update(order);
        driverService.save(driver);

        OrderPart                   partWithoutDriver                   =
order.getFirstOrderPartWithoutDriver();

        if          (partWithoutDriver          !=          null)          {
            long  timeDiff  =  partWithoutDriver.getReadinessTime()  -
orderPart.getReadinessTime();
            long     timeOnWay     =     order.getDeliveryTime()     -
orderPart.getReadinessTime();
            if          (driver.getCar().getFreeCargoWeight()          >=
partWithoutDriver.getCargoWeight()
                    &&    driver.getCar().getFreeCargoVolume(    )    >=
partWithoutDriver.getCargoVolume()
                    &&    TimeUnit.MILLISECONDS.toMinutes(timeDiff)    >
TimeUnit.MILLISECONDS.toMinutes(timeOnWay)
            )                                                          {
                processOrder(order,     partWithoutDriver,     driver);
            }                              else                        {
                processOrder(order,      partWithoutDriver,      null);
            }
        }
    }

    private void subscribeForUpdates(Order order, OrderPart orderPart,
DriverResultListener                    callback)                    {
        DatabaseReference                  deliveries                  =
FirebaseDatabase.getInstance().getReference(FIREBASE_REFERENCE_DELIVER
IES);
        DatabaseReference                   orderRef                   =
deliveries.child(String.valueOf(order.getId())      +      "-"      +
orderPart.getId());

        orderRef.addListenerForSingleValueEvent(new
ValueEventListener()                                                   {
            @Override
            public  void  onDataChange(DataSnapshot  dataSnapshot)  {
                DriverResult                result                  =
```

```
dataSnapshot.getValue(DriverResult.class);
                if (result != null) {
                    callback.onResultObtained(result);
                } else {
                    System.err.println("DataSnapshot has null data
===>  " + dataSnapshot);
                }
            }

            @Override
            public void onCancelled(DatabaseError error) {
                System.err.println("Firebase DB Error occurred ===> "
+ error.getMessage());
            }
        });
    }

    private interface DriverResultListener {
        void onResultObtained(DriverResult result);
    }
}
@Component("greedyAlgorithm")
public class GreedyAlgorithm {

    private final int[] radiusZones = { 1_000, 3_000, 5_000, 10_000,
20_000 };

    private final DurationEvaluator durationEvaluator;
    private final OrderPriceEvaluator orderPriceEvaluator;

    @Autowired
    public GreedyAlgorithm(DurationEvaluator durationEvaluator,
OrderPriceEvaluator orderPriceEvaluator) {
        this.durationEvaluator = durationEvaluator;
        this.orderPriceEvaluator = orderPriceEvaluator;
    }

    Driver findPermanentDriver(OrderPart orderPart, List<Driver>
allDrivers) {
        return findAppropriateDriver(orderPart, allDrivers, driver ->
!driver.getIsFreelancer(), 0);
    }

    Driver findAnyDriver(OrderPart orderPart, List<Driver> allDrivers,
List<Long> ids) {
        return findAppropriateDriver(orderPart, allDrivers, driver ->
!ids.contains(driver.getId()), 0);
    }

    private Driver findAppropriateDriver(OrderPart orderPart,
List<Driver> allDrivers, Predicate<Driver> predicate, int position) {
        if (position >= radiusZones.length) {
            return null;
        }
```

```
        Location                        location                        =
orderPart.getTakeOutAddress().toLocation();

        Predicate<Driver>       complexPredicate       =       predicate
                .and(Driver::getIsAvailable)
                .and(driver  ->  driver.getCar().getFreeCargoVolume()  >
orderPart.getCargoVolume()  &&  driver.getCar().getFreeCargoWeight()  >
orderPart.getCargoWeight())
                .and(driver                                             ->
LocationUtils.isInsideCircle(radiusZones[position],          location,
driver.getCarLocation()))
                .and(driver                    ->                    {
                    long                   duration                   =
durationEvaluator.evaluate(location,         driver.getCarLocation());
                    return                  duration                  !=
DurationEvaluator.DURATION_NOT_CALCULATED                             &&
                            TimeUnit.SECONDS.toMinutes(duration)     <=
DurationEvaluator.DRIVER_NOTIFYING_TIME_MINUTES;
                });

        return                                      allDrivers.stream()
                .filter(complexPredicate)
                .min(getDriverComparator(location))
                .orElseGet(()    ->    findAppropriateDriver(orderPart,
allDrivers,         predicate,         position         +         1));
    }
    private Comparator<Driver> getDriverComparator(Location location)
{
        return         (driver1,         driver2)         ->         {
            double                        price1                        =
orderPriceEvaluator.evaluateForSingleDriver(
                    new                 LatLng(location.getLatitude(),
location.getLongitude()),
                    new          LatLng(driver1.getCar().getLatitude(),
driver1.getCar().getLongitude()),
                    driver1.getSalaryFactor()
            );

            double                        price2                        =
orderPriceEvaluator.evaluateForSingleDriver(
                    new                 LatLng(location.getLatitude(),
location.getLongitude()),
                    new          LatLng(driver2.getCar().getLatitude(),
driver2.getCar().getLongitude()),
                    driver2.getSalaryFactor()
            );
            return           Double.compare(price1,           price2);
        };
    }
}
```