ДОДАТОК А

ГРАФІЧНИЙ МАТЕРІАЛ

ГЮИК. 507600.001

(позначення документа)

ЗАТВЕРДЖЕНО
ГЮИК. 507600.001 – ЛУ(ПС)

Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів
Графічний матеріал

ГЮИК. 507600.001 – ЛУ(ПС)

ЛИСТІВ 5

2020 г.

Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
_____проф. Ситніков Д.Е.
(підпис)

Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів

Графічний матеріал

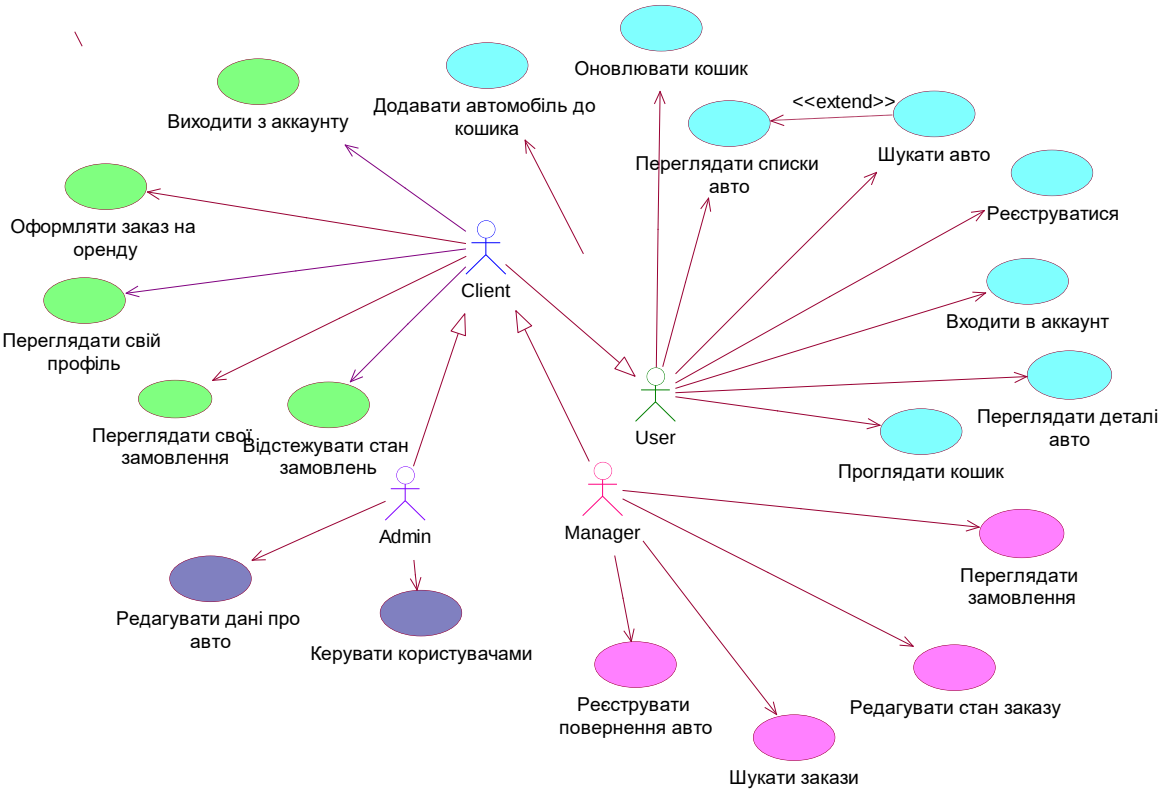
ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК. 507600.001 – ЛУ(ПС)

РОЗРОБИЛА:
ст.гр. ІТПм-19-1
Богун А. Є.

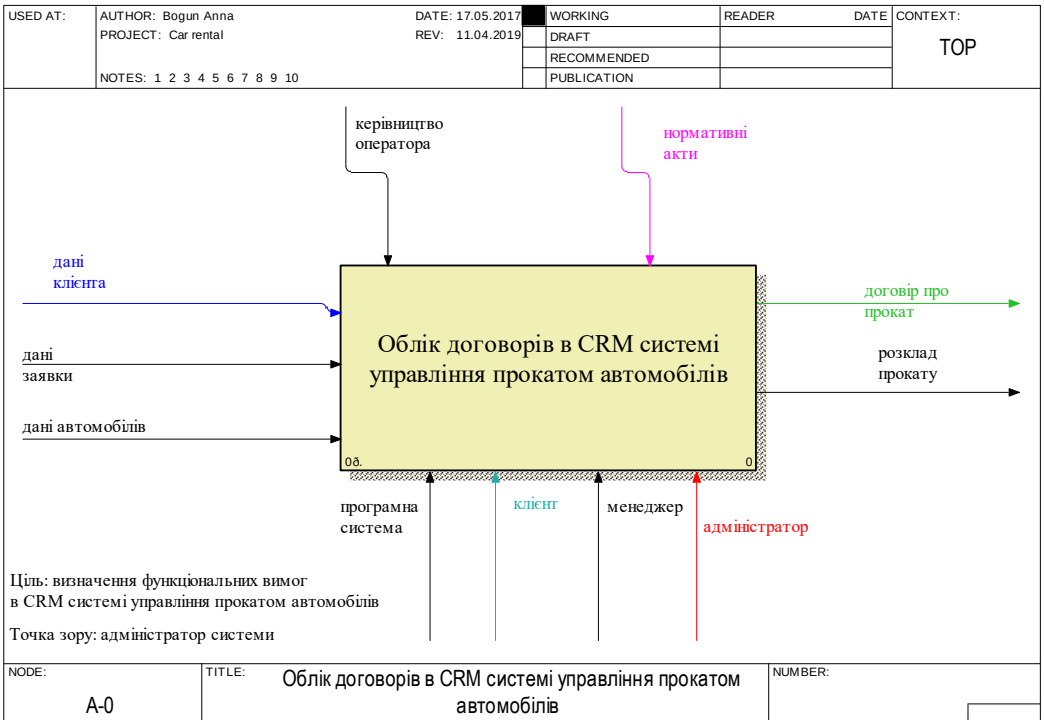
2020 г.

ДІАГРАМА ПРЕЦЕДЕНТІВ



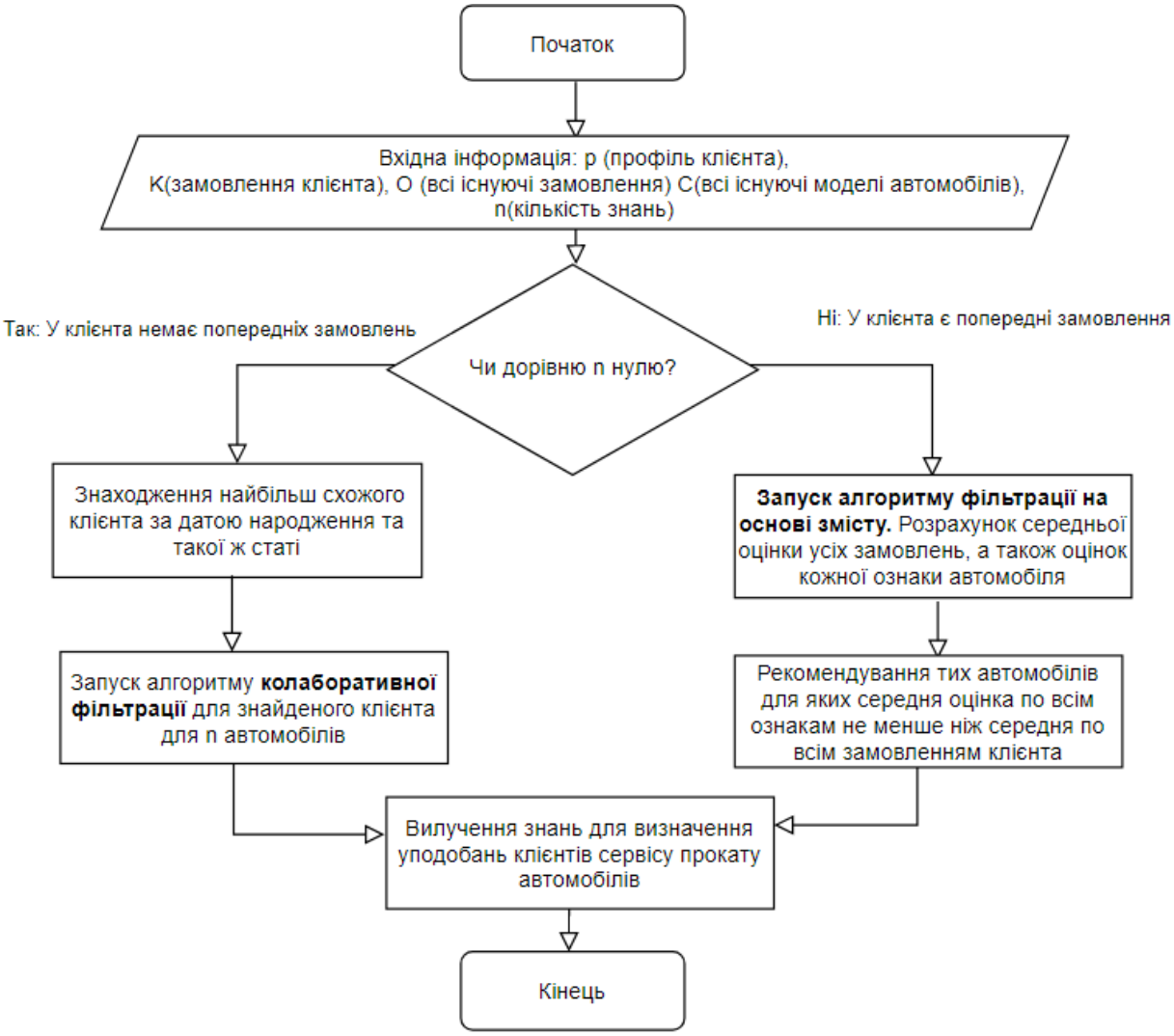
Розробила	*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*	
Перевірив	*Ситніков Д.Е.*				
				ІТПм-19-1	*Лист 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Листів 5*

КОНТЕКСНА ДІАГРАМА IDEF0



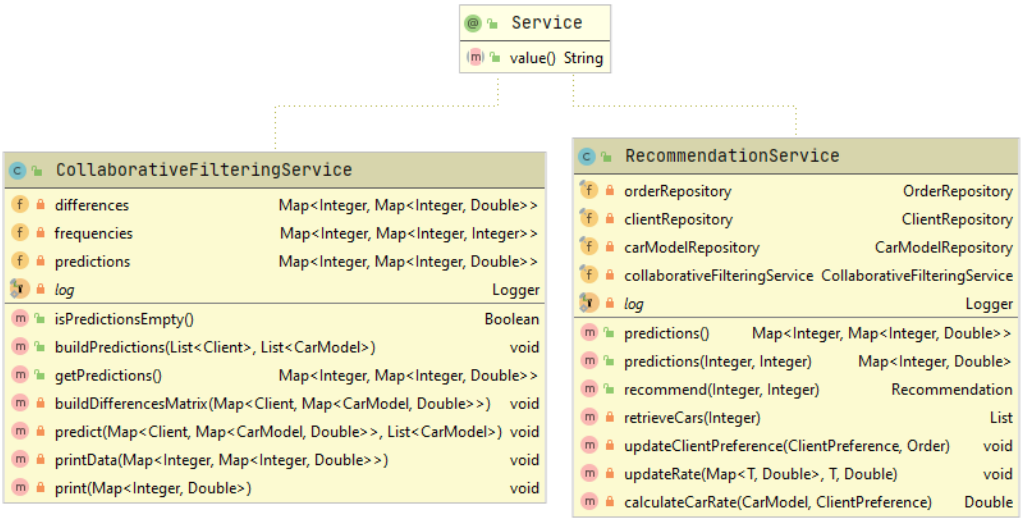
Розробила	*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*	
Перевірив	*Ситніков Д.Е.*				
				ІТПм-19-1	*Лист 2*
Затвердив	*Гребеннік І.В.*			*СТ*	*Листів 5*

СХЕМА АЛГОРИТМУ РОБОТИ МЕТОДУ ВИЛУЧЕННЯ ЗНАНЬ УПОДОБАНЬ КЛІЄНТІВ



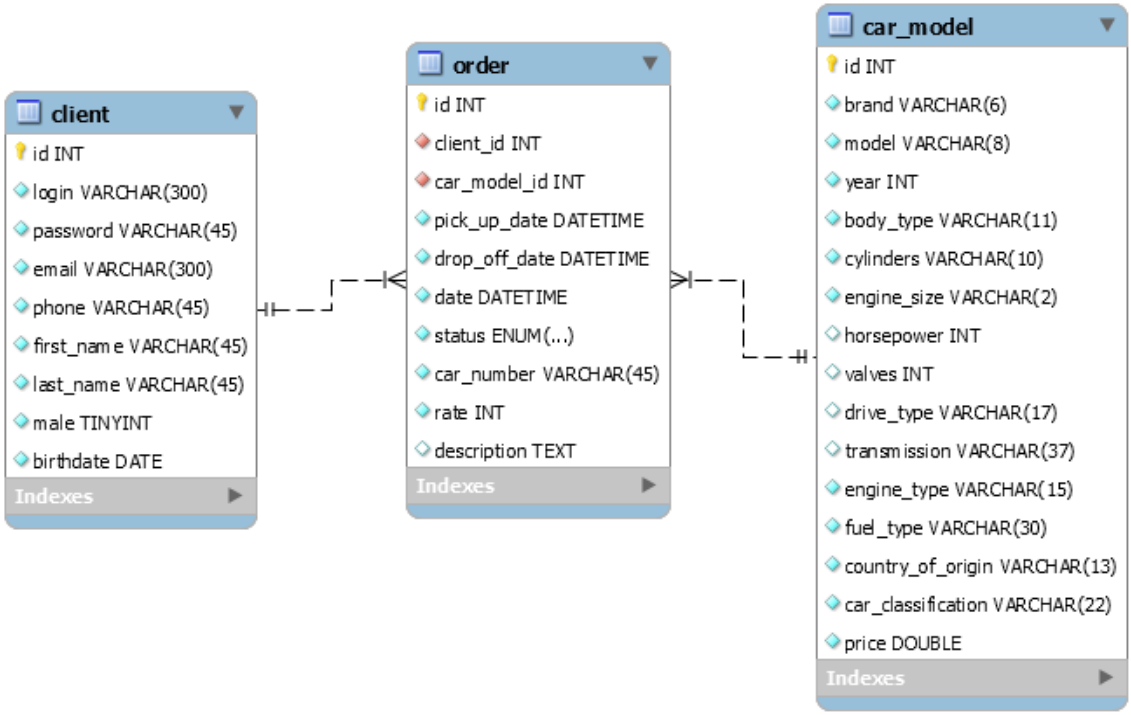
Розробила	*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*	
Перевірив	*Ситніков Д.Е.*				
				ІТПм-19-1	*Лист 3*
Затвердив	*Гребеннік І.В.*			*СТ*	*Листів 5*

ДІАГРАМА КЛАСІВ-СЕРВІСІВ



Розробила	*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*	
Перевірив	*Ситніков Д.Е.*				
				ІТПм-19-1	*Лист 4*
Затвердив	*Гребеннік І.В.*			*СТ*	*Листів 5*

МОДЕЛЬ ДАНИХ



Розробила	*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*	
Перевірив	*Ситніков Д.Е.*				
				ІТПм-19-1	*Лист 5*
Затвердив	*Гребеннік І.В.*			*СТ*	*Листів 5*

ДОДАТОК Б

ТЕКСТ ПРОГРАМИ

ГЮИК. 507600.001 – 01 12 01

(позначення документа)

ЗАТВЕРДЖЕНО
ГЮИК. 507600.001 ИЗ–ЛУ

Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів

Текст програми

ГЮИК. 507600.001 ИЗ–ЛУ

ЛИСТІВ 8

2020 г.

Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
_____проф. Ситніков Д.Е.
(підпис)

Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК. 507600.001 ИЗ–ЛУ

РОЗРОБИЛА:
ст.гр. ІТПм-19-1
Богун А.Є.

2020 г.

```
@Service
@Slf4j
public class CollaborativeFilteringService {
    private Map<Integer, Map<Integer, Double>> differences = new HashMap<>();
    private Map<Integer, Map<Integer, Integer>> frequencies = new HashMap<>();
    private Map<Integer, Map<Integer, Double>> predictions = new HashMap<>();

    public Boolean isPredictionsEmpty() {
        return predictions.isEmpty();
    }

    public void buildPredictions(List<Client> clients, List<CarModel> cars) {
        log.info("Collaborative Filtering - Before the Prediction");
        Map<Client, Map<CarModel, Double>> inputData = new HashMap<>();
        for (Client client : clients) {
            Map<CarModel, Double> orders = new HashMap<>();
            for (Order order : client.getOrders()) {
                orders.put(order.getCarModel(), order.getRate());
            }
            inputData.put(client, orders);
        }
        buildDifferencesMatrix(inputData);
        predict(inputData, cars);
    }

    public Map<Integer, Map<Integer, Double>> getPredictions() {
        return predictions;
    }

    /**
     * Based on the available data, calculate the relationships between the
     * cars and number of occurences
     *
     * @param data existing client data and their cars' ratings
     */
    private void buildDifferencesMatrix(Map<Client, Map<CarModel, Double>> data)
{

        for (Map<CarModel, Double> client : data.values()) {
            for (Entry<CarModel, Double> carRate : client.entrySet()) {
                if (!differences.containsKey(carRate.getKey().getId())) {
                    differences.put(carRate.getKey().getId(), new
HashMap<Integer, Double>());
                    frequencies.put(carRate.getKey().getId(), new
HashMap<Integer, Integer>());
                }
                for (Entry<CarModel, Double> secondCarRate : client.entrySet())
{
                    int oldCount = 0;
                    if
(frequencies.get(carRate.getKey().getId()).containsKey(secondCarRate.getKey().ge
tId())) {
                        oldCount =
frequencies.get(carRate.getKey().getId()).get(secondCarRate.getKey().getId()).in
tValue();
                    }
                    double oldDiff = 0.0;
                    if
(differences.get(carRate.getKey().getId()).containsKey(secondCarRate.getKey().ge
tId())) {
```

```
                        oldDiff =
differences.get(carRate.getKey().getId()).get(secondCarRate.getKey().getId()).do
ubleValue();
                    }
                    double observedDiff = carRate.getValue() -
secondCarRate.getValue();

frequencies.get(carRate.getKey().getId()).put(secondCarRate.getKey().getId(),
oldCount + 1);

differences.get(carRate.getKey().getId()).put(secondCarRate.getKey().getId(),
oldDiff + observedDiff);
                }
            }
        }
        for (Integer j : differences.keySet()) {
            for (Integer i : differences.get(j).keySet()) {
                double oldValue = differences.get(j).get(i).doubleValue();
                int count = frequencies.get(j).get(i).intValue();
                differences.get(j).put(i, oldValue / count);
            }
        }
    }

    /**
     * Based on existing data predict all missing ratings. If prediction is not
     * possible, the value will be equal to -1
     *
     * @param data existing client data and their cars' ratings
     */
    private void predict(Map<Client, Map<CarModel, Double>> data,
                         List<CarModel> cars) {
        Map<Integer, Double> uPred = new HashMap<Integer, Double>();
        Map<Integer, Integer> uFreq = new HashMap<Integer, Integer>();
        for (Integer j : differences.keySet()) {
            uFreq.put(j, 0);
            uPred.put(j, 0.0);
        }
        for (Entry<Client, Map<CarModel, Double>> e : data.entrySet()) {
            for (CarModel j : e.getValue().keySet()) {
                for (Integer k : differences.keySet()) {
                    try {
                        double predictedValue =
differences.get(k).get(j).doubleValue() + e.getValue().get(j).doubleValue();
                        double finalValue = predictedValue *
frequencies.get(k).get(j).intValue();
                        uPred.put(k, uPred.get(k) + finalValue);
                        uFreq.put(k, uFreq.get(k) +
frequencies.get(k).get(j).intValue());
                    } catch (NullPointerException e1) {
                    }
                }
            }
            Map<Integer, Double> clean = new HashMap<Integer, Double>();
            for (Integer j : uPred.keySet()) {
                if (uFreq.get(j) > 0) {
                    clean.put(j, uPred.get(j).doubleValue() /
uFreq.get(j).intValue());
                }
            }
            for (CarModel j : cars) {
```

```
                if (e.getValue().containsKey(j)) {
                    clean.put(j.getId(), e.getValue().get(j));
                } else if (!clean.containsKey(j)) {
                    clean.put(j.getId(), -1.0);
                }
            }
            predictions.put(e.getKey().getId(), clean);
        }
        printData(predictions);
    }

    private void printData(Map<Integer, Map<Integer, Double>> data) {
        for (Integer clientId : data.keySet()) {
            log.info(clientId + "(clientId):");
            print(data.get(clientId));
        }
    }

    private void print(Map<Integer, Double> hashMap) {
        NumberFormat formatter = new DecimalFormat("#0.000");
        for (Integer j : hashMap.keySet()) {
            log.info(" " + j + " --> " +
formatter.format(hashMap.get(j).doubleValue()));
        }
    }
}



@Service
@Slf4j
@RequiredArgsConstructor
public class RecommendationService {
    private final OrderRepository orderRepository;

    private final ClientRepository clientRepository;

    private final CarModelRepository carModelRepository;

    private final CollaborativeFilteringService collaborativeFilteringService;

    public Map<Integer, Map<Integer, Double>> predictions() {
        List<CarModel> cars =
IteratorUtils.toList(carModelRepository.findAll().iterator());
        List<Client> clients =
IteratorUtils.toList(clientRepository.findAll().iterator());
        if (collaborativeFilteringService.isPredictionsEmpty()) {
            collaborativeFilteringService.buildPredictions(clients, cars);
        }
        return collaborativeFilteringService.getPredictions();
    }

    public Map<Integer, Double> predictions(Integer clientId, Integer count) {
        Map<Integer, Map<Integer, Double>> predictions = predictions();
        return predictions.get(clientId)
                .entrySet()
                .stream()
                .limit(count)
                .collect(Collectors.toMap(
                        entry -> entry.getKey(),
                        entry -> entry.getValue()));
```

```
    }

    public Recommendation recommend(Integer clientId, Integer count) {
        Optional<Client> clientOptional = clientRepository.findById(clientId);
        if (!clientOptional.isPresent()) {
            throw new BadRequestAppException("Client with id " + clientId + "
was not found");
        }
        Client client = clientOptional.get();

        Recommendation recommendation = new Recommendation();
        recommendation.setClientId(client.getId());
        if (client.getOrders().isEmpty()) {
            log.info("No previous orders for client " + clientId + ". Prediction
is started");
            Client nearestClient =
clientRepository.findNearestClientByIdAndMale(clientId, client.getMale());
            predictions(nearestClient.getId(),
count).entrySet().forEach(prediction -> recommendation

.addRecommendation(carModelRepository.findById(prediction.getKey()).get(),
                            prediction.getValue() > 0 ? true : false));
            return recommendation;
        }
        Double averageClientRate =
client.getOrders().stream().mapToDouble(Order::getRate).average().getAsDouble();
        log.info("Average order rate for client " + clientId + " is " +
averageClientRate);
        ClientPreference clientPreference = new ClientPreference();

        List<CarModel> cars = retrieveCars(count);
        client.getOrders()
                .stream()
                .forEach(order -> updateClientPreference(clientPreference,
order));
        log.info("Car preference map for client" + clientId + " is " +
clientPreference);
        Map<Integer, Double> carRates = new TreeMap<>();
        cars.forEach(car -> carRates.put(car.getId(), calculateCarRate(car,
clientPreference)));

        carRates.entrySet().forEach(rate ->

recommendation.addRecommendation(carModelRepository.findById(rate.getKey()).get(
),
                        rate.getValue() > averageClientRate ? true : false));

        return recommendation;
    }

    private List retrieveCars(Integer count) {
        if (Objects.nonNull(count)) {
            return carModelRepository.findAll(PageRequest.of(1, count))
                    .get()
                    .collect(Collectors.toList());
        } else {
            return
IteratorUtils.toList(carModelRepository.findAll().iterator());
        }
    }
```

```
    private void updateClientPreference(ClientPreference clientPreference, Order
order) {
        CarModel car = order.getCarModel();
        Double rate = order.getRate();

        updateRate(clientPreference.getPriceRate(), car.getPrice(), rate);
        updateRate(clientPreference.getModelRate(), car.getModel(), rate);
        updateRate(clientPreference.getBrandRate(), car.getBrand(), rate);
        updateRate(clientPreference.getYearRate(), car.getYear(), rate);
        updateRate(clientPreference.getBodyTypeRate(), car.getBodyType(), rate);
        updateRate(clientPreference.getCylindersRate(), car.getCylinders(),
rate);
        updateRate(clientPreference.getEngineSizeRate(), car.getEngineSize(),
rate);
        updateRate(clientPreference.getHorsepowerRate(), car.getHorsepower(),
rate);
        updateRate(clientPreference.getValvesRate(), car.getValves(), rate);
        updateRate(clientPreference.getDriveTypeRate(), car.getDriveType(),
rate);
        updateRate(clientPreference.getTransmissionRate(),
car.getTransmission(), rate);
        updateRate(clientPreference.getEngineTypeRate(), car.getEngineSize(),
rate);
        updateRate(clientPreference.getFuelTypeRate(), car.getFuelType(), rate);
        updateRate(clientPreference.getCountryOfOriginRate(),
car.getCountryOfOrigin(), rate);
        updateRate(clientPreference.getCarClassificationRate(),
car.getCarClassification(), rate);
    }

    private <T> void updateRate(Map<T, Double> preferenceMap, T parameterValue,
Double rate) {
        if (Objects.nonNull(parameterValue)) {
            if (!preferenceMap.containsKey(parameterValue)) {
                preferenceMap.put(parameterValue, Double.valueOf(rate));
            } else {
                preferenceMap.put(parameterValue,
(preferenceMap.get(parameterValue) + rate) / 2);
            }
        }
    }

    private Double calculateCarRate(CarModel car, ClientPreference
clientPreference) {
        Double rate = 0.0;
        rate += clientPreference.getPriceRate().getOrDefault(car.getPrice(),
0.0);
        rate += clientPreference.getModelRate().getOrDefault(car.getModel(),
0.0);
        rate += clientPreference.getBrandRate().getOrDefault(car.getBrand(),
0.0);
        rate += clientPreference.getYearRate().getOrDefault(car.getYear(), 0.0);
        rate +=
clientPreference.getBodyTypeRate().getOrDefault(car.getBodyType(), 0.0);
        rate +=
clientPreference.getCylindersRate().getOrDefault(car.getCylinders(), 0.0);
        rate +=
clientPreference.getEngineSizeRate().getOrDefault(car.getEngineSize(), 0.0);
        if (Objects.nonNull(car.getHorsepower())) {
            rate +=
clientPreference.getHorsepowerRate().getOrDefault(car.getHorsepower(), 0.0);
```

```
        }
        if (Objects.nonNull(car.getValves())) {
            rate +=
clientPreference.getValvesRate().getOrDefault(car.getValves(), 0.0);
        }
        if (Objects.nonNull(car.getDriveType())) {
            rate +=
clientPreference.getDriveTypeRate().getOrDefault(car.getDriveType(), 0.0);
        }
        if (Objects.nonNull(car.getTransmission())) {
            rate +=
clientPreference.getTransmissionRate().getOrDefault(car.getTransmission(), 0.0);
        }
        rate +=
clientPreference.getEngineTypeRate().getOrDefault(car.getEngineType(), 0.0);
        rate +=
clientPreference.getFuelTypeRate().getOrDefault(car.getFuelType(), 0.0);
        rate +=
clientPreference.getCountryOfOriginRate().getOrDefault(car.getCountryOfOrigin(),
0.0);
        rate +=
clientPreference.getCarClassificationRate().getOrDefault(car.getCarClassificatio
n(), 0.0);
        return rate / 15;
    }
}



@Repository
public interface ClientRepository extends PagingAndSortingRepository<Client,
Integer> {
    @Query(value = "SELECT * FROM client WHERE id <> :id  AND male = :male " +
            "ORDER BY ABS( DATEDIFF( birthdate, (SELECT c.birthdate FROM client
c WHERE c.id = :id))) LIMIT 1",
            nativeQuery = true)
    Client findNearestClientByIdAndMale(@Param("id") Integer id, @Param("male")
Boolean male);
}



@RestController
@RequiredArgsConstructor
public class RecommendationController {

    private final RecommendationService recommendationService;

    @RequestMapping(value = "/recommend/{clientId}", method = RequestMethod.GET,
produces = MediaType.APPLICATION_JSON_VALUE)
    public Recommendation recommend(@PathVariable Integer clientId,
@RequestParam(required = false) Integer count) {
        return recommendationService.recommend(clientId, count);
    }

    @RequestMapping(value = "/predictions", method = RequestMethod.GET, produces
= MediaType.APPLICATION_JSON_VALUE)
    public Map<Integer, Map<Integer, Double>> predictions() {
        return recommendationService.predictions();
    }

    @RequestMapping(value = "/predictions/{clientId}/{count}", method =
RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
```

```
    public Map<Integer, Double> predictions(@PathVariable Integer clientId,
@PathVariable Integer count) {
        return recommendationService.predictions(clientId, count);
    }
}


@ComponentScan
@Configuration
@EnableAutoConfiguration
public class CarAnalyzerConfig {
    @Value("${spring.datasource.driver-class-name}")
    private String driverClassName;
    @Value("${spring.datasource.username}")
    private String datasourceUsername;
    @Value("${spring.datasource.password}")
    private String datasourcePassword;
    @Value("${spring.datasource.url}")
    private String datasourceUrl;

    @Bean
    public DataSource mysqlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(driverClassName);
        dataSource.setUrl(datasourceUrl);
        dataSource.setUsername(datasourceUsername);
        dataSource.setPassword(datasourcePassword);

        return dataSource;
    }

    @Bean
    public JdbcIO.DataSourceConfiguration beamDataSourceConfiguration() {
        return JdbcIO.DataSourceConfiguration.create(driverClassName,
datasourceUrl)
                .withUsername(datasourceUsername)
                .withPassword(datasourcePassword);
    }
}


package io.gatling.funspec.example

import io.gatling.core.Predef._
import io.gatling.http.Predef._

class GatlingRecommendApiTest extends Simulation {

  val httpConf = http.baseUrl("http://localhost:8083")
    .header("Accept", "application/json")
  val repeatCount = 10
  val random = scala.util.Random

  def reccomend() = {
    repeat(repeatCount) {
      exec(http("Recommend API test")
        .get("/recommend/" + random.nextInt(63)+ "?count=30")
        .check(status.is(200)))
    }
  }
```

```
  val scn = scenario("Recommend API test")
    .exec(reccomend())

  setUp(
    scn.inject(atOnceUsers(1))
  ).protocols(httpConf)

}
```

ДОДАТОК В

SQL-скрипт бази даних

ГЮИК. 507600.001 – 01 12 01

(позначення документа)

ЗАТВЕРДЖЕНО
ГЮИК. 507600.001ИЗ–ЛУ

Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів

SQL-скрипт бази даних

ГЮИК. 507600.001 ИЗ–ЛУ

ЛИСТІВ 2

2020 г.

Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
_____проф. Ситніков Д.Е.
(підпис)

Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів

SQL-скрипт бази даних

ГЮИК. 507600.001 ИЗ–ЛУ

РОЗРОБИЛА:
ст.гр. КН-15-1
Богун А.Є.

2020 г

```
-- MySQL Script generated by MySQL Workbench
-- Sun Dec  6 23:54:51 2020
-- Model: New Model    Version: 1.0
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema cars
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema cars
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `cars` DEFAULT CHARACTER SET utf8 ;
USE `cars` ;

-- -----------------------------------------------------
-- Table `cars`.`car_model`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `cars`.`car_model` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `brand` VARCHAR(6) NOT NULL,
  `model` VARCHAR(8) NOT NULL,
  `year` INT NOT NULL,
  `body_type` VARCHAR(11) NOT NULL,
  `cylinders` VARCHAR(10) NOT NULL,
  `engine_size` VARCHAR(2) NOT NULL,
  `horsepower` INT NULL DEFAULT NULL,
  `valves` INT NULL DEFAULT NULL,
  `drive_type` VARCHAR(17) NULL DEFAULT NULL,
  `transmission` VARCHAR(37) NULL DEFAULT NULL,
  `engine_type` VARCHAR(15) NOT NULL,
  `fuel_type` VARCHAR(30) NOT NULL,
  `country_of_origin` VARCHAR(13) NOT NULL,
  `car_classification` VARCHAR(22) NOT NULL,
  `price` DOUBLE NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 401828693
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `cars`.`client`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `cars`.`client` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `login` VARCHAR(300) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `email` VARCHAR(300) NOT NULL,
  `phone` VARCHAR(45) NOT NULL,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `male` TINYINT NOT NULL DEFAULT '1',
  `birthdate` DATE NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id UNIQUE` (`id` ASC) VISIBLE,
```

```
  UNIQUE INDEX `login_UNIQUE` (`login` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 292
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `cars`.`order`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `cars`.`order` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `client_id` INT UNSIGNED NOT NULL,
  `car_model_id` INT NOT NULL,
  `pick_up_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `drop_off_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `status` ENUM('newed', 'inprogress', 'completed', 'rejected') NOT NULL DEFAULT
'completed',
  `car_number` VARCHAR(45) NOT NULL,
  `rate` INT NOT NULL,
  `description` TEXT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
  INDEX `fk_order_user1_idx` (`client_id` ASC) VISIBLE,
  INDEX `fk_car_model_id_idx` (`car_model_id` ASC) VISIBLE,
  CONSTRAINT `fk_car_model`
    FOREIGN KEY (`car_model_id`)
    REFERENCES `cars`.`car_model` (`id`)
    ON UPDATE CASCADE,
  CONSTRAINT `fk_client`
    FOREIGN KEY (`client_id`)
    REFERENCES `cars`.`client` (`id`)
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 2342
DEFAULT CHARACTER SET = utf8;

USE `cars` ;

-- -----------------------------------------------------
-- procedure add_brands
-- -----------------------------------------------------

DELIMITER $$
USE `cars`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_brands`(IN differ INT, IN
amount INT)
BEGIN
DECLARE count int default 0;
WHILE count < amount DO
INSERT INTO `brand`
 (id, `name`, `image`, `description`)
 VALUES (DEFAULT, CONCAT('name #', count+differ),
CONCAT('image #', count+differ), CONCAT('description #', count+differ));
SET count = count + 1;
END WHILE;
END$$

DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

ДОДАТОК Г

ВІДОМІСТЬ АТЕСТАЦІЙНОЇ РОБОТИ

ГЮИК. 507600.001

(позначення документа)

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	**ГЮИК.507600.001ПЗ**	Пояснювальна записка	107 с.
2	**ГЮИК. 507600.001-01 12 01**	Текст програми	8 с., вкл. в ПЗ
3	**ГЮИК. 507600.001-01 12 01**	SQL-скрипт бази даних	2 с., вкл. в ПЗ
		Графічні документи	
5		Діаграма прецедентів	Включено в ПЗ
6		Контекстна діаграма	Включено в ПЗ
		IDEF0	
7		Контекстна діаграма	Включено в ПЗ
8		Схема алгоритму роботи	Включено в ПЗ
		методу вилучення знань	
		Уподобань клієнтів	
9		Діаграма класів-сервісів	Включено в ПЗ
10		Модель даних	Включено в ПЗ

					ГЮИК. 507600.001Д3	
Зм.	*Лист*	*№ докум*	*Підпис*	*Дата*		
Розробила		*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*	Лист 1 / Листів 1
Перевірив		*Ситніков Д.Е.*				
Н. Контр.		*Ситніков Д.Е.*				*ХНУРЕ Кафедра СТ*
Затвердив		*Гребеннік І.В.*				

ДОДАТОК Д

СПЕЦИФІКАЦІЯ

ГЮИК. 507600.001

(позначення документа)

Код	Назва	Примітка
	Документація	
ГЮИК. 507600.001 -01 12 01 М	«Текст програми»	Додаток Б
ГЮИК. 507600.001 -01 12 01 М	«SQL-скрипт бази даних»	Додаток В

					ГЮИК. 507600.001Д3			
Зм.	*Лист*	*№ докум*	*Підпис*	*Дата*				
Розробила		*Богун А.Є.*			*Розробка методу вилучення знань для визначення уподобань клієнтів сервісу оренди автомобілів*		Лист	Листів
Перевірив		*Ситніков Д.Е.*					*1*	*1*
Н. Контр.		*Ситніков Д.Е.*				*ХНУРЕ Кафедра СТ*		