ДОДАТОК А

Текст програми

Класс WebSecurityConfiguration

```
package com.truck.truckcontent.configuration;

import com.truck.truckcontent.service.TruckDriverDetailsService;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builde
rs.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecu
rity;
import
org.springframework.security.config.annotation.web.builders.WebSecur
ity;
import
org.springframework.security.config.annotation.web.configuration.Ena
bleWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.Web
SecurityConfigurerAdapter;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
public class WebSecurityConfiguration extends
WebSecurityConfigurerAdapter {

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Autowired
    private TruckDriverDetailsService truckDriverDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
throws Exception {
```

```
auth.userDetailsService(truckDriverDetailsService).passwordEncoder(p
asswordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.
                authorizeRequests()

.requestMatchers(PathRequest.toStaticResources().atCommonLocations()
).permitAll()
                .antMatchers("/").permitAll()
                .antMatchers("/*.js").permitAll()
                .antMatchers("/*.css").permitAll()
                .antMatchers("/login").permitAll()
                .antMatchers("/registration").permitAll()
                .antMatchers("/createOrder").permitAll()

.antMatchers("/truck_driver_orders").hasAuthority("DRIVER")

.antMatchers("/truck_driver_orders_history").hasAuthority("DRIVER")
                .antMatchers("/admin_trucks").hasAuthority("ADMIN")

.antMatchers("/admin_truck_drivers").hasAuthority("ADMIN")

.antMatchers("/admin_truck_orders").hasAnyAuthority("ADMIN", "DISP")

.antMatchers("/admin_truck_statistics").hasAuthority("ADMIN").anyReq
uest()
                .authenticated().and().csrf().disable().formLogin()
                .loginPage("/login").failureUrl("/login?error=true")
                .defaultSuccessUrl("/", true)
                .usernameParameter("email")
                .passwordParameter("password")
                .and().logout()
                .logoutRequestMatcher(new
AntPathRequestMatcher("/logout"))

.logoutSuccessUrl("/login").and().exceptionHandling()
                .accessDeniedPage("/access-denied");
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web
                .ignoring()
                .antMatchers("/resources/**", "/static/**");
    }
}
```

Класс LoginController

```
package com.truck.truckcontent.controller.trucks;

import com.truck.truckcontent.model.trucks.TruckDriver;
import com.truck.truckcontent.service.TruckDriverService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.validation.Valid;

@Controller
public class LoginController {

    @Autowired
    private TruckDriverService truckDriverService;

    @RequestMapping(value={"/login"}, method = RequestMethod.GET)
    public ModelAndView login(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("login");
        return modelAndView;
    }


    @RequestMapping(value="/registration", method =
RequestMethod.GET)
    public ModelAndView registration(){
        ModelAndView modelAndView = new ModelAndView();
        TruckDriver truckDriver = new TruckDriver();
        modelAndView.addObject("truckDriver", truckDriver);
        modelAndView.setViewName("registration");
        return modelAndView;
    }

    @RequestMapping(value = "/registration_driver", method =
RequestMethod.POST)
    public ModelAndView createNewUser(@Valid TruckDriver
truckDriver, BindingResult bindingResult) {
        ModelAndView modelAndView = new ModelAndView();
        TruckDriver truckDriverExists =
truckDriverService.findTruckDriverByEmail(truckDriver.getEmail());
        if (truckDriverExists != null) {
            bindingResult
                    .rejectValue("email", "error.truckDriver",
```

```
                            "Водій з поштою " +
truckDriver.getEmail() + " вже зареєстрований в системі");
        }
        if (bindingResult.hasErrors()) {
            modelAndView.setViewName("trucks/create_truck_driver");
        } else {
            truckDriverService.saveUser(truckDriver);
            modelAndView.addObject("successMessage", "Водія було
зареєстровано в системі: " + truckDriver.getFirstName() + " " +
truckDriver.getSecondName());
            modelAndView.addObject("truckDriver", new
TruckDriver());
            modelAndView.setViewName("trucks/create_truck_driver");
        }
        return modelAndView;
    }
}
```

Клас TruckController

```
package com.truck.truckcontent.controller.trucks;

import com.truck.truckcontent.dao.trucks.TruckDriverRepository;
import com.truck.truckcontent.dao.trucks.TruckRepository;
import com.truck.truckcontent.model.trucks.Truck;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class TruckController {

    @Autowired
    private TruckRepository truckRepository;

    @Autowired
    private TruckDriverRepository truckDriverRepository;

    @GetMapping("/createTruck")
    public String createTruck() {
        return "trucks/create_truck";
    }

    @PostMapping(path = "/createTruck")
    public @ResponseBody
    String createTruck(@RequestParam String model,
                                    @RequestParam float length,
                                    @RequestParam float height,
```

```
                                    @RequestParam float volume,
                                    @RequestParam float capacity,
                                    @RequestParam String number) {
        if (truckRepository.findByNumber(number) != null) {
            return "Авто з номером " + number + " вже зареєстровано в
системі";
        }

        Truck truck = new Truck();
        truck.setModel(model);
        truck.setLength(length);
        truck.setHeight(height);
        truck.setVolume(volume);
        truck.setCapacity(capacity);
        truck.setNumber(number);
        truckRepository.save(truck);
        return "Авто було зареєстровано в системі: " +
truck.getModel() + ", Номер: " + truck.getNumber();
    }

    @PostMapping(path = "/deleteTruck")
    public @ResponseBody
    String deleteTruckDriver(@RequestParam String id) {
        Truck truck =
truckRepository.findById(Integer.valueOf(id)).orElse(null);
        if (truck != null && truck.getTruckDriver() != null) {

truckDriverRepository.findById(truck.getTruckDriver().getId()).ifPre
sent(driver -> {
                driver.setTruck(null);
                truckDriverRepository.save(driver);
            });

        }

        truckRepository.delete(truck);
        return "deleteTruck";
    }
}
```

Клас TruckDriverController

```
package com.truck.truckcontent.controller.trucks;

import com.truck.truckcontent.dao.trucks.TruckDriverRepository;
import com.truck.truckcontent.dao.trucks.TruckOrderRepository;
import com.truck.truckcontent.dao.trucks.TruckRepository;
import com.truck.truckcontent.model.trucks.Truck;
import com.truck.truckcontent.model.trucks.TruckDriver;
import com.truck.truckcontent.model.trucks.TruckOrder;
import org.apache.tomcat.util.codec.binary.Base64;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.ModelAndView;

import java.io.IOException;
import java.util.List;

@Controller
public class TruckDriverController {

    @Autowired
    private TruckDriverRepository truckDriverRepository;

    @Autowired
    private TruckRepository truckRepository;

    @Autowired
    private TruckOrderRepository truckOrderRepository;

    @RequestMapping(value="/createTruckDriver",       method       =
RequestMethod.GET)
    public ModelAndView registration(){
        ModelAndView modelAndView = new ModelAndView();
        TruckDriver truckDriver = new TruckDriver();
        modelAndView.addObject("truckDriver", truckDriver);
        modelAndView.setViewName("trucks/create_truck_driver");
        return modelAndView;
    }

    @PostMapping(path = "/createTruckDriver")
    public @ResponseBody
    String createTruckDriver(@RequestParam String firstName,
                                    @RequestParam String secondName,
                                    @RequestParam     MultipartFile
image,
                                    @RequestParam   String   phone)
throws IOException {
        if (truckDriverRepository.findByPhone(phone) != null) {
            return "Водій з номером " + phone + " вже зареєстрований
в системі";
        }

        TruckDriver truckDriver = new TruckDriver();
        truckDriver.setFirstName(firstName);
        truckDriver.setSecondName(secondName);
```

```
        truckDriver.setPhone(phone);

truckDriver.setImage(Base64.encodeBase64String(image.getBytes()));
        truckDriverRepository.save(truckDriver);
        return "Водія було зареєстровано в системі: " +
truckDriver.getFirstName() + " " + truckDriver.getSecondName();
    }

    @PostMapping(path = "/assign_auto")
    public @ResponseBody
    String assignAuto(@RequestParam String driverId,
                      @RequestParam String truckId) {
        TruckDriver                    driver                    =
truckDriverRepository.findById(Integer.valueOf(driverId)).orElse(nul
l);
        Truck                        truck                        =
truckRepository.findById(Integer.valueOf(truckId)).orElse(null);

        if (driver != null && truck != null) {
            Truck oldTruck = driver.getTruck();
            if (oldTruck != null) {
                oldTruck.setTruckDriver(null);
                truckRepository.save(oldTruck);
            }

            truck.setTruckDriver(driver);
            truckRepository.save(truck);
            return truck.getModel() + " " + truck.getNumber() + "
призачено   водію:   "   +   driver.getFirstName()   +   "   "   +
driver.getSecondName();
        }

        return "reject_auto";
    }

    @PostMapping(path = "/deleteTruckDriver")
    public @ResponseBody
    String deleteTruckDriver(@RequestParam String id) {
        TruckDriver                    driver                    =
truckDriverRepository.findById(Integer.valueOf(id)).orElse(null);
        if (driver != null && driver.getTruck() != null) {

truckRepository.findById(driver.getTruck().getId()).ifPresent(truck
-> {
                truck.setTruckDriver(null);
                truckRepository.save(truck);
            });

            List<TruckOrder>             driversOrders             =
truckOrderRepository.findAllByTruckDriver(driver);
```

```
            driversOrders.forEach(order                                    ->
order.setTruckDriver(null));
            truckOrderRepository.saveAll(driversOrders);
        }

        truckDriverRepository.delete(driver);
        return "deleteTruckDriver";
    }
}
```

Клас TruckOrderController

```
package com.truck.truckcontent.controller.trucks;

import com.truck.truckcontent.dao.trucks.TruckClientRepository;
import com.truck.truckcontent.dao.trucks.TruckDriverRepository;
import com.truck.truckcontent.dao.trucks.TruckOrderRepository;
import com.truck.truckcontent.model.trucks.TruckClient;
import com.truck.truckcontent.model.trucks.TruckDriver;
import com.truck.truckcontent.model.trucks.TruckOrder;
import com.truck.truckcontent.type.TruckOrderStatus;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Controller
public class TruckOrderController {

    @Autowired
    private TruckClientRepository truckClientRepository;

    @Autowired
    private TruckOrderRepository truckOrderRepository;

    @Autowired
    private TruckDriverRepository truckDriverRepository;

    @GetMapping("/createOrder")
```

```
    public String createOrder() {
        return "index";
    }

    @PostMapping(path = "/createOrder")
    public @ResponseBody
    String createOrder(@RequestParam String phone,
                                    @RequestParam String fromPoint,
                                    @RequestParam String toPoint,
                                    @RequestParam float distance,
                                    @RequestParam float price,
                                    @RequestParam String
additionalInfo) {
        TruckClient client = new TruckClient();
        client.setPhone(phone);
        truckClientRepository.save(client);

        TruckOrder truckOrder = new TruckOrder();
        truckOrder.setFromPoint(fromPoint);
        truckOrder.setToPoint(toPoint);
        truckOrder.setDistance(distance);
        truckOrder.setPrice(price);
        truckOrder.setCreateDate(LocalDate.now());
        truckOrder.setAdditionalInfo(additionalInfo);
        truckOrder.setTruckClient(client);
        truckOrder.setTruckOrderStatus(TruckOrderStatus.PENDING);
        truckOrderRepository.save(truckOrder);
        return "Дякуємо! Диспетчер зв'яжеться з Вами для уточнення
деталей замовлення";
    }

    @PostMapping(path = "/assignDriverForOrder")
    public @ResponseBody
    String assignDriverForOrder(@RequestParam String orderId,
                                @RequestParam String driverId) {
        TruckOrder order =
truckOrderRepository.findById(Integer.valueOf(orderId)).orElse(null)
;
        TruckDriver truckDriver =
truckDriverRepository.findById(Integer.valueOf(driverId)).orElse(nul
l);

        if (order != null && truckDriver != null) {
            order.setTruckDriver(truckDriver);
            order.setTruckOrderStatus(TruckOrderStatus.PROCESSING);
            truckOrderRepository.save(order);
            return "Замовлення №" + order.getId() + " призачено
водію: " + truckDriver.getFirstName() + " " +
truckDriver.getSecondName();
        }
```

```
        return "reject_driver_assign";
    }

    @PostMapping(path = "/rejectOrder")
    public @ResponseBody
    String rejectOrder(@RequestParam String id) {

truckOrderRepository.findById(Integer.valueOf(id)).ifPresent(order -
> {
            int durationMinutes = (int)
Duration.between(order.getCreateTime(),
LocalDateTime.now()).toMinutes();
            order.setCompletionTime(durationMinutes);
            order.setTruckOrderStatus(TruckOrderStatus.REJECTED);
            truckOrderRepository.save(order);
        });

        return "reject_order";
    }

    @PostMapping(path = "/completeOrder")
    public @ResponseBody
    String completeOrder(@RequestParam String id) {

truckOrderRepository.findById(Integer.valueOf(id)).ifPresent(order -
> {
            int durationMinutes = (int)
Duration.between(order.getCreateTime(),
LocalDateTime.now()).toMinutes();
            order.setCompletionTime(durationMinutes);
            order.setTruckOrderStatus(TruckOrderStatus.COMPLETED);
            truckOrderRepository.save(order);
        });

        return "complete_order";
    }

    @GetMapping("/truck_driver_orders")
    public String getTruckDrivers(Model model) {
        UserDetails userDetails = (UserDetails)
SecurityContextHolder.getContext().getAuthentication().getPrincipal(
);
        TruckDriver truckDriver =
truckDriverRepository.findByEmail(userDetails.getUsername());
        List<TruckOrder> orders = new ArrayList<>();
        truckOrderRepository.findAllByTruckDriver(truckDriver)
                .stream()
                .filter(order ->
order.getTruckOrderStatus().equals(TruckOrderStatus.PROCESSING))
                .forEach(orders::add);
        model.addAttribute("driverOrders", orders);
```

```
        return "trucks/truck_driver_orders";
    }

    @GetMapping("/truck_driver_orders_history")
    public String getTruckDriversHistory(Model model) {
        UserDetails userDetails = (UserDetails)
SecurityContextHolder.getContext().getAuthentication().getPrincipal(
);
        TruckDriver truckDriver =
truckDriverRepository.findByEmail(userDetails.getUsername());
        List<TruckOrder> orders = new ArrayList<>();
        truckOrderRepository.findAllByTruckDriver(truckDriver)
                .stream()
                .filter(order ->
!order.getTruckOrderStatus().equals(TruckOrderStatus.PROCESSING))
                .forEach(orders::add);
        model.addAttribute("driverOrders", orders);
        return "trucks/truck_driver_orders";
    }
}
```

Клас AdminController

```
package com.truck.truckcontent.controller;

import com.truck.truckcontent.dao.truck.DriverRepository;
import com.truck.truckcontent.dao.trucks.TruckDriverRepository;
import com.truck.truckcontent.dao.trucks.TruckOrderRepository;
import com.truck.truckcontent.dao.trucks.TruckRepository;
import com.truck.truckcontent.model.trucks.OrderCostsStatistics;
import com.truck.truckcontent.model.trucks.OrderCountStatistics;
import com.truck.truckcontent.model.trucks.OrderLoadStatistics;
import com.truck.truckcontent.model.trucks.OrderStatusStatistics;
import com.truck.truckcontent.model.trucks.TruckDriver;
import com.truck.truckcontent.model.trucks.TruckOrder;
import com.truck.truckcontent.type.TruckOrderStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
```

```
import java.util.Map;
import java.util.TreeMap;
import java.util.stream.IntStream;

@Controller
public class AdminController {

    private static final List<String> DEFAULT_HOURS = Arrays.asList(
            "06:00 - 07:00",
            "07:00 - 08:00",
            "08:00 - 09:00",
            "09:00 - 10:00",
            "10:00 - 11:00",
            "11:00 - 12:00",
            "12:00 - 13:00",
            "12:00 - 13:00",
            "13:00 - 14:00",
            "14:00 - 15:00",
            "15:00 - 16:00",
            "16:00 - 17:00",
            "18:00 - 19:00",
            "19:00 - 20:00",
            "20:00 - 21:00",
            "21:00 - 22:00"
            );
    private static final int STATISTICS_DAYS_PERIOD = 15;
    private static final int DEFAULT_WORKING_HOURS = 120 * 60;

    @Autowired
    private DriverRepository driverRepository;

    @Autowired
    private TruckDriverRepository truckDriverRepository;

    @Autowired
    private TruckOrderRepository truckOrderRepository;

    @Autowired
    private TruckRepository truckRepository;

    @GetMapping("/admin_truck_statistics")
    public String getStatistic(Model model) {
        List<TruckDriver> truckDrivers = new ArrayList<>();
        truckDriverRepository.findAll().forEach(truckDriver -> {
            if (truckDriver.getRoles().stream().anyMatch(role ->
role.getRole().equals("DRIVER"))) {
                truckDrivers.add(truckDriver);
            }
        });
        model.addAttribute("countTrucks", truckRepository.count());
```

```
        model.addAttribute("countOrders",
truckOrderRepository.count());
        model.addAttribute("countTruckDrivers",
truckDrivers.size());
        model.addAttribute("readyResources",
truckDriverRepository.findByTruckIsNotNull().size());
        model.addAttribute("listCostsStatistics",
getOrderCostsStatistics());
        model.addAttribute("intensity", String.format("%.3f",
getIntensity()));
        model.addAttribute("timeOfService", String.format("%.1f",
truckOrderRepository.countAverageServiceTime() / 60.0));
        model.addAttribute("possibilityEmpty",
getPossibilityOfEmptyQueue((float) (getIntensity() *
truckOrderRepository.countAverageServiceTime() / 60.0),

truckDriverRepository.findByTruckIsNotNull().size()));
        long percentOfDeny =
truckOrderRepository.collectTruckOrderStatusStatistics(Collections.s
ingletonList(TruckOrderStatus.REJECTED)).get(0).getCount() * 100 /
(truckOrderRepository.collectTruckOrderStatusStatistics(Collections.
singletonList(TruckOrderStatus.REJECTED)).get(0).getCount() +
truckOrderRepository.collectTruckOrderStatusStatistics(Collections.s
ingletonList(TruckOrderStatus.COMPLETED)).get(0).getCount());
        model.addAttribute("possibilityOfDeny", percentOfDeny);
        model.addAttribute("possibilityOfCompletion", 100 -
percentOfDeny);
        model.addAttribute("averageChannelsInAction",
String.format("%.3f",getIntensity() * (1 - (percentOfDeny /
100.0))));
        model.addAttribute("bandWidth", String.format("%.3f",
(truckOrderRepository.countAverageServiceTime() / 60.0) *
getIntensity() * (1 - (percentOfDeny / 100.0))));
        return "trucks/admin_truck_statistics";
    }

    @GetMapping("/admin_truck_drivers")
    public String getTruckOrders(Model model) {
        List<TruckDriver> truckDrivers = new ArrayList<>();
        truckDriverRepository.findAll().forEach(truckDriver -> {
            if (truckDriver.getRoles().stream().anyMatch(role ->
role.getRole().equals("DRIVER"))) {
                truckDrivers.add(truckDriver);
            }
        });
        model.addAttribute("truckDrivers", truckDrivers);
        model.addAttribute("trucks",
truckRepository.findByTruckDriverIsNull());
        return "trucks/admin_truck_drivers";
    }
```

```
    @GetMapping("/admin_truck_orders")
    public String getTruckDrivers(Model model) {
        model.addAttribute("truckOrders",
truckOrderRepository.findAll());
        truckRepository.findByTruckDriverIsNotNull().forEach(truck -
> {
            List<TruckOrder> processingTrucksOrders = new
ArrayList<>();
            truck.getTruckDriver().getTruckOrders().stream()
                    .filter(order ->
order.getTruckOrderStatus().equals(TruckOrderStatus.PROCESSING))
                    .forEach(processingTrucksOrders::add);

truck.getTruckDriver().setTruckOrders(processingTrucksOrders);
        });
        model.addAttribute("trucks",
truckRepository.findByTruckDriverIsNotNull());
        return "trucks/admin_truck_orders";
    }

    @GetMapping("/admin_trucks")
    public String getTrucks(Model model) {
        model.addAttribute("trucks", truckRepository.findAll());
        model.addAttribute("truckDrivers",
truckDriverRepository.findAll());
        return "trucks/admin_trucks";
    }

    @GetMapping("/getStatisticsForMonth")
    public @ResponseBody
    List<OrderCountStatistics> getStatisticsForMonth() {
        List<OrderCountStatistics> result = new ArrayList<>();
        List<OrderCountStatistics> orderCountStatistics =
truckOrderRepository.collectOrderCountStatisticsForLastMonth();
        for (int i = 0; i <= 15; i++) {
            LocalDate date = LocalDate.now().minusDays(i);
            orderCountStatistics.stream()
                    .filter(orderCountStatisticsEntry ->
orderCountStatisticsEntry.getCreateDate().equals(date))
                    .forEach(result::add);
            if
(orderCountStatistics.stream().noneMatch(orderStatistics ->
orderStatistics.getCreateDate().equals(date))) {
                result.add(0, new OrderCountStatistics(date, 0));
            }
        }
        Collections.sort(result);
        return result;
    }

    @GetMapping("/getStatisticsForHoursPercent")
```

```
    public @ResponseBody
    Map<String, Integer> getStatisticsForHoursPercent() {
        Map<String, Integer> hoursPercentageMap = new TreeMap<>();
        DEFAULT_HOURS.forEach(defaultHour ->
hoursPercentageMap.put(defaultHour, 0));
        List<TruckOrder> orderCountStatistics =
truckOrderRepository.findAllByCreateTimeBetween(LocalDateTime.now().
minusDays(STATISTICS_DAYS_PERIOD), LocalDateTime.now());
        orderCountStatistics.forEach(order -> {
            int orderHour = order.getCreateTime().getHour();
            String hours = orderHour + ":00 - " + (orderHour + 1) +
":00";
            hoursPercentageMap.put(hours,
hoursPercentageMap.getOrDefault(hours,0) + 1);
        });
        hoursPercentageMap.entrySet().forEach(entry ->
entry.setValue((int) Math.round((double) entry.getValue() * 100 /
orderCountStatistics.size())));

        return hoursPercentageMap;
    }

    @GetMapping("/getStatisticsForDaysPercent")
    public @ResponseBody
    List<OrderLoadStatistics> getStatisticsForDaysPercent() {
        //need to create additional list to exclude old entries
        List<OrderLoadStatistics> resultLoadStatistics = new
ArrayList<>();
        List<OrderLoadStatistics> orderLoadStatistics =
truckOrderRepository.collectLoadStatisticsForLastMonth();
        orderLoadStatistics.forEach(statistics ->
statistics.setLoadInMinutes((int)
Math.round((double)statistics.getLoadInMinutes() * 100 /
DEFAULT_WORKING_HOURS)));
        for (int i = 0; i <= 15; i++) {
            LocalDate date = LocalDate.now().minusDays(i);
            resultLoadStatistics.add(new OrderLoadStatistics(date,
0));
        }
        for (OrderLoadStatistics orderLoadStatistic :
orderLoadStatistics) {
            resultLoadStatistics.stream()
                    .filter(orderStatistics ->
orderStatistics.getCreateDate().equals(orderLoadStatistic.getCreateD
ate()))
                    .forEach(resultOrderStatistic ->
resultOrderStatistic.setLoadInMinutes(orderLoadStatistic.getLoadInMi
nutes()));
        }
        Collections.sort(resultLoadStatistics);
        return resultLoadStatistics;
```

```
    }

    @GetMapping("/getStatusStatisticsPercent")
    public @ResponseBody
    List<OrderStatusStatistics> getStatusStatisticsPercent() {
        return
truckOrderRepository.collectTruckOrderStatusStatistics(Arrays.asList
(TruckOrderStatus.REJECTED, TruckOrderStatus.COMPLETED));
    }

    @GetMapping("/getStatisticsIntensity")
    public @ResponseBody
    float getStatisticsIntensity() {
        return getIntensity();
    }

    @GetMapping("/getStatisticsMinResources")
    public @ResponseBody
    double getStatisticsMinResources() {
        float intensity = getIntensity();
        float avgServiceTimeHours = (float)
truckOrderRepository.countAverageServiceTime() / 60;
        return Math.ceil(intensity * avgServiceTimeHours);
    }

    private List<OrderCostsStatistics> getOrderCostsStatistics() {
        float intesivity = getIntensity();
        float avgServiceTime = (float)
truckOrderRepository.countAverageServiceTime() / 60; //here we get
hours

        float minResources = intesivity * avgServiceTime;
        int minResourcesCeiled = (int) Math.ceil(minResources);

        List<OrderCostsStatistics> orderCostsStatistics = new
ArrayList<>();
        for (int i = minResourcesCeiled; i < minResourcesCeiled + 6;
i++) {
            BigDecimal possibilityOfEmptyQueue =
getPossibilityOfEmptyQueue(minResources, i);
            BigDecimal countOfClientsInQueue =
getCountOfClientsInQueue(minResources, i);
            BigDecimal costs = getCosts(minResources, i);
            orderCostsStatistics.add(new OrderCostsStatistics(i,
possibilityOfEmptyQueue, countOfClientsInQueue, costs));
        }

        return orderCostsStatistics;
    }

    /**
```

```
     * TEST METHOD
     */
    @GetMapping("/getPossibilityOfEmptyQueue")
    public @ResponseBody
    BigDecimal getPossibilityOfEmptyQueue(@RequestParam float
minResources,
                                          @RequestParam int channels) {
        return countPossibilityOfEmptyQueue(minResources, channels);
    }

    /**
     * TEST METHOD
     */
    @GetMapping("/getPossibilityOfQueue")
    public @ResponseBody
    BigDecimal getPossibilityOfQueue(@RequestParam float
minResources,
                                     @RequestParam int channels)
{
        return countPossibilityOfQueue(minResources, channels,
countPossibilityOfEmptyQueue(minResources, channels).floatValue());
    }

    /**
     * TEST METHOD
     */
    @GetMapping("/getCountOfClientsInQueue")
    public @ResponseBody
    BigDecimal getCountOfClientsInQueue(@RequestParam float
minResources,
                                        @RequestParam int channels) {
        return countClientsInQueue(minResources, channels,
countPossibilityOfEmptyQueue(minResources, channels).floatValue());
    }

    /**
     * TEST METHOD
     */
    @GetMapping("/getTimeOfWaiting")
    public @ResponseBody
    BigDecimal getTimeOfWaiting(@RequestParam float minResources,
                                @RequestParam int channels,
                                @RequestParam float intensity) {
        return countTimeOfWaiting(minResources, channels,
intensity);
    }

    private BigDecimal getCosts(float minResources, int channels) {
        float intensity = getIntensity();
```

```
        return new BigDecimal(channels / intensity + channels *
countTimeOfWaiting(minResources, channels,
intensity).floatValue()).setScale(2, RoundingMode.HALF_UP);
    }

    private BigDecimal countTimeOfWaiting(float minResources, int
channels, float intensity) {
        return countClientsInQueue(minResources, channels,
countPossibilityOfEmptyQueue(minResources,
channels).floatValue()).divide(new BigDecimal(intensity),
RoundingMode.HALF_UP);
    }

    //we need to get orders per hour
    private float getIntensity() {
        return (float)
truckOrderRepository.findAllByCreateTimeBetween(LocalDateTime.now().
minusDays(STATISTICS_DAYS_PERIOD), LocalDateTime.now()).size() /
(STATISTICS_DAYS_PERIOD * 12); //we multiply 15 days on 12 hours (12
hours -> 1 working day)
    }

    private BigDecimal countPossibilityOfEmptyQueue(float
minResources, int channels) {
        float result = 1;
        for (int i = 1; i <= channels; i++) {
            result += Math.pow(minResources, i) /
factorialUsingStreams(i);
        }
        result += Math.pow(minResources, channels + 1) /
(factorialUsingStreams(channels) * (channels - minResources));
        BigDecimal bd = new
BigDecimal(Double.toString(Math.pow(result, -1)));
        bd = bd.setScale(3, RoundingMode.HALF_UP);

        return bd;
    }

    private BigDecimal countPossibilityOfQueue(float minResources,
int channels, float possibilityOfEmptyQueue) {
        double result = Math.pow(minResources, channels + 1) /
(factorialUsingStreams(channels) * (channels - minResources)) *
possibilityOfEmptyQueue;
        BigDecimal bd = new BigDecimal(Double.toString(result));
        bd = bd.setScale(3, RoundingMode.HALF_UP);
        return bd;
    }

    private BigDecimal countClientsInQueue(float minResources, int
channels, float possibilityOfEmptyQueue) {
```

```
        double result = Math.pow(minResources, channels + 1) /
(channels * factorialUsingStreams(channels) * Math.pow(1 -
minResources/channels, 2)) * possibilityOfEmptyQueue;
        BigDecimal bd = new BigDecimal(Double.toString(result));
        bd = bd.setScale(3, RoundingMode.HALF_UP);
        return bd;
    }
    private int factorialUsingStreams(int n) {
        return IntStream.rangeClosed(1, n).reduce(1, (int x, int y)
-> x * y);
    }
}
```

Клас RoleRepository

```
package com.truck.truckcontent.dao.trucks;

import com.truck.truckcontent.model.trucks.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface RoleRepository extends JpaRepository<Role, Integer>
{
    Role findByRole(String role);
}
```

Клас TruckClientRepository

```
package com.truck.truckcontent.dao.trucks;

import com.truck.truckcontent.model.trucks.TruckClient;
import org.springframework.data.repository.CrudRepository;

public interface TruckClientRepository extends
CrudRepository<TruckClient, Integer> {
}
```

Клас TruckDriverRepository

```
package com.truck.truckcontent.dao.trucks;

import com.truck.truckcontent.model.trucks.TruckDriver;
import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface TruckDriverRepository extends
CrudRepository<TruckDriver, Integer> {

    TruckDriver findByEmail(String email);

    TruckDriver findByPhone(String phone);
```

```
    List<TruckDriver> findByTruckIsNull();

    List<TruckDriver> findByTruckIsNotNull();

    long count();
}
```

Клас TruckOrderRepository

```
package com.truck.truckcontent.dao.trucks;

import com.truck.truckcontent.model.trucks.OrderCountStatistics;
import com.truck.truckcontent.model.trucks.OrderLoadStatistics;
import com.truck.truckcontent.model.trucks.OrderStatusStatistics;
import com.truck.truckcontent.model.trucks.TruckDriver;
import com.truck.truckcontent.model.trucks.TruckOrder;
import com.truck.truckcontent.type.TruckOrderStatus;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

import java.time.LocalDateTime;
import java.util.List;

public interface TruckOrderRepository extends
CrudRepository<TruckOrder, Integer> {

    List<TruckOrder> findAllByTruckOrderStatus(TruckOrderStatus
truckOrderStatus);

    List<TruckOrder> findAllByTruckDriver(TruckDriver truckDriver);

    List<TruckOrder> findAllByCreateTimeBetween(LocalDateTime
createTimeFrom, LocalDateTime createTimeTo);

    long count();

    @Query("SELECT new
com.truck.truckcontent.model.trucks.OrderCountStatistics(v.createDat
e, COUNT(v)) FROM TruckOrder v GROUP BY v.createDate ORDER BY
v.createDate")
    List<OrderCountStatistics>
collectOrderCountStatisticsForLastMonth();

    @Query("SELECT new
com.truck.truckcontent.model.trucks.OrderLoadStatistics(v.createDate
, SUM(v.completionTime)) FROM TruckOrder v GROUP BY v.createDate
ORDER BY v.createDate")
    List<OrderLoadStatistics> collectLoadStatisticsForLastMonth();
```

```
    @Query("SELECT new
com.truck.truckcontent.model.trucks.OrderStatusStatistics(v.truckOrd
erStatus, COUNT(v)) FROM TruckOrder v WHERE v.truckOrderStatus IN
(:statuses) GROUP BY v.truckOrderStatus")
    List<OrderStatusStatistics>
collectTruckOrderStatusStatistics(List<TruckOrderStatus> statuses);

    @Query("SELECT AVG(v.completionTime) FROM TruckOrder v")
    int countAverageServiceTime();
}
```

Клас TruckRepository

```
package com.truck.truckcontent.dao.trucks;

import com.truck.truckcontent.model.trucks.Truck;
import
org.springframework.data.repository.PagingAndSortingRepository;

import java.util.List;

public          interface          TruckRepository          extends
PagingAndSortingRepository<Truck, Integer> {

    Truck findByNumber(String number);

    List<Truck> findByTruckDriverIsNull();

    List<Truck> findByTruckDriverIsNotNull();

    long count();
}
```

Клас TruckOrderStatusConverter

```
package com.truck.truckcontent.model.trucks.converter;

import com.google.common.collect.ImmutableMap;
import com.truck.truckcontent.type.TruckOrderStatus;

import javax.persistence.AttributeConverter;
import java.util.Map;

public       class       TruckOrderStatusConverter       implements
AttributeConverter<TruckOrderStatus, String> {

    private Map<TruckOrderStatus, String> map = ImmutableMap.of(
            TruckOrderStatus.PENDING,
TruckOrderStatus.PENDING.getStatus(),
```

```
            TruckOrderStatus.REJECTED,
TruckOrderStatus.REJECTED.getStatus(),
            TruckOrderStatus.PROCESSING,
TruckOrderStatus.PROCESSING.getStatus(),
            TruckOrderStatus.COMPLETED,
TruckOrderStatus.COMPLETED.getStatus()
    );

    @Override
    public      String      convertToDatabaseColumn(TruckOrderStatus
truckOrderStatus) {
        if (truckOrderStatus == null) {
            return null;
        }

        return map.get(truckOrderStatus);
    }

    @Override
    public TruckOrderStatus convertToEntityAttribute(String dbData) {
        if (dbData == null) {
            return null;
        }

        return map.entrySet().stream()
                .filter(entry -> dbData.equals(entry.getValue()))
                .map(Map.Entry::getKey)
                .findFirst().orElse(null);
    }
}
```

Клас OrderCostsStatistics

```
package com.truck.truckcontent.model.trucks;
import java.math.BigDecimal;
public class OrderCostsStatistics {
    private int channels;
    private BigDecimal possibilityOfEmptyQueue;
    private BigDecimal avgClientsInQueue;
    private BigDecimal relativeCosts;

    public     OrderCostsStatistics(int     channels,     BigDecimal
possibilityOfEmptyQueue,  BigDecimal  avgClientsInQueue,  BigDecimal
relativeCosts) {
        this.channels = channels;
        this.possibilityOfEmptyQueue = possibilityOfEmptyQueue;
        this.avgClientsInQueue = avgClientsInQueue;
        this.relativeCosts = relativeCosts;
    }

    public int getChannels() {
```

```
        return channels;
    }

    public void setChannels(int channels) {
        this.channels = channels;
    }

    public BigDecimal getPossibilityOfEmptyQueue() {
        return possibilityOfEmptyQueue;
    }

    public         void         setPossibilityOfEmptyQueue(BigDecimal
possibilityOfEmptyQueue) {
        this.possibilityOfEmptyQueue = possibilityOfEmptyQueue;
    }

    public BigDecimal getAvgClientsInQueue() {
        return avgClientsInQueue;
    }

    public void setAvgClientsInQueue(BigDecimal avgClientsInQueue) {
        this.avgClientsInQueue = avgClientsInQueue;
    }

    public BigDecimal getRelativeCosts() {
        return relativeCosts;
    }

    public void setRelativeCosts(BigDecimal relativeCosts) {
        this.relativeCosts = relativeCosts;
    }
}
```

Клас OrderCountStatistics

```
package com.truck.truckcontent.model.trucks;

import java.time.LocalDate;

public         class         OrderCountStatistics         implements
Comparable<OrderCountStatistics> {

    private LocalDate createDate;
    private long count;

    public OrderCountStatistics(LocalDate createDate, long count) {
        this.createDate = createDate;
        this.count = count;
    }

    public LocalDate getCreateDate() {
```

```
        return createDate;
    }

    public void setCreateDate(LocalDate createDate) {
        this.createDate = createDate;
    }

    public long getCount() {
        return count;
    }

    public void setCount(long count) {
        this.count = count;
    }

    @Override
    public int compareTo(OrderCountStatistics o) {
        return this.getCreateDate().compareTo(o.getCreateDate());
    }
}
```

Клас TruckDriverDetailsService

```
package com.truck.truckcontent.service;

import com.truck.truckcontent.model.trucks.Role;
import com.truck.truckcontent.model.trucks.TruckDriver;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import
org.springframework.security.core.userdetails.UsernameNotFoundExcept
ion;
import org.springframework.stereotype.Service;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;

import javax.transaction.Transactional;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

@Service
public class TruckDriverDetailsService implements UserDetailsService
{
```

```
    @Autowired
    private TruckDriverService truckDriverService;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        TruckDriver truckDriver =
truckDriverService.findTruckDriverByEmail(email);
        List<GrantedAuthority> authorities =
getUserAuthority(truckDriver.getRoles());
        return buildUserForAuthentication(truckDriver, authorities);
    }

    private List<GrantedAuthority> getUserAuthority(Set<Role>
userRoles) {
        Set<GrantedAuthority> roles = new HashSet<>();
        for (Role role : userRoles) {
            roles.add(new SimpleGrantedAuthority(role.getRole()));
        }
        List<GrantedAuthority> grantedAuthorities = new
ArrayList<>(roles);
        return grantedAuthorities;
    }

    private UserDetails buildUserForAuthentication(TruckDriver
truckDriver, List<GrantedAuthority> authorities) {
        return new
org.springframework.security.core.userdetails.User(truckDriver.getEm
ail(), truckDriver.getPassword(),
                truckDriver.getActive(), true, true, true,
authorities);
    }
}
```

Клас TruckDriverService

```
package com.truck.truckcontent.service;

import com.truck.truckcontent.dao.trucks.RoleRepository;
import com.truck.truckcontent.dao.trucks.TruckDriverRepository;
import com.truck.truckcontent.model.trucks.Role;
import com.truck.truckcontent.model.trucks.TruckDriver;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.Collections;
import java.util.HashSet;
```

```
@Service
public class TruckDriverService {


    private TruckDriverRepository truckDriverRepository;
    private RoleRepository roleRepository;
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Autowired
    public TruckDriverService(TruckDriverRepository
truckDriverRepository,
                        RoleRepository roleRepository,
                        BCryptPasswordEncoder bCryptPasswordEncoder)
{
        this.truckDriverRepository = truckDriverRepository;
        this.roleRepository = roleRepository;
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;
    }

    public TruckDriver findTruckDriverByEmail(String email) {
        return truckDriverRepository.findByEmail(email);
    }

    public TruckDriver saveUser(TruckDriver truckDriver) {

truckDriver.setPassword(bCryptPasswordEncoder.encode(truckDriver.get
Password()));
        truckDriver.setActive(true);
        Role userRole = roleRepository.findByRole("DRIVER");
        truckDriver.setRoles(new
HashSet<>(Collections.singletonList(userRole)));
        return truckDriverRepository.save(truckDriver);
    }


}
```

Клас TruckContentApplication

```
package com.truck.truckcontent;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TruckContentApplication {

     public static void main(String[] args) {
          SpringApplication.run(TruckContentApplication.class,
args);}
```

ДОДАТОК Б

Відомість кваліфікаційної роботи магістра

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	95с.
	Інші документи	
2.	Презентаційні матеріали	15 плакатів

		Прізвище та ініціали.	Підп.	Дата			
Розробила		Левицька М.В.			Система оцінки ефективності управління заявками для логістичного підприємства	Шифр групи	Код напр./спец.
Перевірив		Шевченко О.Ю.				СШІм-20-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					