

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО  
Научный руководитель,  
доцент департамента  
Программной инженерии

УТВЕРЖДАЮ  
Академический руководитель  
образовательной программы  
«Программная инженерия»

\_\_\_\_\_  
Г. Н. Жукова  
«\_\_\_» \_\_\_\_\_ 2022 г.

\_\_\_\_\_  
В. В. Шилов  
«\_\_\_» \_\_\_\_\_ 2022 г.

**ВЕБ-ПРИЛОЖЕНИЕ “NOTIPRICE”**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.06.02-01 12 01-1-ЛУ**

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.06.02-01 12 01-1-1

Исполнитель

студент группы БПИ198

\_\_\_\_\_/Бакытбек уулу Н. /

«11» мая 2022 г.

**Москва 2022**

УТВЕРЖДЕН  
RU.17701729.06.02-01 12 01-1

**ВЕБ-ПРИЛОЖЕНИЕ “NOTIPRICE”**

**Текст программы**

**RU.17701729.06.02-01 12 01-1**

**Листов 18**

<i>Инв. № подл</i>	<i>Подп. и дата</i>	<i>Взам. инв. №</i>	<i>Инв. № дубл.</i>	<i>Подп. и дата</i>
RU.17701729.06.02-01 12 01-1				

## Оглавление

1	Controller.....	3
1.1	AuthController.....	3
1.2	ProductController .....	4
1.3	UserController .....	6
2	Config .....	7
2.1	ServiceConfig.....	7
3	dao.....	9
3.1	ProductDao .....	9
3.2	SubscriptionDao .....	12
3.3	UserDao.....	14
3.4	ProductDto.....	16
3.5	UserDto .....	18
4	Entity.....	19
4.1	Product.....	19
4.2	Subscription.....	20
4.3	User .....	20
5	exception.....	21
5.1	ControllerExceptionHandler .....	21
5.2	RestTemplateResponseErrorHandler .....	22
6	scanner .....	23
6.1	ScarperKt.....	26
7	security.....	27
7.1	CustomUserDetails.....	27
7.2	JwtFilter.....	28
7.3	CustomUserDetailsService.....	30
7.4	JwtProvider.....	30

8	telegram .....	32
9	notiprice .....	33
9.1	NotipriceApplication .....	33
10	Service .....	34
10.1	ProductService .....	34
10.2	UserService .....	36
	Лист регистрации изменений .....	38

# 1 Controller

## 1.1 AuthController

```
package com.notiprice.controller

import com.notiprice.security.JwtProvider
import com.notiprice.dto.UserDto
import com.notiprice.dto.toEntity
import com.notiprice.entity.toDto
import com.notiprice.service.UserService
import org.springframework.web.bind.annotation.*

/**
 * Контроллер для аутентификации и регистрации.
 */
@RestController
@RequestMapping("/auth")
class AuthController(private val userService: UserService, private val jwtProvider:
JwtProvider) {
    /**
     * Регистрация пользователя.
     */
    @PostMapping("sign-up")
    fun addUser(@RequestBody user: UserDto): UserDto {
        val savedUser = userService.addUser(user.toEntity()).toDto()
        savedUser.password = ""
        return savedUser
    }

    /**
     * Проверяет пароль пользователя, если пароли совпадают, возвращает токен, если нет, то
     * бросает исключение.
     */
    @PostMapping("/sign-in")
    fun login(@RequestBody user: UserDto): String {
        val savedUser = userService.login(user.toEntity()).toDto()

        return jwtProvider.generateToken(savedUser.username)
    }
}
```

## 1.2 ProductController

```
package com.notiprice.controller

import com.notiprice.dto.ProductDto
import com.notiprice.dto.toEntity
import com.notiprice.entity.Product
import com.notiprice.entity.toDto
import com.notiprice.service.ProductService
import org.springframework.http.MediaType
import org.springframework.web.bind.annotation.*

/**
 * Контроллер отслеживаемых товаров.
 */
@RestController
@RequestMapping("/products")
class ProductController(private val productService: ProductService) {
    /**
     * Создание нового товара.
     */
    @PostMapping
    fun addProduct(@RequestBody product: ProductDto, @RequestParam username: String):
    ProductDto {

        return productService.addProduct(product.toEntity(), username).toDto()
    }

    /**
     * Изменение данных о товаре.
     */
    @PutMapping("/{id}")
    fun updateProduct(@PathVariable id: Long, @RequestBody product: ProductDto) {

        productService.updateProduct(product.toEntity())
    }

    /**
     * Удаление товара.
     */
    @DeleteMapping("/{id}")
    fun deleteProduct(@PathVariable id: Long) {

        productService.deleteProduct(id)
    }

    /**
     * Получение товара по пользовательскому имени.
     */
    @GetMapping
    fun getProducts(@RequestParam username: String): List<ProductDto> {

        return productService.getAllUserProducts(username).map(Product::toDto)
    }
}
```

```

    }

    /**
     * Получение товара по идентификатору.
     */
    @GetMapping("/{id}")
    fun getProductById(@PathVariable id: Long): ProductDto {

        return productService.getProductById(id).toDto()
    }

    /**
     * Получение xpath продукта по URL. В базе данных ищутся xpath от базового домена URL.
     * Найденные xpath проверяются можно ли получить значение по этому xpath значение.
     * Если таких несколько, то выбирается самый популярный.
     */
    @GetMapping("/xpath")
    fun getProductXPathByUrl(@RequestParam url: String): String {

        return productService.getProductXPathByUrl(url)
    }

    /**
     * Получает страницу по URL и выделяет элемент по xpath и возвращает страницу.
     */
    @GetMapping(value = ["/html"], produces = [MediaType.TEXT_HTML_VALUE])
    fun getHtmlWithHighlightedElement(@RequestParam url: String, @RequestParam xpath:
String): String {

        return productService.getHtmlWithHighlightedElement(url, xpath)
    }
}

```

### 1.3 UserController

```
package com.notiprice.controller

import com.notiprice.dto.UserDto
import com.notiprice.dto.toEntity
import com.notiprice.entity.toDto
import com.notiprice.service.UserService
import org.springframework.web.bind.annotation.*

/**
 * Контроллер пользователей.
 */
@RestController
@RequestMapping("/users")
class UserController(private val userService: UserService) {

    /**
     * Изменение данных о пользователе.
     */
    @PutMapping("/{id}")
    fun updateUser(@PathVariable id: Long, @RequestBody user: UserDto) {

        userService.updateUser(user.toEntity())
    }

    /**
     * Удаление пользователя.
     */
    @DeleteMapping("/{id}")
    fun deleteUser(@PathVariable id: Long) {
        userService.deleteProduct(id)
    }

    /**
     * Получение данных о пользователе по пользовательскому имени.
     */
    @GetMapping("/get")
    fun getUserByUsername(@RequestParam username: String): UserDto {

        return userService.getUserByUsername(username).toDto()
    }

    /**
     * Получение данных о пользователе по идентификатору.
     */
    @GetMapping("/{id}")
    fun getUserById(@PathVariable id: Long): UserDto {
        return userService.getProductById(id).toDto()
    }
}
```



## 2 Config

### 2.1 ServiceConfig

```
package com.notiprice.config

import com.notiprice.security.JwtFilter
import com.notiprice.exception.RestTemplateResponseErrorHandler
import org.springframework.beans.factory.annotation.Value
import org.springframework.boot.web.client.RestTemplateBuilder
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration
import org.springframework.scheduling.annotation.EnableScheduling
import org.springframework.security.config.annotation.web.builders.HttpSecurity
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
import org.springframework.security.config.http.SessionCreationPolicy
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter
import org.springframework.web.client.RestTemplate
import org.springframework.web.cors.CorsConfiguration
import org.springframework.web.cors.CorsConfigurationSource
import org.springframework.web.cors.UrlBasedCorsConfigurationSource
import java.time.Duration
import java.util.*

/**
 * Конфигурация для программы.
 */
@Configuration
@EnableScheduling
@EnableWebSecurity
class ServiceConfig(private val jwtFilter: JwtFilter) : WebSecurityConfigurerAdapter() {
    /**
     * Создание экземпляра класса RestTemplate.
     * "Instances of the JdbcTemplate class are threadsafe once configured"
     * https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/jdbc.html
     */
    @Bean
    fun restTemplate(
        builder: RestTemplateBuilder,
        @Value("\${timeout.seconds.connect}") connectTimeout: Long,
        @Value("\${timeout.seconds.read}") readTimeout: Long
    ): RestTemplate = builder
        .errorHandler(RestTemplateResponseErrorHandler())
        .setConnectTimeout(Duration.ofSeconds(connectTimeout))
        .setReadTimeout(Duration.ofSeconds(readTimeout))
        .build()

    /**
     * Конфигурация безопасности программы.
     */
}
```

```

    */
    override fun configure(http: HttpSecurity?) {
        http!!.httpBasic().disable().csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and().authorizeRequests()
            .antMatchers("/products*").hasRole("USER")
            .antMatchers("/users*").hasRole("USER")
            .antMatchers("/auth*").permitAll()
            .and()
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter::class.java)
            .cors().configurationSource(corsConfigurationSource())
    }

    /**
     * Конфигурация для CORS Policy.
     */
    @Bean
    fun corsConfigurationSource(): CorsConfigurationSource {
        val configuration = CorsConfiguration()
        configuration.allowedOrigins = listOf("")
        configuration.allowedMethods = listOf("GET", "POST", "PUT", "DELETE")
        configuration.allowedHeaders = Collections.singletonList("")
        val source = UrlBasedCorsConfigurationSource()
        source.registerCorsConfiguration("/**", configuration)
        return source
    }
}

```

## 3 dao

### 3.1 ProductDao

```
package com.notiprice.dao

import com.notiprice.entity.Product
import org.springframework.jdbc.core.JdbcTemplate
import org.springframework.jdbc.core.query
import org.springframework.jdbc.support.GeneratedKeyHolder
import org.springframework.jdbc.support.KeyHolder
import org.springframework.stereotype.Component
import java.sql.Connection
import java.sql.ResultSet
import java.sql.Statement
import java.sql.Types

/**
 * DAO продукта для работы с базой данных.
 */
@Component
class ProductDao(private val jdbcTemplate: JdbcTemplate) {
    /**
     * Сохранение экземпляра класса Product в базе данных.
     */
    fun save(product: Product): Product {
        val keyHolder: KeyHolder = GeneratedKeyHolder()

        val numOfUpdates = jdbcTemplate.update({ connection: Connection ->
            val ps = connection
                .prepareStatement(
                    "insert into $products " +
                    "($name, $price, $currency, $url, $xpath, $priceStr, $lastCheck)
" +
                    "values (?, ?, ?, ?, ?, ?, ?)",
                    Statement.RETURN_GENERATED_KEYS
                )
            ps.setString(1, product.name)
            ps.setDouble(2, product.price)
            ps.setString(3, product.currency)
            ps.setString(4, product.url)
            ps.setString(5, product.xpath)
            ps.setString(6, product.priceStr)
            ps.setLong(7, product.lastCheck)
            ps
        }, keyHolder, )

        require(numOfUpdates == 1)

        product.id = keyHolder.key as Long

        return product
    }
}
```

```

}

/**
 * Получение продукта по идентификатору.
 */
fun findByIdOrNull(id: Long): Product? {

    return jdbcTemplate.query(
        "select * from $products where $id = ?",
        arrayOf<Any>(id),
        intArrayOf(Types.BIGINT)
    ) { rs: ResultSet, _: Int ->
        Product(
            rs.getLong(Companion.id),
            rs.getString(name),
            rs.getDouble(price),
            rs.getString(currency),
            rs.getString(url),
            rs.getString(xpath),
            rs.getString(priceStr),
            rs.getLong(lastCheck),
        )
    }.firstOrNull()
}

/**
 * Изменение данных о продукте.
 */
fun update(product: Product) {

    val numOfUpdates = jdbcTemplate.update(
        "update $products " +
            "set $name = ?, " +
            "$price = ?, " +
            "$currency = ?, " +
            "$url = ?, " +
            "$xpath = ?, " +
            "$priceStr = ?, " +
            "$lastCheck = ? " +
            "where $id = ?",
        product.name,
        product.price,
        product.currency,
        product.url,
        product.xpath,
        product.priceStr,
        product.lastCheck,
        product.id
    )

    require(numOfUpdates == 1)
}

```

```

/**
 * Удаление продукта.
 */
fun delete(productId: Long) {

    val numOfUpdates = jdbcTemplate.update(
        "delete $products where id = ?",
        productId
    )

    require(numOfUpdates == 1)
}

/**
 * Получение товаров пользователя по пользовательскому имени.
 */
fun findAllUserProducts(username: String): List<Product> = jdbcTemplate.query(
    "select * from $products join ${SubscriptionDao.subscriptions} on " +
        "$products.$id = " +
        "${SubscriptionDao.subscriptions}.${SubscriptionDao.productId} join ${UserDao.users} on " +
        "${UserDao.users}.${UserDao.chatId} = " +
        "${SubscriptionDao.subscriptions}.${SubscriptionDao.chatId} " +
        "where ${UserDao.users}.${UserDao.username} = ?", username
) { rs: ResultSet, _: Int ->
    Product(
        rs.getLong(id),
        rs.getString(name),
        rs.getDouble(price),
        rs.getString(currency),
        rs.getString(url),
        rs.getString(xpath),
        rs.getString(priceStr),
        rs.getLong(lastCheck)
    )
}

/**
 * Получение товаров для сканирования. Возвращает товары, которые не проверялись
определенный
 * интервал времени в секундах timeInterval.
 */
fun findToCheck(timeIntervalInSeconds: Int, limit: Int): List<Product> {
    val now = System.currentTimeMillis()
    return jdbcTemplate.query(
        "select * from $products where " +
            "$lastCheck + ? <= ?" +
            "order by $lastCheck limit ?",
        timeIntervalInSeconds * 1000, now, limit
    ) { rs: ResultSet, _: Int ->
        Product(
            rs.getLong(id),
            rs.getString(name),
            rs.getDouble(price),

```

```

        rs.getString(currency),
        rs.getString(url),
        rs.getString(xpath),
        rs.getString(priceStr),
        rs.getLong(lastCheck)
    )
}
}

/**
 * Получение xpath-ов по URL.
 */
fun findXPathByUrl(baseUrl: String): List<String> {

    return jdbcTemplate.query(
        "select $xpath, count(id) as cnt from products where $url like ? group by $xpath
order by cnt desc",
        "%$baseUrl%"
    ) { rs: ResultSet, _: Int ->
        rs.getString(xpath)
    }
}

companion object {
    const val products = "products"
    const val id = "id"
    const val name = "name"
    const val price = "price"
    const val currency = "currency"
    const val url = "url"
    const val xpath = "xpath"
    const val priceStr = "price_str"
    const val lastCheck = "last_check"
}
}

```

## 3.2 SubscriptionDao

```

package com.notiprice.dao

import com.notiprice.entity.Subscription
import org.springframework.jdbc.core.JdbcTemplate
import org.springframework.stereotype.Component
import java.sql.ResultSet
import java.sql.Types

/**
 * DAO для класса Subscription для работы с базой данных.
 */
@Component
class SubscriptionDao(private val jdbcTemplate: JdbcTemplate) {
    /**

```

```

* Добавление в базу данных экземпляра класса Subscription.
*/
fun save(subscription: Subscription): Subscription {

    val numOfUpdates = jdbcTemplate.update(
        "insert into $subscriptions ($chatId, $productId) values (?, ?)",
        subscription.chatId, subscription.productId
    )

    require(numOfUpdates == 1)

    return subscription
}

/**
* Получение Subscription по идентификатору.
*/
fun findByIdOrNull(chatId: Long, productId: Long): Subscription? {

    return jdbcTemplate.query(
        "select * from $subscriptions where $chatId = ? and $productId = ?",
        arrayOf<Any>(chatId, productId),
        intArrayOf(Types.BIGINT, Types.BIGINT)
    ) { rs: ResultSet, _: Int ->
        Subscription(
            rs.getLong(Companion.chatId),
            rs.getLong(Companion.productId)
        )
    }.firstOrNull()
}

/**
* Изменение данных Subscription.
*/
fun update(product: Subscription) {

    val numOfUpdates = jdbcTemplate.update(
        "update $subscriptions " +
        "set $chatId = ?, " +
        "$productId = ? " +
        "where $chatId = ? and $productId = ?",
        product.chatId, product.productId
    )

    require(numOfUpdates == 1)
}

/**
* Удаление Subscription.
*/
fun delete(chatId: Long, productId: Long) {

    val numOfUpdates = jdbcTemplate.update(

```

```

        "delete $subscriptions where ${Companion.chatId} = ? and ${Companion.productId}
= ?",
        chatId, productId
    )

    require(numOfUpdates == 1)
}

/**
 * Получение идентификаторов чата по идентификатору продуктов.
 */
fun findChatIdsByProductId(productId: Long): List<Long> {

    return jdbcTemplate.query(
        "select distinct $chatId from $subscriptions where $productId = ?",
        arrayOf<Any>(productId),
        intArrayOf(Types.BIGINT)
    ) { rs: ResultSet, _: Int ->
        rs.getLong(chatId)
    }
}

companion object {
    const val subscriptions = "subscriptions"
    const val chatId = "chat_id"
    const val productId = "product_id"
}
}

```

### 3.3 UserDao

```

package com.notiprice.dao

import com.notiprice.entity.User
import org.springframework.jdbc.core.JdbcTemplate
import org.springframework.stereotype.Component
import java.sql.ResultSet
import java.sql.Types

/**
 * DAO для класса User для работы с базой данных.
 */
@Component
class UserDao(private val jdbcTemplate: JdbcTemplate) {
    /**
     * Добавление в базу данных экземпляра класса User.
     */
    fun save(user: User): User {

        val numOfUpdates = jdbcTemplate.update(
            "insert into users ($chatId, $username, $password) values (?, ?, ?)",
            user.chatId, user.username, user.password
        )
    }
}

```



```

    )
    require(numOfUpdates == 1)

    return user
}

/**
 * Получение User по идентификатору.
 */
fun findByIdOrNull(id: Long): User? {

    return jdbcTemplate.query(
        "select * from $users where $chatId = ?",
        arrayOf<Any>(id),
        intArrayOf(Types.BIGINT)
    ) { rs: ResultSet, _: Int ->
        User(
            rs.getLong(chatId),
            rs.getString(username),
            rs.getString(password)
        )
    }.firstOrNull()
}

/**
 * Получение User по пользовательскому имени.
 */
fun findByUsernameOrNull(name: String): User? {

    return jdbcTemplate.query(
        "select * from $users where $username = ?",
        arrayOf<Any>(name),
        intArrayOf(Types.VARCHAR)
    ) { rs: ResultSet, _: Int ->
        User(
            rs.getLong(chatId),
            rs.getString(username),
            rs.getString(password)
        )
    }.firstOrNull()
}

/**
 * Изменение данных User.
 */
fun update(user: User) {

    val numOfUpdates = jdbcTemplate.update(
        "update $users " +
            "set $chatId = ?, " +
            "$username = ?, " +
            "$password = ?",
        user.chatId, user.username, user.password
    )
}

```

```

        require(numOfUpdates == 1)
    }

    /**
     * Удаление User.
     */
    fun delete(userId: Long) {

        val numOfUpdates = jdbcTemplate.update(
            "delete $users where id = ?",
            userId
        )
        require(numOfUpdates == 1)
    }
    companion object {
        const val users = "users"
        const val chatId = "chat_id"
        const val username = "username"
        const val password = "password"
    }
}

```

### 3.4 ProductDto

```
package com.notiprice.dto
```

```
import com.notiprice.entity.Product
```

```

/**
 * DTO товара.
 */
data class ProductDto(
    /**
     * Идентификатор товара.
     */
    var id: Long = 0,
    /**
     * Название товара.
     */
    var name: String,
    /**
     * Цена товара в виде десятичного числа.
     */
    var price: Double = 0.0,
    /**
     * Валюта.
     */
    var currency: String = "",
    /**
     * URL на товар.
     */
    var url: String,
    /**

```

```
        * Путь до цены на странице товара.
        */
    var xpath: String,
    /**
        * Значение по пути цены товара на странице.
        */
    var priceStr: String = ""
)

fun ProductDto.toEntity() = Product(id, name, price, currency, url, xpath, priceStr, 0L)
```

### 3.5 UserDto

```
package com.notiprice.dto

import com.notiprice.entity.User

/**
 * DTO пользователя.
 */
data class UserDto(
    /**
     * Идентификатор чата пользователя в Телеграме. Первичный ключ.
     */
    val chatId: Long,
    /**
     * Логин пользователя.
     */
    val username: String,
    /**
     * Пароль пользователя.
     */
    var password: String,
)

fun UserDto.toEntity() = User(chatId, username, password)
```

## 4 Entity

### 4.1 Product

```
package com.notiprice.entity

import com.notiprice.dto.ProductDto

/**
 * Для отображения таблицы products из базы данных.
 */
data class Product(
    /**
     * Идентификатор товара.
     */
    var id: Long = 0,
    /**
     * Название товара.
     */
    var name: String,
    /**
     * Цена товара в виде десятичного числа.
     */
    var price: Double = 0.0,
    /**
     * Валюта.
     */
    var currency: String,
    /**
     * URL на товар.
     */
    var url: String,
    /**
     * Путь до цены на странице товара.
     */
    var xpath: String,
    /**
     * Значение по пути цены товара на странице.
     */
    var priceStr: String = "",
    /**
     * Timestamp времени последней проверки цены товара.
     */
    var lastCheck: Long
)

fun Product.toDto() = ProductDto(id, name, price, currency, url, xpath, priceStr)
```

## 4.2 Subscription

```
package com.notiprice.entity

/**
 * Для отображения таблицы subscriptions из базы данных.
 */
data class Subscription(
    /**
     * Внешний ключ таблицы users.
     */
    val chatId: Long,
    /**
     * Внешний ключ таблицы products.
     */
    val productId: Long
)
```

## 4.3 User

```
package com.notiprice.entity

import com.notiprice.dto.UserDto

/**
 * Для отображения таблицы users из базы данных.
 */
data class User(
    /**
     * Идентификатор чата пользователя в Телеграме. Первичный ключ.
     */
    val chatId: Long,
    /**
     * Логин пользователя.
     */
    val username: String,
    /**
     * Пароль пользователя.
     */
    var password: String,
)

fun User.toDto() = UserDto(chatId, username, password)
```

## 5 exception

### 5.1 ControllerExceptionHandler

```
package com.notiprice.exception

import mu.KotlinLogging
import org.springframework.http.HttpStatus
import org.springframework.web.bind.annotation.ExceptionHandler
import org.springframework.web.bind.annotation.ResponseStatus
import org.springframework.web.bind.annotation.RestControllerAdvice
import org.springframework.web.client.ResourceAccessException

private val Log = KotlinLogging.logger {}

/**
 * Перехватывает исключение, если они возникают в результате вызова контроллера.
 */
@RestControllerAdvice
class ControllerExceptionHandler {

    /**
     * Перехватывает исключения типа IllegalArgumentException.
     */
    @ExceptionHandler
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    fun handleNoSuchElementException(e: IllegalArgumentException): Map<String, String> {
        Log.warn(e.message, e)
        return ErrorResponse(e)
    }

    /**
     * Перехватывает исключения типа NoSuchElementException.
     */
    @ExceptionHandler
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    fun handleNoSuchElementException(e: NoSuchElementException): Map<String, String> {
        Log.warn(e.message, e)
        return ErrorResponse(e)
    }

    /**
     * Перехватывает исключения типа ResourceAccessException.
     */
    @ExceptionHandler
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    fun handleResourceAccessException(e: ResourceAccessException): Map<String, String> {
        Log.warn(e.message, e)
        return ErrorResponse(e)
    }

    /**
     * Перехватывает исключения типа Exception.
     */
}
```

```

    */
    @ExceptionHandler
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    fun handleException(e: Exception): Map<String, String> {
        Log.warn(e.message, e)
        return ErrorResponse(e)
    }

    /**
     * Создание ответа с информацией об ошибке.
     */
    private fun ErrorResponse(e: Exception): Map<String, String> = mapOf(
        "status" to "error",
        "exception" to e.javaClass.simpleName,
        "message" to e.message.orEmpty()
    )
}

```

## 5.2 RestTemplateResponseErrorHandler

```

package com.notiprice.exception

import mu.KotlinLogging
import org.slf4j.LoggerFactory
import org.springframework.http.client.ClientHttpResponse
import org.springframework.stereotype.Component
import org.springframework.web.client.ResponseErrorHandler

private val Log = KotlinLogging.logger {}

/**
 * Перехватывает исключения из RestTemplate.
 */
@Component
class RestTemplateResponseErrorHandler : ResponseErrorHandler {
    override fun hasError(httpResponse: ClientHttpResponse): Boolean =
        httpResponse.statusCode.is4xxClientError || httpResponse.statusCode.is5xxServerError
    override fun handleError(httpResponse: ClientHttpResponse) {
        if (httpResponse.statusCode.is4xxClientError) {
            Log.warn("Bad request to the external server", httpResponse)
        } else if (httpResponse.statusCode.is5xxServerError) {
            Log.error("The external server error", httpResponse)
        }
    }
}

```



## 6 scanner

```
package com.notiprice.scanner

import com.notiprice.dao.ProductDao
import com.notiprice.dao.SubscriptionDao
import com.notiprice.entity.Product
import kotlinx.coroutines.*
import mu.KotlinLogging
import org.springframework.beans.factory.annotation.Value
import org.springframework.http.ResponseEntity
import org.springframework.scheduling.annotation.EnableScheduling
import org.springframework.scheduling.annotation.Scheduled
import org.springframework.stereotype.Component
import org.springframework.web.client.RestTemplate
import java.util.concurrent.TimeUnit

private val logger = KotlinLogging.logger {}

/**
 * Сканирует продукты, если цена меняется, то отправляет сообщение пользователю.
 */
@EnableScheduling
@Component
class PriceScanner(
    /**
     * DAO продукта для работы с базой данных.
     */
    private val productDao: ProductDao,
    /**
     * DAO для класса Subscription для работы с базой данных.
     */
    private val subscriptionDao: SubscriptionDao,
    /**
     * Клиент для HTTP запросов.
     */
    private val restTemplate: RestTemplate,
    /**
     * Интервал времени в секундах. Показывает интервал проверки продуктов.
     */
    @Value("\${scan.recheck.in.seconds}") private val timeIntervalInSeconds: Int,
    /**
     * Сколько товаров обрабатывать одновременно.
     */
    @Value("\${process.product.limit}") private val limit: Int
) {
    /**
     * Запускает сканирование с интервалом fixedDelayString в секундах.
     * Проверяет товары, которые не проверялись определенный интервал времени в секундах
     timeIntervalInSeconds,
     * если их цена меняется, то пользователям отправляются сообщения через Телеграм бот.
     */
    @Scheduled(fixedDelayString = "\${scan.fixedDelay.in.seconds}", timeUnit =
```

```

TimeUnit.SECONDS)
    fun scan(

    ) = runBlocking {

        var products: List<Product>

        do {
            products = productDao.findToCheck(timeIntervalInSeconds, limit)

            products.map {
                launch(Dispatchers.IO) {

                    val currentPrice = getValueByXPath(url = it.url, xpath = it.xpath)

                    if (currentPrice == null || currentPrice == "") {
                        logger.info { "Cannot get price from the object" }
                        it.lastCheck = System.currentTimeMillis()
                        productDao.update(it)
                        return@launch
                    }

                    if (currentPrice == it.priceStr) {

                        logger.info { "Price wasn't changed: $currentPrice" }
                        it.lastCheck = System.currentTimeMillis()
                        productDao.update(it)
                        return@launch
                    }

                    sendNotifications(it, currentPrice)

                    it.lastCheck = System.currentTimeMillis()
                    it.priceStr = currentPrice
                    productDao.update(it)

                }
            }.joinAll()

        } while (products.isNotEmpty())

    }

/**
 * Отправка сообщения пользователям, которые следят за товаром.
 */
fun sendNotifications(product: Product, currentPrice: String) {
    val chatIds = subscriptionDao.findChatIdsByProductId(product.id)

    for (chatId in chatIds) {

        val text = "Price of the product was changed" +
            "\nname ${product.name}" +

```

```

        "\nprice $currentPrice" +
        "\nurl ${product.url}"

    val response: ResponseEntity<String> =
        restTemplate.getForEntity(
            "https://api.telegram.org/bot5119272724:AAGaZ5I0o10EpDAZIqT-
TXtJiJqBNxfpb_w/" +
                "sendMessage?chat_id=$chatId&text=$text",
            String::class.java
        )

    if (response.statusCode.is2xxSuccessful) {
        //logger.info { product.toString() }
        logger.info { "message was sent to telegram: new price: $currentPrice" }
    } else {
        logger.info { "Cannot send message..." }
    }
}
}
}
}

```

## 6.1 ScarperKt

```
package com.notiprice.scanner

import mu.KotlinLogging
import org.jsoup.Jsoup
import us.codecraft.xsoup.Xsoup

private val logger = KotlinLogging.logger {}

/**
 * Извлечение значения из страницы по xpath.
 */
fun getValueByXpath(url: String, xpath: String): String? =
    try {
        //      val apiKey = "cc4f9ce0-bcbc-11ec-94c9-1125c5e45be1"
        //      val apiEndPoint = "https://app.zenscrape.com/api/v1/get" +
        //          "?apikey=$apiKey" +
        //          "&url=$url"

        Xsoup.compile(xpath)
            .evaluate(Jsoup.connect(url).get())
            .elements.first()
            ?.childNodes()?.first()
            ?.outerHtml()?.replace("&nbsp;", " ")
    } catch (th: Throwable) {
        logger.warn { "Cannot read a value by this xpath" }
        null
    }
```

## 7 security

### 7.1 CustomUserDetails

```
package com.notiprice.security

import com.notiprice.entity.User
import org.springframework.security.core.GrantedAuthority
import org.springframework.security.core.authority.SimpleGrantedAuthority
import org.springframework.security.core.userdetails.UserDetails
import java.util.*

fun User.toCustomUserDetails(): CustomUserDetails {
    return CustomUserDetails(username, password,
        Collections.singletonList(SimpleGrantedAuthority("ROLE_USER")))
}
/**
 * Адаптер моего класса User для Spring Security.
 */
class CustomUserDetails(
    private val login: String,
    private val password: String,
    private val grantedAuthorities: MutableCollection<out GrantedAuthority>
) : UserDetails{
    /**
     * Получение ролей пользователя. В данной версии у пользователей только одна роль -
     * ROLE_USER/
     */
    override fun getAuthorities() = grantedAuthorities
    /**
     * Получение пароля пользователя.
     */
    override fun getPassword() = password
    /**
     * Получение пользовательского имени.
     */
    override fun getUsername() = login
    /**
     * Не имеет значения. Для совместимости.
     */
    override fun isAccountNonExpired() = true
    /**
     * Не имеет значения. Для совместимости.
     */
    override fun isAccountNonLocked() = true
    override fun isCredentialsNonExpired() = true
    override fun isEnabled() = true
}
```

## 7.2 JwtFilter

```
package com.notiprice.security

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken
import org.springframework.security.core.context.SecurityContextHolder
import org.springframework.stereotype.Component
import org.springframework.util.StringUtils.hasText
import org.springframework.web.filter.GenericFilterBean
import javax.servlet.FilterChain
import javax.servlet.HttpServletRequest
import javax.servlet.HttpServletResponse
import javax.servlet.http.HttpServletRequest

private const val AUTHORIZATION = "Authorization"

/**
 * Фильтр для поступающих в программу запросов для авторизации.
 */
@Component
class JwtFilter(
    /**
     * Класс для обработки токена.
     */
    private val jwtProvider: JwtProvider,
    /**
     * Сервис для класса CustomUserDetails.
     */
    private val customUserDetailsService: CustomUserDetailsService
) : GenericFilterBean() {
    /**
     * Фильтр для поступающих в программу запросов для авторизации.
     */
    override fun doFilter(request: HttpServletRequest?, response: HttpServletResponse?, chain: FilterChain?) {

        val token = getTokenFromRequest(request as HttpServletRequest)

        if(token != null && jwtProvider.validateToken(token)) {
            val userLogin = jwtProvider.getLoginFromToken(token)

            val customUserDetails = customUserDetailsService.loadUserByUsername(userLogin)

            val auth =
                UsernamePasswordAuthenticationToken(customUserDetails, null,
                customUserDetails.authorities)

            SecurityContextHolder.getContext().authentication = auth
        }

        chain!!.doFilter(request, response)
    }
}
```

```
/**
 * Получение токена из запроса.
 */
private fun getTokenFromRequest(request: HttpServletRequest): String? {
    val bearer = request.getHeader(AUTHORIZATION)

    if(hasText(bearer) && bearer.startsWith("Bearer ")) {
        return bearer.substring(7)
    }

    return null
}
}
```

### 7.3 CustomUserDetailsService

```
package com.notiprice.security

import com.notiprice.service.UserService
import org.springframework.security.core.userdetails.UserDetailsService
import org.springframework.stereotype.Component

/**
 * Сервис для класса CustomUserDetails.
 */
@Component
class CustomUserDetailsService(private val userService: UserService) : UserDetailsService {
    /**
     * Получение экземпляра класса CustomUserDetails по пользовательскому имени.
     */
    override fun loadUserByUsername(username: String?): CustomUserDetails {
        val user = userService.getUserByUsername(username!!)
        return user.toCustomUserDetails()
    }
}
```

### 7.4 JwtProvider

```
package com.notiprice.security

import io.jsonwebtoken.Jwts
import io.jsonwebtoken.SignatureAlgorithm
import org.springframework.beans.factory.annotation.Value
import org.springframework.stereotype.Component
import java.time.LocalDate
import java.time.ZoneId
import java.util.*

/**
 * Класс для обработки токена.
 */
@Component
class JwtProvider(
    /**
     * Секретный ключ для шифрования.
     */
    @Value("\${jwt.secret}") private val secret: String
) {
    /**
     * Генерация токена.
     */
    fun generateToken(login: String): String {
        val date =
            Date.from(LocalDate.now().plusDays(3).atStartOfDay(ZoneId.systemDefault()).toInstant())

        return Jwts.builder()
            .setSubject(login)
            .setExpiration(date)
    }
}
```



```

        .signWith(SignatureAlgorithm.HS512, secret)
        .compact()
    }

    /**
     * Валидация токена.
     */
    fun validateToken(token: String): Boolean {
        return try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token)
            true
        } catch (ex: Exception) {
            false
        }
    }

    /**
     * Получение пользовательского имени из токена.
     */
    fun getLoginFromToken(token: String) : String {
        val claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(token).body

        return claims.subject
    }
}

```

## 8 telegram

```
package com.notiprice.telegram
import org.telegram.telegrambots.bots.TelegramLongPollingBot
import org.telegram.telegrambots.meta.api.methods.send.SendMessage
import org.telegram.telegrambots.meta.api.objects.Update

/**
 * Телеграм бот. Обрабатывает запросы пользователей в чате.
 */
class NotipriceTelegramBot(
    private val botToken: String,
    private val botUsername: String,
    private val notipriceUrl: String
) : TelegramLongPollingBot() {

    override fun getBotToken() = botToken
    override fun getBotUsername() = botUsername

    /**
     * Обработка запросов пользователей.
     */
    override fun onUpdateReceived(update: Update) {
        if (!(update.hasMessage() && update.message.hasText())) {
            return;
        }

        if (update.message.text == commandSignUp) {
            handleSignUp(update.message.chatId.toString())
        }
    }

    /**
     * Отправка ссылки для регистрации.
     */
    private fun handleSignUp(chatId: String) {
        val message = SendMessage()
        message.chatId = chatId
        message.text = "Please go to $notipriceUrl/sign-up/$chatId"
        execute(message)
    }

    companion object {
        const val commandSignUp = "/signup"
    }
}
```

## 9 notiprice

### 9.1 NotipriceApplication

```
package com.notiprice

import com.notiprice.telegram.NotipriceTelegramBot
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication
import org.telegram.telegrambots.meta.TelegramBotsApi
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession

@SpringBootApplication
class NotipriceApplication

fun main(args: Array<String>) {

    val context = runApplication<NotipriceApplication>(*args)

    val botsApi = TelegramBotsApi(DefaultBotSession::class.java)
    botsApi.registerBot(
        NotipriceTelegramBot(
            context.environment.getProperty("bot.token")!!,
            context.environment.getProperty("bot.username")!!,
            context.environment.getProperty("notiprice.url")!!
        )
    )
}
```

## 10 Service

### 10.1 ProductService

```
package com.notiprice.service

import com.notiprice.entity.Product
import com.notiprice.dao.ProductDao
import com.notiprice.dao.SubscriptionDao
import com.notiprice.entity.Subscription
import com.notiprice.scanner.getValueByXpath
import org.jsoup.Jsoup
import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional
import us.codecraft.xsoup.Xsoup
import java.net.URL

/**
 * Сервис для товаров.
 */
@Service
class ProductService(
    private val productDao: ProductDao,
    private val subscriptionDao: SubscriptionDao,
    private val userService: UserService
) {
    /**
     * Добавление товара. Функция является транзакционной.
     */
    @Transactional
    fun addProduct(product: Product, username: String): Product {

        if (product.url.contains("?")) {
            product.url = product.url.split("?").first()
        }
        // TODO: вынести название в subscriptions, добавлять, если нет продуктов с таким же
url и xpath
        product.lastCheck = System.currentTimeMillis()
        val savedProduct = productDao.save(product)

        // throws exception if it doesn't exist
        val user = userService.getUserByUsername(username)

        subscriptionDao.save(Subscriptions(user.chatId, savedProduct.id))

        return savedProduct
    }

    /**
     * Получение товара по идентификатору.
     */
    fun getProductById(id: Long): Product {
        return productDao.findByIdOrNull(id)
    }
}
```

```

        ?: throw IllegalArgumentException("No such element")//ToDo: write a norm mess
    }

    /**
     * Получение продуктов пользователя.
     */
    fun getAllUserProducts(username: String): List<Product> {
        return productDao.findAllUserProducts(username)
    }

    /**
     * Изменение данных о товаре.
     */
    fun updateProduct(product: Product) {
        val prevProduct = getProductById(product.id)
        product.lastCheck = prevProduct.lastCheck
        product.priceStr = prevProduct.priceStr
        productDao.update(product) //ToDo: throw ex there
    }

    /**
     * Удаление продукта.
     */
    fun deleteProduct(id: Long) {
        productDao.delete(id)
    }

    /**
     * Получает страницу по URL и выделяет элемент по xpath и возвращает страницу.
     */
    fun getHtmlWithHighlightedElement(url: String, xpath: String): String {
        val doc = Jsoup.connect(url).get()

        Xsoup.compile(xpath)
            .evaluate(doc)
            .elements.first()
            ?.attr("style", "background-color: #FFFF00")

        return doc.toString()
    }

    /**
     * Получение xpath продукта по URL. В базе данных ищутся xpath от базового домена URL.
     * Найденные xpath проверяются можно ли получить значение по этому xpath значение.
     * Если таких несколько, то выбирается самый популярный.
     */
    fun getProductXPathByUrl(urlString: String): String {

        require(urlString.isNotBlank() && urlString.isNotEmpty())
        val url = URL(urlString)
        val baseUrl = url.host
        val candidates = productDao.findXPathByUrl(baseUrl)
    }

```

```

        for (xpath in candidates) {

            if (getValueByXPath(urlString, xpath) != null) {
                return xpath
            }
        }
        return ""
    }
}

```

## 10.2 UserService

```

package com.notiprice.service

import com.notiprice.dao.UserDao
import com.notiprice.entity.User
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
import org.springframework.stereotype.Service

/**
 * Сервис для работы с пользователем.
 */
@Service
class UserService(private val userDao: UserDao) {
    /**
     * Для хеширования пароля.
     */
    private val passwordEncoder = BCryptPasswordEncoder()

    /**
     * Добавление пользователя.
     */
    fun addUser(user: User): User {

        user.password = passwordEncoder.encode(user.password)

        return userDao.save(user)
    }

    /**
     * Получение пользователя по идентификатору.
     */
    fun getProductById(id: Long): User {
        return userDao.findByIdOrNull(id) ?: throw IllegalArgumentException("No such element")//ToDo: write a norm mess
    }

    /**
     * Получение пользователя по пользовательскому имени.
     */
    fun getUserByUsername(username: String): User {
        return userDao.findByUsernameOrNull(username) ?: throw IllegalArgumentException("No such element")//ToDo: write a norm mess
    }
}

```

```

    /**
     * Проверяет пароль пользователя, если пароли совпадают, возвращает пользователя, если
     нет, то бросает исключение.
     */
    fun login(user: User): User {

        val userDb = getUserByUsername(user.username)

        if(!passwordEncoder.matches(user.password, userDb.password)) {

            throw IllegalArgumentException("Password is incorrect")
        }

        return userDb
    }

    /**
     * Изменение данных о пользователе.
     */
    fun updateUser(user: User) {
        userDao.update(user) //ToDo: throw ex there
    }

    /**
     * Удаление пользователя.
     */
    fun deleteProduct(id: Long) {
        userDao.delete(id)
    }
}

```

## Лист регистрации изменений

[illegible]