|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.05-01 12 01–1 |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.05-01 12 01-1 | | **«HSE COFFEE» - КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ ДЛЯ ЗНАКОМСТВ НА ANDROID & IOS**  **СЕРВЕРНЫЙ КЛИЕНТ**  **Текст программы**  **RU.17701729.04.05-01 12 01–1**  **Листов 48** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2021**

ОГЛАВЛЕНИЕ

[1. ТЕКСТ ПРОГРАММЫ 4](#_Toc71119090)

[1.1. AuthController.kt 4](#_Toc71119091)

[1.2. MeetConroller.kt 7](#_Toc71119092)

[1.3. UserConroller.kt 10](#_Toc71119093)

[1.4. CancelStatus.kt 13](#_Toc71119094)

[1.5. MeetStatus.kt 14](#_Toc71119095)

[1.6. UserStatus.kt 14](#_Toc71119096)

[1.7. JwtResponseWrapper.kt 14](#_Toc71119097)

[1.8. LoginWrapper.kt 14](#_Toc71119098)

[1.9. Degree.kt 15](#_Toc71119099)

[1.10. Faculty.kt 15](#_Toc71119100)

[1.11. Gender.kt 15](#_Toc71119101)

[1.12. ConfirmationCode.kt 16](#_Toc71119102)

[1.13. Contact.kt 17](#_Toc71119103)

[1.14. Meet.kt 17](#_Toc71119104)

[1.15. RefreshToken.kt 18](#_Toc71119105)

[1.16. Search.kt 19](#_Toc71119106)

[1.17. SearchParams.kt 20](#_Toc71119107)

[1.18. User.kt 20](#_Toc71119108)

[1.19. ConfirmationRepository.kt 22](#_Toc71119109)

[1.20. ImageStorageRepository.kt 22](#_Toc71119110)

[1.21. MeetRepository.kt 23](#_Toc71119111)

[1.22. RefreshTokenRepository.kt 23](#_Toc71119112)

[1.23. SearchRepository.kt 23](#_Toc71119113)

[1.24. UserRepository.kt 24](#_Toc71119114)

[1.25. AuthService.kt 24](#_Toc71119115)

[1.26. EmailService.kt 26](#_Toc71119116)

[1.27. ImageStorageService.kt 29](#_Toc71119117)

[1.28. JwtService.kt 31](#_Toc71119118)

[1.29. MeetService.kt 33](#_Toc71119119)

[1.30. RefreshTokenService.kt 38](#_Toc71119120)

[1.31. UserService.kt 39](#_Toc71119121)

[1.32. Run.kt 42](#_Toc71119122)

[1.33. EmailServiceTest.kt 42](#_Toc71119123)

[1.34. JwtServiceTest.kt 43](#_Toc71119124)

[1.35. MeetServiceTest.kt 44](#_Toc71119125)

[1.36. RefreshTokenServiceTest.kt 46](#_Toc71119126)

[1.37. Build.gradle.kts 47](#_Toc71119127)

# ТЕКСТ ПРОГРАММЫ

## AuthController.kt

package com.goga133.hsecoffee.controllers  
  
import com.goga133.hsecoffee.service.EmailService  
import com.goga133.hsecoffee.service.JwtService  
import com.goga133.hsecoffee.service.RefreshTokenService  
import com.goga133.hsecoffee.service.UserService  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.http.HttpStatus  
import org.springframework.http.ResponseEntity  
import org.springframework.stereotype.Controller  
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.ResponseStatus  
import java.util.\*  
  
  
*/\*\*  
 \* Контроллер для авторизации пользователя.  
 \* Авторизация может происходить только через код, высланный на Email с доменами @edu.hse.ru или @hse.ru.  
 \* @see com.goga133.hsecoffee.entity.ConfirmationCode  
 \* После подтверждения Email-кода, пользователю высылается пара JWT из access Token и refresh Token.  
 \* AccessToken - многоразовый токен, но кратковременный.  
 \* RefreshToken - одноразовый токен, но долгосрочный.  
 \* @see com.goga133.hsecoffee.entity.RefreshToken  
 \* Пользователь также может получить AccessToken, отправив свой RefreshToken и fingerprint.  
 \* fingerprint - некий отпечаток устройства, можно называть его идентификатором.  
 \*/*@Controller  
class AuthController {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(AuthController::class.*java*)  
  
 */\*\*  
 \* Сервис для работы с SMTP.  
 \*/* @Autowired  
 private val emailService: EmailService? = null  
  
 */\*\*  
 \* Сервис для работы с пользователями.  
 \*/* @Autowired  
 private val userService: UserService? = null  
  
 */\*\*  
 \* Сервис для работы с JWT.  
 \*/* @Autowired  
 private val jwtService: JwtService? = null  
  
 */\*\*  
 \* Сервис для работы с RefreshToken  
 \*/* @Autowired  
 private val refreshTokenService: RefreshTokenService? = null  
  
 */\*\*  
 \* Метод для отправки Email-кода на почту пользователя.  
 \* POST запрос по адресу /api/code.  
 \*  
 \*  
 \* @param email - email адрес пользователя.  
 \* @return HTTP-ответ с телом из описания ответа сервера.  
 \*/* @RequestMapping(value = ["/api/code"], method = [RequestMethod.*POST*])  
 fun sendCode(@RequestParam(value = "email") email: String): ResponseEntity<String> {  
 *// Проверка на валидность почты:* if (emailService?.isValidMail(email) != true) {  
 val message = "Некорректный домен почты."  
  
 logger.info("Код не был отправлен на $email. Причина: $message")  
 return ResponseEntity.status(HttpStatus.*BAD\_REQUEST*).body(message)  
 }  
  
 if (emailService.trySendCode(email)) {  
 logger.info("Код был выслан на $email.")  
 return ResponseEntity.ok("Код был выслан")  
 }  
  
 logger.debug("Ошибка. Не удалось отправить код на $email.")  
 *// Если не удалось отослать письмо:* return ResponseEntity.status(HttpStatus.*BAD\_REQUEST*).body("Не удалось отправить код")  
 }  
  
 */\*\*  
 \* Метод для подтверждения кода, присланного на почту пользоваля.  
 \* POST запрос по адресу /api/confirm.  
 \*  
 \*  
 \* @param email - email адрес пользователя.  
 \* @param code - код с почты.  
 \* @param fingerprint - уникальный идентификатор устройства.  
 \* @return HTTP-ответ с телом из JSON двух строковых полей (токенов) или описанием ошибки.  
 \*/* @RequestMapping(value = ["/api/confirm"], method = [RequestMethod.*POST*])  
 fun confirmCode(  
 @RequestParam email: String,  
 @RequestParam code: Int,  
 @RequestParam fingerprint: String  
 ): ResponseEntity<String> {  
 if (emailService?.isValidCode(email, code) == true) {  
 val user = userService?.getUserByEmailOrCreate(email) ?: return ResponseEntity(  
 "Incorrect email",  
 HttpStatus.*BAD\_REQUEST* ).*also* **{** logger.debug("Пользователь с email = $email не был найден.")  
 **}** return ResponseEntity.ok(  
 jwtService?.getJsonTokens(user, fingerprint).*also* **{** logger.info("Пользовать с email = $email подтвердил учётную запись.")  
 **}** ?: return ResponseEntity(  
 "Server error",  
 HttpStatus.*INTERNAL\_SERVER\_ERROR* ).*also* **{** logger.warn(  
 "Возникла серверная ошибка при получении токенов." +  
 "Email = $email; Code = $code; fingerprint = $fingerprint.\""  
 )  
 **}** )  
 }  
 logger.info("Код $code не является валидным для $email.")  
 return ResponseEntity.badRequest().body("Некорректный код или email.")  
 }  
  
 */\*\*  
 \* Метод для обновления пары Refresh - Access токенов.  
 \* POST запрос по адресу /api/refresh.  
 \*  
 \* @param email - email адрес пользователя.  
 \* @param refreshToken - Refresh Token пользовотеля, представляет из себя [UUID].  
 \* @param fingerprint - уникальный идентификатор устройства.  
 \* @return HTTP-ответ с телом из JSON двух строковых полей (токенов) или описанием ошибки.  
 \*/* @RequestMapping(value = ["/api/refresh"], method = [RequestMethod.*POST*])  
 @ResponseStatus(HttpStatus.*OK*)  
 fun refreshToken(  
 @RequestParam email: String,  
 @RequestParam refreshToken: UUID,  
 @RequestParam fingerprint: String  
 ): ResponseEntity<String> {  
 *// Если user == null, значит пользователя нет в БД, а значит операция невозможна.* val user = userService?.getUserByEmail(email)  
 ?: return ResponseEntity.badRequest()  
 .body("Пользователя с такой почтой не существует.").*also* **{** logger.debug("Пользователь с email = $email не найден.")  
 **}** if (refreshTokenService?.isValid(user, refreshToken, fingerprint) == true) {  
 return ResponseEntity.ok(  
 jwtService?.getJsonTokens(user, fingerprint).*also* **{** logger.info("Пользователь $user обновил RefreshToken.")  
 **}** ?: return ResponseEntity(  
 "Server error",  
 HttpStatus.*INTERNAL\_SERVER\_ERROR* ).*also* **{** logger.warn("Пользователю $user не удалось обновить RefreshToken.")  
 **}** )  
 }  
  
 logger.debug("Обновитель Refresh Token для user = $user невозможно. Данные некорректные.")  
  
 return ResponseEntity.badRequest().body("Невозможно обновить токен.")  
 }  
}

## MeetConroller.kt

package com.goga133.hsecoffee.controllers  
  
import com.fasterxml.jackson.module.kotlin.jacksonObjectMapper  
import com.goga133.hsecoffee.data.status.CancelStatus  
import com.goga133.hsecoffee.data.status.MeetStatus  
import com.goga133.hsecoffee.entity.Meet  
import com.goga133.hsecoffee.entity.SearchParams  
import com.goga133.hsecoffee.service.AuthService  
import com.goga133.hsecoffee.service.MeetService  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.http.HttpStatus  
import org.springframework.http.ResponseEntity  
import org.springframework.stereotype.Controller  
import org.springframework.web.bind.annotation.PathVariable  
import org.springframework.web.bind.annotation.RequestBody  
import org.springframework.web.bind.annotation.RequestMapping  
import org.springframework.web.bind.annotation.RequestMethod  
  
*/\*\*  
 \* Контроллер для управления встречами авторизованных пользователей.  
 \* Авторизованный пользователь может получать статус его текущей встречи, выполнять поиск или отменять его встречи  
 \* по параметрам, а также получать список завершённых встреч.  
 \*/*@Controller  
class MeetController {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(MeetController::class.java)  
  
 */\*\*  
 \* Сервис для работы с авторизацией  
 \*/* @Autowired  
 private val authService: AuthService? = null  
  
 */\*\*  
 \* Сервис для работы с встречами.  
 \*/* @Autowired  
 private val meetService: MeetService? = null  
  
 */\*\*  
 \* Метод для получения текущей встречи, если её нет возвращается неизвестная встреча с [MeetStatus.NONE]  
 \* GET запрос по адресу /api/meet/{token}, где token - access токен пользователя.  
 \*  
 \* @return HTTP-ответ с телом из JSON представления объекта встречи или описания ошибки.  
 \* @see com.goga133.hsecoffee.entity.Meet  
 \*/* @RequestMapping(value = ["/api/meet/{token}"], method = [RequestMethod.GET])  
 fun getMeet(@PathVariable("token") token: String): ResponseEntity<String>? {  
 val loginWrapper = authService?.logByToken(token)  
  
 if (loginWrapper?.isSuccessful() != true) {  
 return loginWrapper?.responseEntity  
 }  
  
 val user = loginWrapper.user!!  
  
  
 *// Получаем встречу, если получить не удалось - возвращаем ошибку.* val meet = meetService?.getMeet(user)  
  
 logger.info("Для user = $user встреча равна $meet")  
 *// Если встреча успешно получена - возращаем её JSON представление.* return ResponseEntity.ok(jacksonObjectMapper().writeValueAsString(meet))  
 }  
  
 */\*\*  
 \* Метод для начала поиска встречи. Если встреча была уже начата - возвращается [MeetStatus.ACTIVE],  
 \* если же встреча уже ищется - [MeetStatus.SEARCH], если произошла ошибка - [MeetStatus.ERROR]  
 \* GET запрос по адресу /api/meet/{token}, где token - access токен пользователя.  
 \*  
 \* @param searchParams - Body из JSON объекта SearchParams  
 \* @return HTTP-ответ с телом из MeetStatus или описания ошибки.  
 \* @see com.goga133.hsecoffee.entity.SearchParams  
 \* @see com.goga133.hsecoffee.entity.Meet  
 \*/* @RequestMapping(value = ["/api/search/{token}"], method = [RequestMethod.POST])  
 fun findMeet(  
 @PathVariable("token") token: String,  
 @RequestBody searchParams: SearchParams  
 ): ResponseEntity<String>? {  
 *// TODO: Добавить для пола* val loginWrapper = authService?.logByToken(token)  
  
 if (loginWrapper?.isSuccessful() != true) {  
 return loginWrapper?.responseEntity  
 }  
  
 val user = loginWrapper.user!!  
  
 val meetStatus = meetService?.searchMeet(user, searchParams)  
  
 if (meetStatus == MeetStatus.ERROR) {  
 return ResponseEntity("Возникла серверная ошибка", HttpStatus.INTERNAL\_SERVER\_ERROR)  
 }  
  
 logger.info("Для user = $user стастус встречи - $meetStatus")  
 return ResponseEntity.ok(meetStatus.toString())  
 }  
  
 */\*\*  
 \* Метод для прерывания поиска встречи.  
 \*  
 \* DELETE запрос по адресу /api/meet/{token}, где token - access токен пользователя.  
 \*  
 \* @return HTTP-ответ с телом из [MeetStatus]  
 \* @see com.goga133.hsecoffee.entity.Meet  
 \*/* @RequestMapping(value = ["/api/meet/{token}"], method = [RequestMethod.DELETE])  
 fun cancelSearch(@PathVariable("token") token: String): ResponseEntity<String>? {  
 val loginWrapper = authService?.logByToken(token)  
  
 if (loginWrapper?.isSuccessful() != true) {  
 return loginWrapper?.responseEntity  
 }  
  
 val user = loginWrapper.user!!  
  
 val cancelStatus = meetService?.cancelSearch(user);  
 logger.info("Отмена встречи для user = $user прозошла со статусом $cancelStatus.")  
  
 when (cancelStatus) {  
 CancelStatus.FAIL -> {  
 return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Невозможно отменить встречу.")  
 }  
 else -> {  
 meetService?.getMeet(user)?.meetStatus?.name?.let **{** return ResponseEntity.ok(it)  
 **}** return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Невозможно отменить встречу.")  
 }  
 }  
 }  
  
 */\*\*  
 \* Метод для получения списка из законченный встреч. Законченной встречей считается такая встреча,  
 \* у которой [MeetStatus.FINISHED]  
 \*  
 \* DELETE запрос по адресу /api/meet/{token}, где token - access токен пользователя.  
 \*  
 \* @return HTTP-ответ с телом из описания серверного ответа.  
 \* @see com.goga133.hsecoffee.entity.Meet  
 \*/* @RequestMapping(value = ["/api/meets/{token}"], method = [RequestMethod.GET])  
 fun getMeets(@PathVariable("token") token: String): ResponseEntity<String>? {  
 val loginWrapper = authService?.logByToken(token)  
  
 if (loginWrapper?.isSuccessful() != true) {  
 return loginWrapper?.responseEntity  
 }  
  
 val user = loginWrapper.user!!  
  
 *// Список встреч:* val meets = meetService?.getMeets(user)  
  
 logger.info("Встречи для пользователя user = $user => $meets")  
 return ResponseEntity.ok(jacksonObjectMapper().writeValueAsString(meets))  
 }  
}

## UserConroller.kt

package com.goga133.hsecoffee.controllers  
  
import com.fasterxml.jackson.module.kotlin.jacksonObjectMapper  
import com.goga133.hsecoffee.entity.Contact  
import com.goga133.hsecoffee.entity.User  
import com.goga133.hsecoffee.service.AuthService  
import com.goga133.hsecoffee.service.ImageStorageService  
import com.goga133.hsecoffee.service.UserService  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.http.HttpStatus  
import org.springframework.http.ResponseEntity  
import org.springframework.stereotype.Controller  
import org.springframework.web.bind.annotation.\*  
import org.springframework.web.multipart.MultipartFile  
import java.io.IOException  
  
  
*/\*\*  
 \* Контроллер для управления аккаунтом пользователя.  
 \* Пользователь может выставлять настройки, получать текущии, а также загружать фотографию.  
 \* Пользователь не может выставлять себе следующие поля: id, email, createdDate.  
 \*/*@Controller  
class UserController {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(UserController::class.java)  
  
 */\*\*  
 \* Сервис для работы с авторизацией  
 \*/* @Autowired  
 private val authService: AuthService? = null  
  
 */\*\*  
 \* Сервис для работы с пользователями.  
 \*/* @Autowired  
 private val userService: UserService? = null  
  
 */\*\*  
 \* Сервис для работы с загрузкой фотографий.  
 \*/* @Autowired  
 private val imageStorageService: ImageStorageService? = null  
  
  
 */\*\*  
 \* Метод для выставления настроек для пользователя.  
 \* Пользователь не может выставлять себе следующие поля: id, email, createdDate.  
 \* PUT запрос по адресу /api/user/settings/{token}, где token - access токен пользователя.  
 \*  
 \* @return HTTP-ответ с телом из JSON представления объекта пользователя или описания ошибки.  
 \* @see com.goga133.hsecoffee.entity.User  
 \*/* @RequestMapping(value = ["/api/user/settings/{token}"], method = [RequestMethod.PUT])  
 fun setSettings(@PathVariable("token") token: String, @RequestBody newUser: User): ResponseEntity<String>? {  
 val loginWrapper = authService?.logByToken(token)  
  
 if(loginWrapper?.isSuccessful() != true){  
 return loginWrapper?.responseEntity  
 }  
  
 if(userService?.setSettings(loginWrapper.user!!, newUser) == true){  
 logger.info("Пользователю $newUser обновлены настройки.")  
 return ResponseEntity.ok(jacksonObjectMapper().writeValueAsString(this))  
 }  
  
 logger.warn("С пользователем $newUser возникла ошибка при обновлении настроек.")  
  
 return ResponseEntity.badRequest().body("Возникла ошибка.")  
  
 }  
  
 */\*\*  
 \* Метод получения текущих настроек пользователя.  
 \* GET запрос по адресу /api/user/settings/{token}, где token - access токен пользователя.  
 \*  
 \* @return HTTP-ответ с телом из JSON представления объекта пользователя или описания ошибки.  
 \* @see com.goga133.hsecoffee.entity.User  
 \*/* @RequestMapping(value = ["/api/user/settings/{token}"], method = [RequestMethod.GET])  
 @ResponseStatus(HttpStatus.OK)  
 fun getSettings(@PathVariable("token") token: String): ResponseEntity<String>? {  
 val loginWrapper = authService?.logByToken(token)  
  
 if(loginWrapper?.isSuccessful() != true){  
 return loginWrapper?.responseEntity  
 }  
  
 val user = loginWrapper.user!!  
  
 logger.info("Выданы настройки для пользователя $user")  
 return ResponseEntity.ok(jacksonObjectMapper().writeValueAsString(user))  
 }  
  
 */\*\*  
 \* Метод для загрузки фотографии пользователя.  
 \* Максимальный объём выставлен в поле [properties.spring.servlet.multipart.max-file-size]  
 \*  
 \* POST по адресу /api/user/image/{token}, где token - access токен пользователя.  
 \*  
 \* @param image - фотография, загружаемая пользователем.  
 \* @return HTTP-ответ с телом из JSON представления объекта пользователя или описания ошибки.  
 \* @see com.goga133.hsecoffee.entity.User  
 \*/* @RequestMapping(value = ["/api/user/image/{token}"], method = [RequestMethod.POST])  
 fun setImage(  
 @PathVariable("token") token: String,  
 @RequestParam("image") image: MultipartFile  
 ): ResponseEntity<String>? {  
 val loginWrapper = authService?.logByToken(token)  
  
 if(loginWrapper?.isSuccessful() != true){  
 return loginWrapper?.responseEntity  
 }  
  
 val user = loginWrapper.user!!  
  
 if (imageStorageService?.correctFile(image) == false) {  
 logger.debug("Пользователь $user заслал некорректный файл.")  
 return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Некорректная фотография.")  
 }  
  
 try {  
 imageStorageService?.store(image, user)  
 } catch (ioException: IOException) {  
 logger.error("Произошла ошибка при работе с файловой системой.", ioException)  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Серверная ошибка.")  
 }  
  
 logger.info("Польватель $user успешно загрузил фотографию размером в ${image.size} байт.")  
 return ResponseEntity.ok("Успешно.")  
 }  
  
}

## CancelStatus.kt

package com.goga133.hsecoffee.data.status  
  
*/\*\*  
 \* Enum class. Перечисление статус-кодов для отмены встреч.  
 \*/*enum class CancelStatus {  
 *SUCCESS*, *FAIL*, *NOT\_ALLOWED*}

## MeetStatus.kt

package com.goga133.hsecoffee.data.status  
  
*/\*\*  
 \* Enum class. Перечисление статус-кодов для состояния встречи.  
 \*/*enum class MeetStatus {  
 ACTIVE, FINISHED, SEARCH, NONE, ERROR  
}

## UserStatus.kt

package com.goga133.hsecoffee.data.status  
  
*/\*\*  
 \* Enum class. Перечисление статус-кодов для состояния пользователя.  
 \*/*enum class UserStatus{  
 UNKNOWN, ACTIVE, DELETED, BANNED  
}

## JwtResponseWrapper.kt

package com.goga133.hsecoffee.data.wrappers  
  
import org.springframework.http.HttpStatus  
  
*/\*\*  
 \* Data class. Предназначается для JWT, чтобы отдать контроллеру уже готовый ответ в случае ошибки.  
 \*/*data class JwtResponseWrapper(val httpStatus: HttpStatus, val message: String, val email: String?)

## LoginWrapper.kt

package com.goga133.hsecoffee.data.wrappers  
  
import com.goga133.hsecoffee.entity.User  
import org.springframework.http.ResponseEntity  
  
*/\*\*  
 \* Data class. Обёртка над результатом ответа после выполнения авторизации.  
 \*/*data class LoginWrapper(val user : User? = null, val responseEntity: ResponseEntity<String>? = null){  
 fun isSuccessful(): Boolean {  
 return user != null  
 }  
}

## Degree.kt

package com.goga133.hsecoffee.data  
  
*/\*\*  
 \* Степень образования.  
 \*/*enum class Degree {  
 NONE,  
 BACHELOR,  
 MAGISTRACY,  
 SPECIALTY,  
 POSTGRADUATE  
}

## Faculty.kt

package com.goga133.hsecoffee.data  
  
*/\*\*  
 \* Enum class. Перечисление доступных факультетов НИУ ВШЭ.  
 \*/*enum class Faculty {  
 *NONE*,  
 *LYCEUM*,  
 *MATH*,  
 *ECONOMY*,  
 *ELECTRONIC*,  
 *COMPUTER*,  
 *BUSINESS*,  
 *LAWYER*,  
 *JURISPRUDENCE*,  
 *HUMANITARIAN*,  
 *SOCIAL*,  
 *MEDIA*,  
 *WORLD\_ECONOMY*,  
 *MIEF*,  
 *PHYSICS*,  
 *CITY*,  
 *CHEMICAL*,  
 *BIOLOGY*,  
 *GEOGRAPHY*,  
 *LANGUAGE*,  
 *STATISTIC*,  
 *BANK*}

## Gender.kt

package com.goga133.hsecoffee.data  
  
*/\*\*  
 \* Enum class. Пол пользователя.  
 \*/*enum class Gender {  
 *NONE*,  
 *MALE*,  
 *FEMALE*}

## ConfirmationCode.kt

package com.goga133.hsecoffee.entity  
  
import java.util.\*  
import javax.persistence.\*  
import kotlin.random.Random  
  
*/\*\*  
 \* Data-class. Email-код для подтверждения аккаунта.  
 \*/*@Entity  
data class ConfirmationCode(  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id")  
 val id: Long = 0,  
  
 @Column(name = "code")  
 val code: Int = Random.nextInt(MIN\_CODE, MAX\_CODE),  
  
 @Column(name = "created\_date")  
 @Temporal(TemporalType.TIMESTAMP)  
 val createdDate: Date = Date(),  
  
 @Column(name = "email")  
 val email: String? = null  
) {  
  
 companion object {  
 private const val MIN\_CODE = 100000  
 private const val MAX\_CODE = 1000000  
 }  
  
 constructor(email: String) : this(  
 email = email,  
 createdDate = Date(),  
 code = Random.nextInt(MIN\_CODE, MAX\_CODE)  
 ) {  
 *//* }  
  
  
 constructor(email: String, code: Int) : this(  
 email = email,  
 createdDate = Date(),  
 code = code  
 ) {  
 *//* }  
}

## Contact.kt

package com.goga133.hsecoffee.entity  
  
import javax.persistence.\*  
  
*/\*\*  
 \* Data-class. Контакт.  
 \*/*@Entity  
data class Contact(  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "contact\_id")  
 val id: Long = 0,  
  
 @ManyToOne(targetEntity = User::class, fetch = FetchType.EAGER, cascade = [CascadeType.ALL])  
 @JoinColumn(name = "user\_id")  
 var user: User? = null,  
 @Column(name = "name")  
 val name: String,  
 @Column(name = "value")  
 val value: String  
) {  
 constructor() : this(name = "", value = "") {  
  
 }  
}

## Meet.kt

package com.goga133.hsecoffee.entity  
  
import com.goga133.hsecoffee.data.status.MeetStatus  
import java.time.Duration  
import java.time.Instant  
import java.util.\*  
import javax.persistence.\*  
  
*/\*\*  
 \* Data-class. Встреча.  
 \*/*@Entity  
data class Meet(  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id")  
 val id: Long = 0,  
  
 @ManyToOne(targetEntity = User::class, fetch = FetchType.EAGER)  
 @JoinColumn(nullable = false, name = "user1\_id")  
 val user1: User?,  
  
 @ManyToOne(targetEntity = User::class, fetch = FetchType.EAGER)  
 @JoinColumn(nullable = false, name = "user2\_id")  
 val user2: User?,  
  
 @Column(name = "status")  
 @Enumerated(EnumType.STRING)  
 var meetStatus: MeetStatus = MeetStatus.NONE,  
  
 @Column(name = "created\_date")  
 @Temporal(TemporalType.TIMESTAMP)  
 val createdDate: Date = Date(),  
  
 @Temporal(TemporalType.TIMESTAMP)  
 val expiresDate: Date = Date.from(Instant.now().plus(Duration.ofMinutes(1)))  
) {  
 constructor() : this(user1 = null, user2 = null, meetStatus = MeetStatus.NONE)  
 constructor(user1: User?, meetStatus: MeetStatus) : this(user1 = user1, user2 = null, meetStatus = meetStatus)  
 constructor(user1: User, user2: User, meetStatus: MeetStatus) : this(  
 id = 0,  
 user1 = user1,  
 user2 = user2,  
 meetStatus = meetStatus  
 )  
}

## RefreshToken.kt

package com.goga133.hsecoffee.entity  
  
import java.time.Duration  
import java.time.Instant  
import java.util.\*  
import javax.persistence.\*  
  
*/\*\*  
 \* Data-class. Токен для обновления сессии. Длительность жизни: 60 дней.  
 \*/*@Entity  
data class RefreshToken(  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 val id: Long,  
  
 @OneToOne(targetEntity = User::class, fetch = FetchType.*EAGER*)  
 @JoinColumn(nullable = false, name = "user\_id")  
 val user: User,  
 @Column(name = "uuid")  
 val uuid: UUID = UUID.randomUUID(),  
 @Column(name = "fingerprint")  
 val fingerprint: String,  
 @Temporal(TemporalType.*TIMESTAMP*)  
 val expiresDate: Date = Date.from(Instant.now().plus(Duration.ofDays(60))),  
  
 @Temporal(TemporalType.*TIMESTAMP*)  
 val createdDate: Date = Date.from(Instant.now())  
) {  
 constructor(user: User, fingerprint: String) : this(  
 id = 0,  
 user = user,  
 fingerprint = fingerprint  
 )  
  
 constructor(email: String?) : this(user = User(email), fingerprint = "")  
  
 constructor() : this(User(null), "") {  
  
 }  
}

## Search.kt

package com.goga133.hsecoffee.entity  
  
import java.util.\*  
import javax.persistence.\*  
  
*/\*\*  
 \* Data-class. Поиск. Требуется для реализации поиска встреч.  
 \*/*@Entity  
data class Search(  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id")  
 val id: Long = 0,  
  
 @OneToOne(targetEntity = User::class, fetch = FetchType.EAGER)  
 @JoinColumn(nullable = false, name = "user\_id")  
 val finder: User,  
  
 @OneToOne(targetEntity = SearchParams::class, fetch = FetchType.EAGER, cascade = [CascadeType.ALL])  
 @JoinColumn(nullable = false, name = "search\_params")  
 var searchParams: SearchParams,  
  
 @Column(name = "created\_date")  
 @Temporal(TemporalType.TIMESTAMP)  
 val createdDate: Date = Date()  
) {  
 constructor(finder: User, searchParams: SearchParams) : this(0, finder, searchParams)  
 constructor() : this(User(null), SearchParams()) {  
  
 }  
}

## SearchParams.kt

package com.goga133.hsecoffee.entity  
  
import com.goga133.hsecoffee.data.Degree  
import com.goga133.hsecoffee.data.Faculty  
import com.goga133.hsecoffee.data.Gender  
import javax.persistence.\*  
  
*/\*\*  
 \* Data-class. Параметры поиска встреч.  
 \*/*@Entity  
data class SearchParams(  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id")  
 val id: Long = 0,  
  
 @Column(name = "genders")  
 @CollectionTable(name = "genders")  
 @ElementCollection(targetClass = Gender::class, fetch = FetchType.EAGER)  
 @Enumerated(EnumType.STRING)  
 val genders: MutableSet<Gender> = mutableSetOf(),  
  
 @Column(name = "min\_course")  
 val minCourse: Int = 1,  
 @Column(name = "max\_course")  
 val maxCourse: Int = 6,  
  
 @Column(name = "degrees")  
 @CollectionTable(name = "degrees")  
 @ElementCollection(targetClass = Degree::class, fetch = FetchType.EAGER)  
 @Enumerated(EnumType.STRING)  
 val degrees: MutableSet<Degree> = mutableSetOf(),  
  
 @Column(name = "faculties")  
 @CollectionTable(name = "faculties")  
 @ElementCollection(targetClass = Faculty::class, fetch = FetchType.EAGER)  
 @Enumerated(EnumType.STRING)  
 val faculties: MutableSet<Faculty> = mutableSetOf()  
)

## User.kt

package com.goga133.hsecoffee.entity  
  
import com.goga133.hsecoffee.data.Degree  
import com.goga133.hsecoffee.data.Faculty  
import com.goga133.hsecoffee.data.Gender  
import com.goga133.hsecoffee.data.status.UserStatus  
import java.util.\*  
import javax.persistence.\*  
import kotlin.collections.HashSet  
  
*/\*\*  
 \* Data-class. Пользователь.  
 \*/*@Entity  
data class User(  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "user\_id")  
 val id: Long = 0,  
  
 @Column(name = "email", nullable = false, unique = true)  
 var email: String = "",  
  
 @Column(name = "first\_name")  
 var firstName: String = "",  
  
 @Column(name = "last\_name")  
 var lastName: String = "",  
  
 @Column(name = "sex")  
 @Enumerated(EnumType.STRING)  
 var gender: Gender = Gender.NONE,  
  
 @Column(name = "created\_date")  
 @Temporal(TemporalType.TIMESTAMP)  
 val createdDate: Date = Date(),  
  
 @Enumerated(EnumType.STRING)  
 @Column(name = "faculty")  
 var faculty: Faculty = Faculty.NONE,  
  
 @Enumerated(EnumType.STRING)  
 @Column(name = "degree")  
 var degree: Degree = Degree.NONE,  
  
 @OneToMany(fetch = FetchType.EAGER, cascade = [CascadeType.ALL], targetEntity = Contact::class)  
 val contacts: MutableList<Contact> = ArrayList(),  
  
 @Column(name = "course")  
 var course: Int = 0,  
  
 @Column(name = "user\_status")  
 @Enumerated(EnumType.STRING)  
 var userStatus: UserStatus = UserStatus.UNKNOWN,  
  
 @Column(name = "about\_me")  
 var aboutMe: String = "",  
  
 @Column(name = "photo\_uri")  
 var photoUri: String = "uploads/default/${gender.name}.png"  
) {  
  
 constructor(email: String?) : this(  
 email = email.toString(),  
 userStatus = UserStatus.ACTIVE  
 )  
  
 constructor() : this(null)  
  
}

## ConfirmationRepository.kt

package com.goga133.hsecoffee.repository  
  
import com.goga133.hsecoffee.entity.ConfirmationCode  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories  
import org.springframework.data.repository.CrudRepository  
import org.springframework.stereotype.Repository  
  
*/\*\*  
 \* Интерфейс для описания операций для взаимодействия с таблицей хранения кодов для подтверждения.  
 \* @see ConfirmationCode  
 \*/*@EnableJpaRepositories  
@Repository("confirmationCode")  
interface ConfirmationCodeRepository : CrudRepository<ConfirmationCode, Long> {  
 fun removeConfirmationTokenByEmail(email: String)  
  
 fun existsByEmail(email: String): Boolean  
  
 fun findByEmail(email: String): ConfirmationCode?  
}

## ImageStorageRepository.kt

package com.goga133.hsecoffee.repository  
  
import com.goga133.hsecoffee.entity.User  
import org.springframework.web.multipart.MultipartFile  
  
*/\*\*  
 \* Интерфейс для описания операций для взаимодействия с изображениями пользователей.  
 \* @see com.goga133.hsecoffee.service.ImageStorageService  
 \*/*interface ImageStorageRepository {  
 */\*\*  
 \* Метод для проверки корректности файла.  
 \*/* fun correctFile(file: MultipartFile): Boolean  
  
 */\*\*  
 \* Метод для сохранения файла для юзера.  
 \*/* fun store(file: MultipartFile, user: User)  
}

## MeetRepository.kt

package com.goga133.hsecoffee.repository  
  
import com.goga133.hsecoffee.entity.Meet  
import com.goga133.hsecoffee.entity.User  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories  
import org.springframework.data.repository.CrudRepository  
import org.springframework.stereotype.Repository  
  
*/\*\*  
 \* Интерфейс для описания операций для взаимодействия с таблицей встреч.  
 \* @see Meet  
 \*/*@EnableJpaRepositories  
@Repository("meetRepository")  
interface MeetRepository : CrudRepository<Meet, Long> {  
 fun findAllByUser1OrUser2(user1: User, user2: User): List<Meet>?  
  
 fun existsMeetById(id: Long): Boolean  
}

## RefreshTokenRepository.kt

package com.goga133.hsecoffee.repository  
  
import com.goga133.hsecoffee.entity.RefreshToken  
import com.goga133.hsecoffee.entity.User  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories  
import org.springframework.data.repository.CrudRepository  
import org.springframework.stereotype.Repository  
  
*/\*\*  
 \* Интерфейс для описания операций для взаимодействия с таблицей для хранения Refresh токенов.  
 \* @see RefreshToken  
 \*/*@EnableJpaRepositories  
@Repository("refreshTokenRepository")  
interface RefreshTokenRepository : CrudRepository<RefreshToken, Long> {  
 fun findRefreshTokenByUser(user: User): RefreshToken?  
}

## SearchRepository.kt

package com.goga133.hsecoffee.repository  
  
import com.goga133.hsecoffee.entity.Search  
import com.goga133.hsecoffee.entity.User  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories  
import org.springframework.data.repository.CrudRepository  
import org.springframework.stereotype.Repository  
  
*/\*\*  
 \* Интерфейс для описания операций для взаимодействия с таблицей поиска встреч.  
 \* @see Search  
 \*/*@EnableJpaRepositories  
@Repository("searchRepository")  
interface SearchRepository : CrudRepository<Search, Long> {  
 fun findSearchByFinder(finder: User): Search?  
}

## UserRepository.kt

package com.goga133.hsecoffee.repository  
  
import com.goga133.hsecoffee.entity.User  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories  
import org.springframework.data.repository.CrudRepository  
import org.springframework.stereotype.Repository  
  
*/\*\*  
 \* Интерфейс для описания операций для взаимодействия с таблицей пользователей.  
 \* @see User  
 \*/*@EnableJpaRepositories  
@Repository("userRepository")  
interface UserRepository : CrudRepository<User, Long> {  
 fun findByEmail(email: String): User?  
  
 fun existsUserById(id: Long): Boolean  
}

## AuthService.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.data.wrappers.LoginWrapper  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.http.HttpStatus  
import org.springframework.http.ResponseEntity  
import org.springframework.stereotype.Service  
  
*/\*\*  
 \* Сервис для работы с авторизацией пользователей.  
 \*/*@Service  
class AuthService {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(AuthService::class.java)  
  
 */\*\*  
 \* Сервис для работы с JWT.  
 \*/* @Autowired  
 private val jwtService: JwtService? = null  
  
 */\*\*  
 \* Сервис для работы с пользователями.  
 \*/* @Autowired  
 private val userService: UserService? = null  
  
  
 */\*\*  
 \* Провести авторизацию по токену.  
 \*/* fun logByToken(token: String): LoginWrapper {  
 *// Если валидация прошла неуспешно - возвращаем код ошибки от валидатора:* val validator = jwtService?.validateToken(token).apply {  
 if (this?.httpStatus != HttpStatus.OK) {  
 logger.debug("Неверный токен: token = $token. Ошибка: ${this?.message}")  
 return LoginWrapper(  
 responseEntity = ResponseEntity.status(this?.httpStatus ?: HttpStatus.BAD\_REQUEST)  
 .body(this?.message)  
 )  
 }  
  
 }  
 *// Читаем Email после валидации:* val email: String =  
 validator?.email ?: return LoginWrapper(  
 responseEntity = ResponseEntity.status(HttpStatus.BAD\_REQUEST)  
 .body("Возникла ошибка при валидации токена.")  
 )  
  
 val user = userService?.getUserByEmail(email) ?: return LoginWrapper(  
 responseEntity = ResponseEntity.status(HttpStatus.BAD\_REQUEST)  
 .body("Возникла серверная ошибка")  
 )  
 .apply { logger.warn("При верном token = $token пользователя не существует.") }  
  
 return LoginWrapper(user = user)  
 }  
}

## EmailService.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.entity.ConfirmationCode  
import com.goga133.hsecoffee.repository.ConfirmationCodeRepository  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.beans.factory.annotation.Value  
import org.springframework.mail.SimpleMailMessage  
import org.springframework.mail.javamail.JavaMailSender  
import org.springframework.stereotype.Service  
import org.springframework.transaction.annotation.Transactional  
import java.time.Instant  
import java.util.\*  
  
*/\*\*  
 \* Сервис для работы с Email, в частности с протоколом SMTP для отправки EMAIL-сообщений.  
 \*  
 \* @param text - Текст письма.  
 \* @param subject - Тема письма.  
 \* @param from - Email-адресс отправителя.  
 \* @param lifeTime - Время действия кода в милисекундах.  
 \* @param domains - Разрешенные домены для почт получателя. Разделяются ';'.  
 \*/*@Service  
class EmailService(  
 private val javaMailSender: JavaMailSender,  
 @Value("\${mail.text}") private val text: String,  
 @Value("\${mail.subject}") private val subject: String,  
 @Value("\${mail.from}") private val from: String,  
 @Value("\${mail.lifetime.ms}") private var lifeTime: Int,  
 @Value("\${mail.domains}") private val domains: String  
) {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(EmailService::class.java)  
  
 */\*\*  
 \* Репозиторий для кодов подтверждения.  
 \*/* @Autowired  
 private val confirmationCodeRepository: ConfirmationCodeRepository? = null  
  
 */\*\*  
 \* Проверка на валидность кода относительно Email-адреса.  
 \* @param receiver - email-адрес.  
 \* @param code - код, отправленный на email-адрес.  
 \* @return true - если код верный, false - если код неверный или произошла ошибка.  
 \*/* fun isValidCode(receiver: String, code: Int): Boolean {  
 val confirmationCode = confirmationCodeRepository?.findByEmail(receiver) ?: return false  
  
 if (confirmationCode.code == code) {  
 if (Instant.now().minusMillis(confirmationCode.createdDate.time)  
 .toEpochMilli() <= lifeTime  
 ) {  
 logger.debug("Код $code для $receiver корректный.")  
 return true  
 }  
  
 logger.debug("Код $code для $receiver недействителен.")  
 return false  
 }  
  
 logger.debug("Код $code для $receiver некорректный.")  
 return false  
 }  
  
 */\*\*  
 \* Проверка на валидность Email-адреса.  
 \* Валидным Email-адресом считается такой, который содержит больше [MIN\_LENGTH\_LOGIN] символов.  
 \* @see MIN\_LENGTH\_LOGIN  
 \* @param email - email-адрес, проверяемый на корректность.  
 \* @return true - если адрес корректный, иначе - false.  
 \*/* fun isValidMail(email: String?): Boolean {  
 if (email.isNullOrEmpty())  
 return false  
  
 domains.split(";").forEach { domain ->  
 if (email.endsWith(domain) && email.length - domain.length > MIN\_LENGTH\_LOGIN) {  
 logger.debug("$email корректный.")  
 return true  
 }  
 }  
  
 logger.debug("$email некорректный.")  
 return false  
 }  
  
 */\*\*  
 \* Метод для генерации, создания и отправки кода на Email-адрес.  
 \* @param receiver - email-адрес, на который нужно отправить код.  
 \* @return True - если отправка успешна, False - иначе.  
 \*/* @Transactional  
 fun trySendCode(receiver: String): Boolean {  
 try {  
 *// Если код существует:* if (confirmationCodeRepository?.existsByEmail(receiver) == true) {  
 val confirmation = confirmationCodeRepository.findByEmail(receiver)  
 logger.debug("Для email = $receiver был найден $confirmation")  
  
 *// Проверка на время:* val delta = (Date.from(confirmation?.createdDate?.time?.let { Instant.now().minusMillis(it) }))  
  
 if (delta.time > lifeTime) {  
 logger.debug("Для email = $receiver срок действия предыщего кода вышел, будет сгенерирован новый.")  
 confirmationCodeRepository.apply {  
 removeConfirmationTokenByEmail(receiver)  
 save(ConfirmationCode(receiver))  
 }  
 }  
 } else {  
 with(ConfirmationCode(receiver)) {  
 logger.debug("Для email = $receiver был сгененирован $this")  
 confirmationCodeRepository?.save(this)  
 }  
 }  
  
 confirmationCodeRepository?.findByEmail(receiver)?.code?.let {  
 sendCode(receiver, it)  
 }  
  
 return true  
  
 } catch (e: Exception) {  
 logger.error("Произошла ошибка при отправке кода подтверждения", e)  
 return false  
 }  
 }  
  
 */\*\*  
 \* Ручное выставление времени. Используется для Unit-тестов.  
 \* @param lifeTime - время действия кода в милисекундах.  
 \*/* fun setLifeTime(lifeTime: Int) {  
 this.lifeTime = lifeTime  
 }  
  
 */\*\*  
 \* Метод для отправки целочисленного кода на Email-адрес.  
 \* @param receiver - email - получатель кода.  
 \* @param code - целочисленный код.  
 \*/* private fun sendCode(receiver: String, code: Int) {  
 val message = SimpleMailMessage().apply {  
 setSubject(subject!!)  
 setText(text!!.replace("{code}", code.toString()))  
 setFrom(from!!)  
 setTo(receiver)  
 }  
  
 javaMailSender.send(message)  
  
 logger.debug("На почту $receiver был отправлен следующий код подтверждения: $code")  
 }  
  
 companion object {  
 */\*\*  
 \* Минимальная длина логика у EMAIL адреса. Логином считается первая часть, разделённая @.  
 \* Например: abcd@edu.hse.ru => логин abcd.  
 \*/* const val MIN\_LENGTH\_LOGIN = 3  
 }  
}

## ImageStorageService.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.entity.User  
import com.goga133.hsecoffee.repository.ImageStorageRepository  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.stereotype.Service  
import org.springframework.web.multipart.MultipartFile  
import java.io.IOException  
import java.nio.file.Files  
import java.nio.file.Path  
import java.nio.file.Paths  
  
*/\*\*  
 \* Сервис для загрузки изображений для пользователя.  
 \* Изображения хранятся в root/uploads, где root - корневая папка.  
 \* Название изображения - email.png, где email - электронная почта пользователя.  
 \* Реализует ImageStorageRepository  
 \* @see ImageStorageRepository  
 \*/*@Service  
class ImageStorageService : ImageStorageRepository {  
 @Autowired  
 private val userService: UserService? = null  
  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(ImageStorageService::class.java)  
  
 */\*\*  
 \* Метод для проверки корректности изображения.  
 \* Изображение должно быть формата image типа png, jpg или jpeg.  
 \*/* override fun correctFile(file: MultipartFile): Boolean {  
 if (file.isEmpty || file.originalFilename.isNullOrEmpty()) {  
 return false  
 }  
  
 logger.debug("Название полученного файла: ${file.originalFilename}")  
  
 if (!isSupportedContentType(file.originalFilename!!)) {  
 logger.debug("Тип файла не является фотографией.")  
 return false  
 }  
  
 logger.debug("Изображение корректное")  
 return true  
 }  
  
 */\*\*  
 \* Метод для провеки корректности типа файла.  
 \*/* private fun isSupportedContentType(contentType: String): Boolean {  
 return contentType.endsWith("png") ||  
 contentType.endsWith("jpg") ||  
 contentType.endsWith("jpeg")  
 }  
  
 */\*\*  
 \* Метод для загрузки фотографии в корневую папку.  
 \* Если фотография уже существовала, она удалится.  
 \*/* override fun store(file: MultipartFile, user: User) {  
 try {  
 *// Удаляем если существует:* Files.deleteIfExists(rootLocation.resolve(user.email + ".png"))  
  
 *// Помещаем в память:* Files.copy(file.inputStream, rootLocation.resolve(user.email + ".png"))  
  
 logger.debug("Фотография для пользователя $user была успешно помещена на сервер.")  
 *// Меняем у пользователя:* userService?.changeFolderAndSave(user)  
 } catch (io: IOException) {  
 logger.error("Произошла ошибка при работе с файлами.", io)  
 }  
 }  
  
 companion object {  
 */\*\*  
 \* Корневая папка для хранения изображений.  
 \*/* private val rootLocation: Path = Paths.get("uploads")  
  
 */\*\*  
 \* Метод для создания корневой папки.  
 \*/* fun init() {  
 if (!Files.exists(rootLocation)) {  
 Files.createDirectory(rootLocation)  
 }  
 }  
 }  
}

## JwtService.kt

package com.goga133.hsecoffee.service  
  
import com.fasterxml.jackson.module.kotlin.jacksonObjectMapper  
import com.goga133.hsecoffee.data.wrappers.JwtResponseWrapper  
import com.goga133.hsecoffee.entity.User  
import io.jsonwebtoken.\*  
import io.jsonwebtoken.security.Keys  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.beans.factory.annotation.Value  
import org.springframework.http.HttpStatus  
import org.springframework.stereotype.Service  
import java.time.Duration  
import java.time.Instant  
import java.util.\*  
  
*/\*\*  
 \* Сервис для работы с Javascript Web Token.  
 \*/*@Service  
class JwtService(  
 @Value("\${jwt.key}") val secretKey: String,  
 @Value("\${jwt.access.min}") val minutes: Long  
) {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(JwtService::class.java)  
  
 */\*\*  
 \* Сервис работы с Refresh токенами.  
 \*/* @Autowired  
 val refreshTokenService: RefreshTokenService? = null  
  
 */\*\*  
 \* Метод для создания JWT токена.  
 \* Алгоритм: [SignatureAlgorithm.HS512]  
 \* Время действия: [minutes]  
 \*  
 \* @param user - Пользователь, относительно которого генерируется токен.  
 \* @return Токен в [String] представлении.  
 \*/* fun createAccessToken(user: User): String {  
 return Jwts.builder()  
 .signWith(Keys.hmacShaKeyFor(secretKey.toByteArray()), SignatureAlgorithm.HS512)  
 .setSubject(user.email)  
 .setIssuer(ISSUER)  
 .setExpiration(Date.from(Instant.now().plus(Duration.ofMinutes(minutes))))  
 .setIssuedAt(Date.from(Instant.now()))  
 .compact()  
 }  
  
 */\*\*  
 \* Метод для валидации токена средствами [Jwts.parserBuilder]  
 \* @param token - JWT токен.  
 \*/* fun validateToken(token: String): JwtResponseWrapper {  
 try {  
 val claim = parseAccessToken(token)  
  
 if (claim.body?.subject == null)  
 return getIncorrectTokenResponse()  
 } catch (e: ExpiredJwtException) {  
 logger.debug("Срок действие токена $token истекло.")  
 return JwtResponseWrapper(HttpStatus.FORBIDDEN, "The token has expired", null)  
 } catch (e: JwtException) {  
 return getIncorrectTokenResponse()  
 }  
  
 logger.debug("Токен $token успешно прошёл валидацию.")  
 return JwtResponseWrapper(  
 HttpStatus.OK,  
 "Correct token",  
 parseAccessToken(token).body?.subject ?: return getIncorrectTokenResponse()  
 )  
 }  
  
 private fun getIncorrectTokenResponse(): JwtResponseWrapper {  
 return JwtResponseWrapper(HttpStatus.UNAUTHORIZED, "Incorrect token", null)  
 }  
  
 */\*\*  
 \* Преобразование токена из [String] в [Jws] спомощью [Jwts.parserBuilder]  
 \* @see Jwts.parserBuilder  
 \* @see Jws  
 \*/* fun parseAccessToken(token: String): Jws<Claims> {  
 return Jwts.parserBuilder()  
 .setSigningKey(Keys.hmacShaKeyFor(secretKey.toByteArray()))  
 .build()  
 .parseClaimsJws(token)  
 }  
  
 */\*\*  
 \* Получение на основе пользователя и отпечатка пару из Access и Refresh токенов в JSON формате.  
 \* @param user - Пользователь, для которого генерируются токены.  
 \* @param fingerPrint - Отпечаток для валидации устройства пользователя.  
 \* @return JSON с полями accessToken и refreshToken  
 \*/* fun getJsonTokens(user: User, fingerPrint: String): String {  
 val accessToken = createAccessToken(user)  
 val refreshToken = refreshTokenService?.createByUser(user, fingerPrint)?.uuid  
  
 logger.debug("Для $user получены access и refresh токены.")  
  
 return jacksonObjectMapper().writeValueAsString(  
 mapOf<String, Any?>(  
 "accessToken" to accessToken,  
 "refreshToken" to refreshToken  
 )  
 )  
 }  
  
 companion object {  
 */\*\*  
 \* Подпись для токена.  
 \*/* private const val ISSUER = "HSE Coffee"  
 }  
}

## MeetService.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.data.status.CancelStatus  
import com.goga133.hsecoffee.entity.Meet  
import com.goga133.hsecoffee.entity.Search  
import com.goga133.hsecoffee.entity.SearchParams  
import com.goga133.hsecoffee.entity.User  
import com.goga133.hsecoffee.data.status.MeetStatus  
import com.goga133.hsecoffee.repository.MeetRepository  
import com.goga133.hsecoffee.repository.SearchRepository  
import com.goga133.hsecoffee.repository.UserRepository  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.beans.factory.annotation.Qualifier  
import org.springframework.stereotype.Service  
import java.util.\*  
import javax.persistence.Transient  
  
*/\*\*  
 \* Сервис для работы с встречами.  
 \*/*@Service  
class MeetService {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(MeetService::class.java)  
  
 */\*\*  
 \* Репозиторий встреч.  
 \*/* @Qualifier("meetRepository")  
 @Autowired  
 private val meetRepository: MeetRepository? = null  
  
 */\*\*  
 \* Репозиторий пользователей.  
 \*/* @Qualifier("userRepository")  
 @Autowired  
 private val userRepository: UserRepository? = null  
  
 */\*\*  
 \* Репозиторий поиска.  
 \*/* @Qualifier("searchRepository")  
 @Autowired  
 private val searchRepository: SearchRepository? = null  
  
 */\*\*  
 \* Метод для получения текущей встречи пользователя.  
 \* Если у пользователя нет встречи, то вернётся пустая встреча с [MeetStatus.NONE]  
 \* @param user - пользователь, относительно которого произоводится поиск текущей встречи.  
 \*/* fun getMeet(user: User): Meet {  
 *// Пользователь может искать, может быть уже в встрече, либо ничего.  
 // Если он ищет - значит в серчрепе что-то найдётся.  
 // Если он не ищет, значит либо встречается, либо ничего.  
 // Тогда проверим на статус последней встречи, если встречи нет или статус финешед - значит он свободен  
 // иначе - он встречается.* if (userRepository == null || !userRepository.existsUserById(user.id)) {  
 logger.warn("Пользователь $user не найден в базе данных.")  
 return Meet()  
 }  
  
 *// Мы ищем среди всех встреч* val meet = meetRepository?.findAllByUser1OrUser2(user, user)?.maxByOrNull **{** it.expiresDate **}** if (searchRepository?.findSearchByFinder(user) != null) {  
 return Meet(user, MeetStatus.SEARCH)  
 } else if (meet != null) {  
 updateMeetStatus(meet)  
  
 if (meet.meetStatus == MeetStatus.FINISHED) {  
 return Meet()  
 }  
  
 return meet  
 }  
  
 return Meet()  
 }  
  
 */\*\*  
 \* Метод, который дыдаёт коллекцию из всех законченных встреч пользователя.  
 \* Встреча считается законченной, если у неё [MeetStatus] = [MeetStatus.FINISHED]  
 \*  
 \* @param user - пользователь, относительно которого произоводится поиск законченных встреч.  
 \* @return коллекция встреч  
 \* @see Meet  
 \*/* fun getMeets(user: User): Collection<Meet> {  
 if (userRepository == null || !userRepository.existsUserById(user.id)) {  
 return arrayListOf()  
 }  
  
 return meetRepository?.findAll()?.filter **{** it **->** (it.user1 == user || it.user2 == user) && it.apply **{** updateMeetStatus(it) **}**.meetStatus == MeetStatus.FINISHED  
 **}** ?: listOf()  
 }  
  
 */\*\*  
 \* Метод для отмены пользователем встречи.  
 \* @param user - пользователь, относительно которого производится отмена встречи.  
 \* @return [CancelStatus.SUCCESS] - если отмена успешна, [CancelStatus.FAIL] - если отмена неудачна.  
 \* @see CancelStatus  
 \*/* @Transient  
 fun cancelSearch(user: User): CancelStatus {  
 if (userRepository == null || !userRepository.existsUserById(user.id)) {  
 logger.warn("Пользователь $user не найден в базе данных.")  
 return CancelStatus.FAIL  
 }  
  
 val finderSearch = searchRepository?.findSearchByFinder(user)  
  
 if (finderSearch != null) {  
 searchRepository?.delete(finderSearch)  
  
 logger.debug("Пользователь $user отменил встречу.")  
 return CancelStatus.SUCCESS  
 }  
  
 return CancelStatus.NOT\_ALLOWED  
 }  
  
 */\*\*  
 \* Метод для поиска встречи.  
 \*  
 \*  
 \* @param user - пользователь, который производит поиск встречи.  
 \* @param searchParams - параметры поиска, с которыми производится поиск встречи пользователем.  
 \*  
 \* @return [MeetStatus.ERROR] - если во время посиска произошла ошибка,  
 \* [MeetStatus.ACTIVE] - если встреча уже идёт или она была найдена.  
 \* [MeetStatus.SEARCH] - если встреча не была найдена и ведётся поиск встречи.  
 \*  
 \* @see MeetStatus  
 \* @see SearchParams  
 \*/* @Transient  
 fun searchMeet(user: User, searchParams: SearchParams): MeetStatus {  
 if (userRepository == null || !userRepository.existsUserById(user.id) || meetRepository == null || searchRepository == null) {  
 return MeetStatus.ERROR  
 }  
  
 val meet = meetRepository.findAllByUser1OrUser2(user, user)?.maxByOrNull **{** it.expiresDate **}**.apply **{** this?.let **{** updateMeetStatus(it)  
 **}  
 }** if (meet?.meetStatus == MeetStatus.ACTIVE)  
 return MeetStatus.ACTIVE  
  
 *// Если его нет в доске поиска:* if (searchRepository.findSearchByFinder(user) == null) {  
 val searches = searchRepository.findAll()  
  
 val finder = searches.firstOrNull **{** CheckerSearch(user, searchParams, it.finder, it.searchParams).check()  
 **}** *// Если поиск неудачен, то добавляем в зал ожидания.* return if (finder?.finder == null) {  
 searchRepository.save(Search(user, searchParams))  
  
 MeetStatus.SEARCH  
 }  
 *// Если поиск удачен - делаем встречу.* else {  
 searchRepository.delete(finder)  
  
 meetRepository.save(Meet(user, finder.finder, MeetStatus.ACTIVE))  
  
 MeetStatus.ACTIVE  
 }  
 }  
 return MeetStatus.SEARCH  
 }  
  
 */\*\*  
 \* Метод для проверки законченности встречи. Если время действия встречи [Meet.expiresDate] вышло,  
 \* то она перейдёт в статус [MeetStatus.FINISHED]  
 \*  
 \* @see Meet  
 \*/* private fun updateMeetStatus(meet: Meet) {  
 if (meetRepository == null || !meetRepository.existsMeetById(meet.id)) {  
 return  
 }  
  
 if (meet.expiresDate.before(Date()) || meet.meetStatus == MeetStatus.FINISHED) {  
 meet.meetStatus = MeetStatus.FINISHED  
  
 logger.debug("Встреча $meet закончена.")  
 }  
  
 meetRepository.save(meet)  
 }  
  
 */\*\*  
 \* Класс для проверки поисковых интересов двух пользователей.  
 \* @param user1 - первый пользователь.  
 \* @param params1 - поисковые параметры первого пользователя.  
 \* @param user2 - второй пользователь.  
 \* @param params2 - поисковые параметры второго пользователя.  
 \*/* private inner class CheckerSearch(  
 val user1: User,  
 val params1: SearchParams,  
 val user2: User,  
 val params2: SearchParams  
 ) {  
 */\*\*  
 \* Метод для проверки годности поисковых запросов двух пользователей.  
 \*/* fun check(): Boolean {  
 return checkFaculties() && checkGenders() && checkDegrees() && checkCourses()  
 }  
  
 */\*\*  
 \* Метод для проверки факультетов  
 \*/* private fun checkFaculties(): Boolean {  
 return params1.faculties.contains(user2.faculty) &&  
 params2.faculties.contains(user1.faculty)  
 }  
  
 */\*\*  
 \* Метод для проверки полов.  
 \*/* private fun checkGenders(): Boolean {  
 return params1.genders.contains(user2.gender) &&  
 params2.genders.contains(user1.gender)  
 }  
  
 */\*\*  
 \* Метод для проверки степени обучения.  
 \*/* private fun checkDegrees(): Boolean {  
 return params1.degrees.contains(user2.degree) &&  
 params2.degrees.contains(user1.degree)  
 }  
  
 */\*\*  
 \* Метод для проверки курсов.  
 \*/* private fun checkCourses(): Boolean {  
 return params1.minCourse <= user2.course &&  
 params1.maxCourse >= user2.course &&  
 params2.minCourse <= user1.course &&  
 params2.maxCourse >= user1.course  
 }  
 }  
}

## RefreshTokenService.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.entity.RefreshToken  
import com.goga133.hsecoffee.entity.User  
import com.goga133.hsecoffee.repository.RefreshTokenRepository  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.beans.factory.annotation.Qualifier  
import org.springframework.stereotype.Service  
import org.springframework.transaction.annotation.Transactional  
import java.util.\*  
  
*/\*\*  
 \* Сервис для работы с Refresh токенами.  
 \*/*@Service  
class RefreshTokenService {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(RefreshTokenService::class.java)  
  
 */\*\*  
 \* Репозиторий для работы с Refresh токенами.  
 \*/* @Qualifier("refreshTokenRepository")  
 @Autowired  
 private val refreshTokenRepository: RefreshTokenRepository? = null  
  
 */\*\*  
 \* Создание Refresh токена [RefreshToken] на основе пользователя и отпечатка.  
 \* Отпечаток используется как уникальный идентификатор устройства.  
 \*  
 \* @param user - пользователь, относительно которого создаётся токен.  
 \* @param fingerprint - отпечаток, т.е идентификатор устройства.  
 \*/* @Transactional  
 fun createByUser(user: User, fingerprint: String): RefreshToken {  
 *// Если существует:* val refreshToken = RefreshToken(user, fingerprint)  
  
 val refreshTokenRep = refreshTokenRepository?.findRefreshTokenByUser(user)  
 if (refreshTokenRep != null) {  
 logger.debug("Токен $refreshTokenRep удалён для $user")  
 refreshTokenRepository?.delete(refreshTokenRep)  
 }  
  
 refreshTokenRepository?.save(refreshToken)  
  
 logger.debug("Для $user был создан $refreshToken")  
 return refreshToken  
 }  
  
 */\*\*  
 \* Проверка на корректность Refresh токена [RefreshToken].  
 \*  
 \* @return True - если Refresh token верный, иначе - False.  
 \*/* fun isValid(user: User, token: UUID, fingerprint: String): Boolean {  
 val refreshToken = refreshTokenRepository?.findRefreshTokenByUser(user) ?: return false  
  
 if (refreshToken.uuid == token &&  
 refreshToken.expiresDate.after(Date()) &&  
 refreshToken.fingerprint == fingerprint  
 ) {  
 logger.debug("Refresh token $token для $user корректный.")  
 return true  
 }  
  
 logger.debug("Refresh token $token для $user неверный.")  
 return false  
 }  
}

## UserService.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.entity.Contact  
import com.goga133.hsecoffee.entity.User  
import com.goga133.hsecoffee.repository.UserRepository  
import org.slf4j.Logger  
import org.slf4j.LoggerFactory  
import org.springframework.beans.BeanUtils  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.beans.factory.annotation.Qualifier  
import org.springframework.beans.factory.annotation.Value  
import org.springframework.stereotype.Service  
import java.net.InetAddress  
  
*/\*\*  
 \* Сервис для работы с пользователями.  
 \*/*@Service  
class UserService(  
 @Value("\${storage.folder}")  
 private val folder: String  
) {  
 */\*\*  
 \* Логгер.  
 \*/* private val logger: Logger = LoggerFactory.getLogger(UserService::class.*java*)  
  
 */\*\*  
 \* Репозиторий для работы с пользователями.  
 \*/* @Qualifier("userRepository")  
 @Autowired  
 private val userRepository: UserRepository? = null  
  
 */\*\*  
 \* Метод для получения пользователя по email. Если такого пользователя нет в базе данных,  
 \* то он создаётся и сохраняется в ней.  
 \*  
 \* @param email - email адрес пользователя.  
 \* @return пользователь с данным email.  
 \*/* fun getUserByEmailOrCreate(email: String): User? {  
 var user = getUserByEmail(email)  
  
 if (user == null) {  
 user = createUserByEmail(email)  
 logger.debug("Пользователь $user успешно создан.")  
 }  
  
 return user  
 }  
  
 */\*\*  
 \* Метод для поиска пользователей с помощью email адреса.  
 \* Если такого пользователя не нашлось, возвращается null.  
 \*  
 \* @param email - email адрес.  
 \* @return [User], [User.email] которого равен [email], или null, если пользователя не нашлось.  
 \*/* fun getUserByEmail(email: String): User? {  
 val user = userRepository?.findByEmail(email = email)  
  
 if (user == null)  
 logger.debug("Неизвестный $user для email = $email")  
 return user  
 }  
  
 */\*\*  
 \* Создание в БД пользователя с email = [email].  
 \*  
 \* @param email - email, для которого нужно создать пользователя.  
 \*/* fun createUserByEmail(email: String): User {  
 val user = User(email)  
  
 userRepository?.save(user)  
 return user  
 }  
  
 */\*\*  
 \* Метод для смена у пользователя пути с его фотографией.  
 \* Фотография пользователя всегда находится по пути uploads/{email}.png.  
 \* @param user - пользователь, для которого нужно поменять путь его фотографии.  
 \*/* fun changeFolderAndSave(user: User) {  
 *with*(user) **{** photoUri = "${folder}${user.email}.png"  
 save(this)  
  
 logger.debug("Пользователю $user был присвоен путь для фотографии.")  
 **}** }  
  
 */\*\*  
 \* Установить настройки пользователю  
 \*/* fun setSettings(oldUser: User, newUser: User): Boolean {  
 try {  
 *// Копируем текущего пользователя в пользователя БД:* BeanUtils.copyProperties(  
 newUser, oldUser,  
 "id", "createdDate", "email", "photoUri", "userStatus", "contacts"  
 )  
  
 oldUser.contacts.*apply* **{** clear()  
 addAll(newUser.contacts.*map* **{** Contact(0, null, **it**.name, **it**.value)  
 **}**)  
 **}** *// Сохраняем* save(oldUser)  
  
 } catch (e: Exception) {  
 logger.error(e.message)  
 return false  
 }  
  
 return true  
 }  
  
 */\*\*  
 \* Сохранение или обновление записи о пользователе в БД.  
 \* @param user - пользователь, чью учётную запись необходимо обновить/сохранить в БД.  
 \*/* fun save(user: User) {  
 userRepository?.save(user)  
 }  
}

## Run.kt

package com.goga133.hsecoffee  
  
import com.goga133.hsecoffee.service.ImageStorageService  
import org.springframework.boot.SpringApplication  
  
*/\*\*  
 \* Мейн метод для запуска Spring Boot приложения.  
 \*/*fun main(args: Array<String>) {  
 SpringApplication.run(HseCoffeeApplication::class.java, \*args)  
  
 ImageStorageService.init()  
}

## EmailServiceTest.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.HseCoffeeApplication  
import com.goga133.hsecoffee.entity.ConfirmationCode  
import com.goga133.hsecoffee.repository.ConfirmationCodeRepository  
import org.junit.jupiter.api.Assertions.assertFalse  
import org.junit.jupiter.api.Assertions.assertTrue  
  
import org.junit.jupiter.api.Test  
import org.junit.runner.RunWith  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase  
import org.springframework.boot.test.context.SpringBootTest  
import org.springframework.test.context.ContextConfiguration  
import org.springframework.test.context.junit4.SpringRunner  
import java.util.concurrent.TimeUnit  
import kotlin.random.Random  
  
  
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)  
@RunWith(SpringRunner::class)  
@SpringBootTest  
@ContextConfiguration(classes = [HseCoffeeApplication::class])  
internal class EmailServiceTest {  
 @Autowired  
 val confirmationCodeRepository: ConfirmationCodeRepository? = null  
  
 @Autowired  
 val emailService: EmailService? = null  
  
 @Test  
 fun is\_correct\_validator\_by\_exist() {  
 val mockCode: Int = Random.nextInt()  
 emailService?.setLifeTime(10000000)  
 assertFalse(emailService!!.isValidCode("test@test$mockCode", mockCode))  
 confirmationCodeRepository?.save(ConfirmationCode("test@test$mockCode", mockCode))  
  
 assertTrue(emailService!!.isValidCode("test@test$mockCode", mockCode))  
 }  
  
 @Test  
 fun is\_correct\_validator\_by\_time() {  
 val mockCode: Int = Random.nextInt()  
  
 assertFalse(emailService!!.isValidCode("test@test$mockCode", mockCode))  
  
 confirmationCodeRepository?.save(ConfirmationCode("test@test$mockCode", mockCode))  
  
 emailService?.setLifeTime(10)  
 TimeUnit.SECONDS.sleep(1)  
 assertFalse(emailService!!.isValidCode("test@test$mockCode", mockCode))  
 }  
}

## JwtServiceTest.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.entity.User  
import org.junit.Assert.assertEquals  
import org.junit.Assert.assertNotEquals  
import org.junit.jupiter.api.Assertions.assertDoesNotThrow  
import org.junit.jupiter.api.Assertions.assertThrows  
import org.junit.jupiter.api.Test  
import org.junit.runner.RunWith  
import org.springframework.boot.test.context.SpringBootTest  
import org.springframework.test.context.junit4.SpringRunner  
  
@RunWith(SpringRunner::class)  
@SpringBootTest  
internal class JwtServiceTest {  
  
 private val mockSecretKey : String = "rXDm2ZfPA8kvi+luLS2+mYTOPzCr+FUC/SLGpehOtgd4TZstuH6COLA2aaUhY2HcoXj8lolkekTES/A6EYUQHA=="  
 private val jwtService = JwtService(mockSecretKey, 60)  
  
 @Test  
 fun createJwt() {  
 val mockUser = User("test@test")  
 val key = jwtService.createAccessToken(mockUser)  
  
 assertEquals(key.split(".")[0], "eyJhbGciOiJIUzUxMiJ9")  
 assertNotEquals(  
 key.split(".")[1],  
 "eyJzdWIiOiJ0ZXN0QHRlc3QiLCJpc3MiOiJIU0UgQ29mZmVlIiwiZXhwIjoxNjA4MTE1MTA2LCJpYXQiOjE2MDgxMTMzMDZ9"  
 )  
  
 assertDoesNotThrow {  
 val valid = jwtService.parseAccessToken(key)  
  
 assertEquals(valid.body.subject, "test@test")  
 }  
 }  
  
 @Test  
 fun validateJwt() {  
 val mockUser = User("test@test")  
 val key = jwtService.createAccessToken(mockUser)  
  
 assertDoesNotThrow {  
 jwtService.parseAccessToken(key)  
 }  
  
 assertThrows(io.jsonwebtoken.security.SignatureException::class.java) {  
 jwtService.parseAccessToken(key + "z")  
 }  
 assertThrows(io.jsonwebtoken.MalformedJwtException::class.java) {  
 jwtService.parseAccessToken("$key.")  
 }  
 }  
}

## MeetServiceTest.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.data.status.MeetStatus  
import com.goga133.hsecoffee.entity.SearchParams  
import com.goga133.hsecoffee.entity.User  
import org.junit.Assert.\*  
import org.junit.jupiter.api.Assertions.assertEquals  
import org.junit.jupiter.api.BeforeAll  
import org.junit.jupiter.api.BeforeEach  
import org.junit.jupiter.api.Test  
import org.junit.jupiter.api.assertDoesNotThrow  
import org.junit.runner.RunWith  
import org.opentest4j.TestAbortedException  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.boot.test.context.SpringBootTest  
import org.springframework.test.context.junit4.SpringRunner  
import java.time.Instant  
import java.util.\*  
import java.util.concurrent.TimeUnit  
  
@RunWith(SpringRunner::class)  
@SpringBootTest  
internal class MeetServiceTest {  
 @Autowired  
 val userService: UserService? = null  
  
 @Autowired  
 val meetService: MeetService? = null  
  
 val user1: User = User("testUSER1")  
 val user2: User = User("testUSER2")  
 val user3: User = User("testUSER3")  
 val user4: User = User("testUSER4")  
 val user5: User = User("testUSER5")  
  
 @BeforeEach  
 fun init\_users() {  
 if (userService == null) {  
 throw TestAbortedException()  
 }  
  
 with(userService!!){  
 save(user1)  
 save(user2)  
 save(user3)  
 save(user4)  
 save(user5)  
 }  
 }  
  
 @Test  
 fun getMeet() {  
 val meet1 = meetService?.getMeet(user1)  
  
 assertTrue(meet1?.meetStatus == MeetStatus.NONE)  
 assertTrue(meet1?.user1 == null)  
 assertTrue(meet1?.user2 == null)  
 TimeUnit.MILLISECONDS.sleep(100)  
 assertTrue(meet1?.createdDate?.before(Date.from(Instant.now())) == true)  
 }  
  
 @Test  
 fun cancelSearch() {  
 meetService?.searchMeet(user5, SearchParams())  
  
 val meet2 = meetService?.getMeet(user5)  
 assertTrue(meet2?.meetStatus == MeetStatus.SEARCH)  
  
 meetService?.cancelSearch(user5)  
 assertTrue(meetService?.getMeet(user5)?.meetStatus == MeetStatus.NONE)  
 }  
  
 @Test  
 fun searchMeet() {  
 meetService?.apply {  
 searchMeet(user4, SearchParams())  
 searchMeet(user3, SearchParams())  
 }  
  
 val meet3 = meetService?.getMeet(user3)  
 val meet4 = meetService?.getMeet(user4)  
  
 assertEquals(meet3?.id, meet4?.id)  
  
 assertEquals(meet3?.meetStatus, MeetStatus.ACTIVE)  
 assertEquals(meet3?.user1?.id, user3.id)  
 assertEquals(meet3?.user2?.id, user4.id)  
  
  
 }  
  
 @Test  
 fun methods\_non\_database\_user(){  
 val nonDbUser = User("123")  
  
 assertDoesNotThrow {  
 meetService?.getMeet(nonDbUser)  
 meetService?.getMeets(nonDbUser)  
 meetService?.cancelSearch(nonDbUser)  
 meetService?.searchMeet(nonDbUser, SearchParams())  
 }  
 }  
  
 @Test  
 fun getMeets() {  
 assertTrue(meetService?.getMeets(user1)?.isEmpty() == true)  
 assertTrue(meetService?.getMeets(user2)?.isEmpty() == true)  
 assertTrue(meetService?.getMeets(user3)?.isEmpty() == true)  
 assertTrue(meetService?.getMeets(user4)?.isEmpty() == true)  
 }  
  
}

## RefreshTokenServiceTest.kt

package com.goga133.hsecoffee.service  
  
import com.goga133.hsecoffee.entity.User  
import org.junit.Assert.assertFalse  
import org.junit.Assert.assertTrue  
import org.junit.jupiter.api.Test  
import org.junit.runner.RunWith  
import org.springframework.beans.factory.annotation.Autowired  
import org.springframework.boot.test.context.SpringBootTest  
import org.springframework.test.context.junit4.SpringRunner  
import java.util.\*  
import javax.transaction.Transactional  
  
@RunWith(SpringRunner::class)  
@SpringBootTest  
@Transactional  
internal class RefreshTokenServiceTest {  
  
 @Autowired  
 val userService : UserService? = null  
  
 @Autowired  
 val refreshTokenService: RefreshTokenService? = null  
  
 @Test  
 fun createByUserAndValidate() {  
 val mockUser: User = User("test@test")  
 val fingerprint: String = "mock\_fingerprint"  
  
 userService?.save(mockUser)  
  
 val refreshToken = refreshTokenService!!.createByUser(mockUser, fingerprint)  
  
 *// Неверный UUID и fingerprint.* assertFalse(refreshTokenService!!.isValid(mockUser, UUID.randomUUID(), "finger"))  
 *// Неверный fingerprint* assertFalse(refreshTokenService!!.isValid(mockUser, refreshToken.uuid, "finger"))  
 *// Неверный UUID. WARNING: В редких случаях может совпасть UUID.* assertFalse(refreshTokenService!!.isValid(mockUser, UUID.randomUUID(), refreshToken.fingerprint))  
 *// Всё верно* assertTrue(refreshTokenService!!.isValid(mockUser, refreshToken.uuid, refreshToken.fingerprint))  
 *// Старые данные* assertTrue(refreshTokenService!!.isValid(mockUser, refreshToken.uuid, refreshToken.fingerprint))  
 }  
}

## Build.gradle.kts

import org.jetbrains.kotlin.gradle.tasks.KotlinCompile  
  
plugins **{** application  
 id("org.springframework.boot") version "2.4.0"  
 id("io.spring.dependency-management") version "1.0.10.RELEASE"  
 kotlin("jvm") version "1.4.10"  
 kotlin("plugin.spring") version "1.4.10"  
**}**group = "com.goga133"  
version = "1.0.1-BETA"  
java.sourceCompatibility = JavaVersion.VERSION\_11  
  
repositories **{** mavenCentral()  
**}**application **{** mainClassName = "com.goga133.hsecoffee.RunKt"  
**}**dependencies **{** implementation("org.springframework.boot:spring-boot-starter-web")  
 implementation("org.springframework.boot:spring-boot-starter-mail")  
 implementation("org.springframework.boot:spring-boot-starter-data-jpa")  
 implementation("org.springframework.boot:spring-boot-starter")  
 implementation("junit:junit:4.12")  
 compileOnly( "org.springframework.boot:spring-boot-configuration-processor")  
 runtimeOnly("mysql:mysql-connector-java")  
 runtimeOnly("com.h2database:h2:1.4.200")  
  
 testImplementation("org.springframework.boot:spring-boot-starter-test")  
 implementation("io.jsonwebtoken:jjwt-api:0.11.2")  
 runtimeOnly("io.jsonwebtoken:jjwt-impl:0.11.2")  
 runtimeOnly("io.jsonwebtoken:jjwt-jackson:0.11.2")  
  
 implementation("com.fasterxml.jackson.module:jackson-module-kotlin")  
 implementation("org.jetbrains.kotlin:kotlin-reflect")  
 implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")  
 implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.1")  
**}**tasks**{** bootJar **{** manifest **{** attributes("Multi-Release" to true)  
 **}** if (project.hasProperty("archiveName")) {  
 archiveFileName.set(project.properties["archiveName"] as String)  
 }  
 **}  
}**tasks.withType<Test> **{** useJUnitPlatform()  
**}**tasks.withType<KotlinCompile> **{** kotlinOptions **{** freeCompilerArgs = listOf("-Xjsr305=strict")  
 jvmTarget = "11"  
 **}  
}**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |