Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Керівник роботи
_____ Ілля АХАЛАДЗЕ
“___” _____ 2024 р.

Мобільний застосунок для контролю особистого часу

Текст програми

КПІ.ІП-1324.045490.05.13

“ПОГОДЖЕНО”
Керівник роботи:
_____ Ілля АХАЛАДЗЕ

Виконавець:
_____ Віталій НЕЩЕРЕТ

Київ – 2024

Файл LoginFragment.kt

```
package com.makelick.anytime.view.login

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.contract.ActivityResultContracts
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.common.api.ApiException
import
com.google.firebase.auth.FirebaseAuthInvalidCredentialsException
import com.google.firebase.auth.FirebaseAuthInvalidUserException
import
com.google.firebase.auth.FirebaseAuthUserCollisionException
import
com.google.firebase.auth.FirebaseAuthWeakPasswordException
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentLoginBinding
import com.makelick.anytime.view.BaseFragment
import com.makelick.anytime.view.MainActivity
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class LoginFragment :

BaseFragment<FragmentLoginBinding>(FragmentLoginBinding::inflate
) {

    private val viewModel: LoginViewModel by viewModels()

    private val googleSignInLauncher:
ActivityResultLauncher<Intent> =

registerForActivityResult(ActivityResultContracts.StartActivityF
orResult()) { result ->
            try {
                val task =
GoogleSignIn.getSignedInAccountFromIntent(result.data)
                val account =
task.getResult(ApiException::class.java)
                viewModel.signInWithGoogle(account)
            } catch (e: ApiException) {
                Toast.makeText(
                    requireContext(),
                    getString(R.string.google_sign_in_failed),
                    Toast.LENGTH_SHORT
```

```
                ).show()
            }
        }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        observeViewModel()
        setupUI()
    }

    private fun observeViewModel() {
        lifecycleScope.launch {
            viewModel.isLoginMode.collect { changeMode(it) }
        }

        lifecycleScope.launch {
            viewModel.result.collect { handleResult(it) }
        }

        lifecycleScope.launch {
            viewModel.isLoading.collect {
                with(binding) {
                    button.isEnabled = !it
                    googleSignInButton.isEnabled = !it
                    changeMode.isEnabled = !it
                    binding.loadingBar.visibility = if (it)
View.VISIBLE else View.GONE
                }
            }
        }
    }

    private fun changeMode(isSignIn: Boolean) {
        with(binding) {
            if (isSignIn) {
                passwordConfirmationLayout.visibility =
View.GONE
                button.text = getString(R.string.sign_in)
                changeMode.text =
getString(R.string.create_new_account)
            } else {
                passwordConfirmationLayout.visibility =
View.VISIBLE
                button.text = getString(R.string.sign_up)
                changeMode.text =
getString(R.string.sign_in_to_existing_account)
            }
        }
    }

    private fun handleResult(result: Result<Unit>) {
        if (result.isFailure) {
```

```
            with(binding) {
                when (result.exceptionOrNull()) {
                    is FirebaseAuthInvalidUserException -> {
                        emailLayout.error =
getString(R.string.invalid_email)
                    }

                    is FirebaseAuthUserCollisionException -> {
                        emailLayout.error =
getString(R.string.email_already_in_use)
                    }

                    is FirebaseAuthWeakPasswordException -> {
                        passwordLayout.error =
getString(R.string.weak_password)
                    }

                    is FirebaseAuthInvalidCredentialsException -
> {
                        emailLayout.error =
getString(R.string.invalid_email)
                    }

                    else -> {
                        passwordLayout.error =
getString(R.string.invalid_email_or_password)
                        emailLayout.error =
getString(R.string.invalid_email_or_password)
                    }
                }
                root.clearFocus()
            }
        } else {
            navigateToTasksFragment()
        }
    }

    private fun navigateToTasksFragment() {

findNavController().navigate(LoginFragmentDirections.actionLogin
FragmentToTasksFragment())
        (activity as
MainActivity).changeBottomNavSelectedId(R.id.tasks)
    }

    private fun setupUI() {
        binding.apply {

            email.setOnFocusChangeListener { _, hasFocus ->
                if (hasFocus) emailLayout.error = null
            }

            password.setOnFocusChangeListener { _, hasFocus ->
```

```
                if (hasFocus) passwordLayout.error = null
            }

            passwordConfirmation.setOnFocusChangeListener { _,
hasFocus ->
                if (hasFocus) passwordConfirmationLayout.error =
null
            }

            changeMode.setOnClickListener {
                viewModel.changeMode()

                clearErrors()
                clearInputs()
                root.clearFocus()
            }

            button.setOnClickListener {
                clearErrors()
                root.clearFocus()
                attemptLogin()
            }

            googleSignInButton.setOnClickListener {

googleSignInLauncher.launch(viewModel.googleSignInIntent)
                root.clearFocus()
            }
        }
    }

    private fun clearErrors() {
        with(binding) {
            emailLayout.error = null
            passwordLayout.error = null
            passwordConfirmationLayout.error = null
        }
    }

    private fun clearInputs() {
        with(binding) {
            email.setText("")
            password.setText("")
            passwordConfirmation.setText("")
        }
    }

    private fun attemptLogin() {
        val emailStr = binding.email.text.toString()
        val passwordStr = binding.password.text.toString()

        if (!viewModel.isValidEmail(emailStr)) {
            binding.emailLayout.error =
```

```
getString(R.string.invalid_email_format)
            return
        }

        if (viewModel.isLoginMode.value) {
            viewModel.login(emailStr, passwordStr)
            return
        }

        if (passwordStr !=
binding.passwordConfirmation.text.toString()) {
            binding.passwordConfirmationLayout.error =
getString(R.string.passwords_do_not_match)
            return
        }

        viewModel.signUp(emailStr, passwordStr)
    }
}
```

Файл LoginViewModel.kt

```
package com.makelick.anytime.view.login

import android.content.Intent
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import
com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.makelick.anytime.model.AccountRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class LoginViewModel @Inject constructor(
    private val accountRepository: AccountRepository
) : ViewModel() {

    val isLoading = MutableStateFlow(false)
    val isLoginMode = MutableStateFlow(true)
    val result = MutableSharedFlow<Result<Unit>>()

    val googleSignInIntent: Intent
        get() = accountRepository.getGoogleSignInIntent()

    fun changeMode() {
        isLoginMode.value = !isLoginMode.value
    }

    fun login(email: String, password: String) {
        viewModelScope.launch {
            isLoading.value = true
```

```
            result.emit(accountRepository.signIn(email,
password))
            isLoading.value = false
        }
    }

    fun signUp(email: String, password: String) {
        viewModelScope.launch {
            isLoading.value = true
            result.emit(accountRepository.signUp(email,
password))
            isLoading.value = false
        }
    }

    fun signInWithGoogle(account: GoogleSignInAccount) {
        viewModelScope.launch {
            isLoading.value = true

result.emit(accountRepository.signInWithGoogle(account))
            isLoading.value = false

        }
    }

    fun isValidEmail(email: String): Boolean {
        val emailRegex = "^[A-Za-z](.*)(@)(.+)(\\.)(.+)$"
        return email.matches(emailRegex.toRegex())
    }
}
```

Файл AccountRepository.kt

```
package com.makelick.anytime.model

import android.content.Context
import android.content.Intent
import android.net.Uri
import com.google.android.gms.auth.api.signin.GoogleSignIn
import
com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.google.android.gms.auth.api.signin.GoogleSignInClient
import
com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.firebase.auth.GoogleAuthProvider
import com.google.firebase.auth.ktx.auth
import com.google.firebase.auth.ktx.userProfileChangeRequest
import com.google.firebase.ktx.Firebase
import com.makelick.anytime.R
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.tasks.await
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
```

```
class AccountRepository @Inject constructor(
    @ApplicationContext private val context: Context
) {
    private val auth = Firebase.auth

    private var googleSignInClient: GoogleSignInClient =
        GoogleSignIn.getClient(
            context,

GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)

.requestIdToken(context.getString(R.string.google_web_client_id)
)
                .requestEmail()
                .build()
        )

    fun getUser() = auth.currentUser

    fun updateProfile(username: String, photoUrl: Uri?) {
        auth.currentUser?.updateProfile(userProfileChangeRequest
{
            displayName = username
            photoUri = photoUrl
        })
    }

    suspend fun signIn(email: String, password: String):
Result<Unit> {
        return try {
            auth.signInWithEmailAndPassword(email,
password).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun signUp(email: String, password: String):
Result<Unit> {
        return try {
            auth.createUserWithEmailAndPassword(email,
password).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    fun getGoogleSignInIntent(): Intent {
        return googleSignInClient.signInIntent
    }
```

```
    suspend fun signInWithGoogle(account: GoogleSignInAccount):
Result<Unit> {
        return try {
            val credential =
GoogleAuthProvider.getCredential(account.idToken, null)
            auth.signInWithCredential(credential).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    fun signOut() {
        auth.signOut()
        googleSignInClient.signOut()
    }
}
```

Файл ProfileFragment.kt

```
package com.makelick.anytime.view.profile

import android.net.Uri
import android.os.Bundle
import android.view.View
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import coil.load
import coil.transform.CircleCropTransformation
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentProfileBinding
import com.makelick.anytime.view.BaseFragment
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class ProfileFragment :
BaseFragment<FragmentProfileBinding>(FragmentProfileBinding::inf
late) {

    private val viewModel: ProfileViewModel by viewModels()

    private val pickImageLauncher:
ActivityResultLauncher<PickVisualMediaRequest> =

registerForActivityResult(ActivityResultContracts.PickVisualMedi
a()) {
            it?.let { uploadImage(it) }
        }
```

```
    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        observeViewModel()
    }

    private fun setupUI() {
        with(binding) {

            viewModel.user?.let { user ->
                profileImage.load(user.photoUrl) {
                    transformations(CircleCropTransformation())
                    fallback(R.drawable.ic_profile)
                    error(R.drawable.ic_profile)
                }
                username.setText(viewModel.user?.displayName)
            }

            edit.setOnClickListener {
                root.clearFocus()
                changeMode()
            }

            username.onFocusChangeListener =
View.OnFocusChangeListener { _, _ ->
                usernameLayout.error = null
            }

            profileImage.setOnClickListener {
                if (viewModel.isEditMode.value) {
                    pickImageLauncher.launch(

PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.I
mageOnly)
                    )
                }
            }

            buttonManageCategories.setOnClickListener {
                navigateToCategories()
            }

            buttonLogOut.setOnClickListener {

MaterialAlertDialogBuilder(requireContext()).apply {
                    setTitle(getString(R.string.sign_out))

setMessage(getString(R.string.sign_out_message))
                    setPositiveButton(getString(R.string.yes)) {
dialog, _ ->
```

```
                        viewModel.signOut()
                        navigateToLogin()
                        dialog.dismiss()
                    }
                    setNegativeButton(getString(R.string.no)) {
dialog, _ -> dialog.dismiss() }
                    show()
                }
            }
        }
    }

    private fun navigateToCategories() {

findNavController().navigate(ProfileFragmentDirections.actionPro
fileFragmentToCategoriesFragment())
    }

    private fun navigateToLogin() {

findNavController().navigate(ProfileFragmentDirections.actionPro
fileFragmentToLoginFragment())
    }

    private fun changeMode() {
        if (viewModel.isEditMode.value) {
            if (binding.username.text.toString().isBlank()) {
                binding.usernameLayout.error =
getString(R.string.error_empty)
            } else {

viewModel.applyProfileChanges(binding.username.text.toString())
            }
        } else {
            viewModel.isEditMode.value = true
        }
    }

    private fun uploadImage(uri: Uri?) {
        lifecycleScope.launch {
            binding.imageLoadingBar.visibility = View.VISIBLE
            viewModel.loadNewImage(uri)
            binding.profileImage.load(viewModel.loadedImageUri)
{
                transformations(CircleCropTransformation())
                fallback(R.drawable.ic_profile)
                error(R.drawable.ic_profile)
            }
            binding.imageLoadingBar.visibility = View.GONE
        }
    }

    private fun observeViewModel() {
```

```
        lifecycleScope.launch {
            viewModel.isEditMode.collect { editMode ->
                if (editMode) {
                    binding.edit.text = getString(R.string.save)
                    binding.usernameLayout.isEnabled = true
                    binding.imageText.visibility = View.VISIBLE
                } else {
                    binding.edit.text = getString(R.string.edit)
                    binding.usernameLayout.isEnabled = false
                    binding.imageText.visibility = View.GONE
                }
            }
        }

        lifecycleScope.launch {
            viewModel.completedTasksCount.collect {
                binding.completedTasks.text = it.toString()
            }
        }

        lifecycleScope.launch {
            viewModel.uncompletedTasksCount.collect {
                binding.uncompletedTasks.text = it.toString()
            }
        }

    }
}
```

Файл ProfileViewModel.kt

```
package com.makelick.anytime.view.profile

import android.net.Uri
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.makelick.anytime.model.AccountRepository
import com.makelick.anytime.model.FirestoreRepository
import com.makelick.anytime.model.StorageRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class ProfileViewModel @Inject constructor(
    private val accountRepository: AccountRepository,
    private val storageRepository: StorageRepository,
    private val firestoreRepository: FirestoreRepository
) : ViewModel() {

    val user = accountRepository.getUser()
    val isEditMode = MutableStateFlow(false)
    val completedTasksCount = MutableStateFlow(0)
    val uncompletedTasksCount = MutableStateFlow(0)
```

```
    var loadedImageUri: Uri? = null

    init {
        viewModelScope.launch {
            firestoreRepository.allTasks.collect {
                completedTasksCount.emit(
                    firestoreRepository.allTasks.value.count {
it.isCompleted }
                )

                uncompletedTasksCount.emit(
                    firestoreRepository.allTasks.value.count {
!it.isCompleted }
                )
            }
        }
    }

    fun signOut() {
        accountRepository.signOut()
    }

    fun applyProfileChanges(username: String) {
        accountRepository.updateProfile(username, loadedImageUri
?: user?.photoUrl)
        isEditMode.value = false
    }

    suspend fun loadNewImage(file: Uri?) {
        if (user != null && file != null) {
            val result = storageRepository.uploadImage(user.uid,
file)
            loadedImageUri = result.getOrNull().takeIf {
result.isSuccess }
        }
    }
}
```

Файл TasksFragment.kt

```
package com.makelick.anytime.view.tasks

import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentTasksBinding
import com.makelick.anytime.model.entity.Task
import com.makelick.anytime.view.BaseFragment
```

```
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class TasksFragment :
BaseFragment<FragmentTasksBinding>(FragmentTasksBinding::inflate
) {

    private val viewModel: TasksViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        observeViewModel()
    }

    private fun setupUI() {
        with(binding) {

            spinnerPriority.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(parent:
AdapterView<*>?, view: View?, position: Int, id: Long) {
                    viewModel.selectedPriority.value =
spinnerPriority.selectedItem.toString().convertPriorityToInt()
                    viewModel.loadTasks()
                }

                override fun onNothingSelected(p0:
AdapterView<*>?) {
                    viewModel.selectedPriority.value = -1
                    viewModel.loadTasks()
                }
            }

            spinnerCategory.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(parent:
AdapterView<*>?, view: View?, position: Int, id: Long) {
                    viewModel.selectedCategory.value =
spinnerCategory.selectedItem.toString()
                    viewModel.loadTasks()
                }

                override fun onNothingSelected(p0:
AdapterView<*>?) {
                    viewModel.selectedCategory.value = "All
categories"
                    viewModel.loadTasks()
                }
            }
```

```
            tasksRecyclerView.apply {
                adapter =
TasksAdapter(viewModel::changeTaskStatus, ::navigateToTaskInfo)
                layoutManager =
LinearLayoutManager(requireContext())
            }

            addTaskButton.setOnClickListener {
                navigateToCreateTask()
            }
        }
    }

    private fun String.convertPriorityToInt(): Int {
        return when (this) {
            getString(R.string.high_priority) -> 3
            getString(R.string.medium_priority) -> 2
            getString(R.string.low_priority) -> 1
            getString(R.string.no_priority) -> 0
            else -> -1
        }
    }

    private fun navigateToTaskInfo(task: Task) {
        val action =
TasksFragmentDirections.actionTasksFragmentToTaskInfoFragment(ta
sk)
        findNavController().navigate(action)
    }

    private fun navigateToCreateTask() {
        val action =
TasksFragmentDirections.actionTasksFragmentToEditTaskFragment(tr
ue, null)
        findNavController().navigate(action)
    }

    private fun observeViewModel() {
        lifecycleScope.launch {
            viewModel.isLoading.collect {
                binding.tasksLoadingBar.visibility = if (it)
View.VISIBLE else View.GONE
            }
        }

        lifecycleScope.launch {
            viewModel.tasks.collect {
                (binding.tasksRecyclerView.adapter as
TasksAdapter).submitList(it)
                binding.emptyTasksText.visibility = if
(it.isEmpty()) View.VISIBLE else View.GONE
            }
```

```
        }

        lifecycleScope.launch {
            viewModel.categories.collect {
                binding.spinnerCategory.adapter = ArrayAdapter(
                    requireContext(),

android.R.layout.simple_spinner_dropdown_item,
                    it
                )
            }
        }
    }
}
```

Файл TasksAdapter.kt

```
package com.makelick.anytime.view.tasks

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.core.content.ContextCompat.getColor
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView.ViewHolder
import com.makelick.anytime.R
import com.makelick.anytime.databinding.ItemTasklistBinding
import com.makelick.anytime.model.entity.Task

class TasksAdapter(
    private val onCheckboxClick: (Task) -> Unit,
    private val onTaskClick: (Task) -> Unit
) : ListAdapter<Task,
TasksAdapter.TasksViewHolder>(TaskDiffCallback()) {
    class TaskDiffCallback : DiffUtil.ItemCallback<Task>() {
        override fun areItemsTheSame(oldItem: Task, newItem:
Task) =
            oldItem.id == newItem.id

        override fun areContentsTheSame(oldItem: Task, newItem:
Task) =
            oldItem == newItem
    }

    inner class TasksViewHolder(private val binding:
ItemTasklistBinding) :
        ViewHolder(binding.root) {
        fun bind(task: Task) {
            with(binding) {

                taskTitle.text = task.title


taskPriority.setBackgroundColor(getPriorityColor(task.priority))
```

```
                taskCheckBox.isChecked = task.isCompleted ==
true
                taskCheckBox.setOnClickListener {
                    onCheckboxClick(task)
                }
                root.setOnClickListener {
                    onTaskClick(task)
                }
            }
        }

        private fun getPriorityColor(priority: Int?) = when
(priority) {
            1 -> getColor(binding.root.context,
R.color.low_priority)
            2 -> getColor(binding.root.context,
R.color.medium_priority)
            3 -> getColor(binding.root.context,
R.color.high_priority)
            else -> getColor(binding.root.context,
R.color.no_priority)
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int) =
        TasksViewHolder(
            ItemTasklistBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )

    override fun onBindViewHolder(holder: TasksViewHolder,
position: Int) {
        holder.bind(getItem(position))
    }
}
```

Файл TasksViewModel.kt

```
package com.makelick.anytime.view.tasks

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.makelick.anytime.model.FirestoreRepository
import com.makelick.anytime.model.entity.Task
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class TasksViewModel @Inject constructor(
```

```
    private val firestoreRepository: FirestoreRepository
) : ViewModel() {

    val isLoading = MutableStateFlow(true)
    val selectedPriority = MutableStateFlow(-1)
    val selectedCategory = MutableStateFlow("All categories")

    val tasks = MutableStateFlow<List<Task>>(emptyList())
    val categories = MutableStateFlow<List<String>>(emptyList())

    init {
        viewModelScope.launch {
            firestoreRepository.allTasks.collect {
                loadTasks()
            }
        }

        viewModelScope.launch {
            firestoreRepository.categories.collect {
                categories.value = loadCategories()
            }
        }
    }

    fun loadTasks() {
        viewModelScope.launch {

tasks.emit(filterTasks(firestoreRepository.allTasks.value))
            isLoading.value = false
        }
    }

    private fun filterTasks(tasks: List<Task>): List<Task> {
        return tasks.filter { task ->
            (selectedPriority.value == -1 || task.priority ==
selectedPriority.value) &&
                    (selectedCategory.value == "All categories"
|| task.category == selectedCategory.value)

        }.sortedBy { it.isCompleted }
    }

    private fun loadCategories(): List<String> {
        val result = mutableListOf("All categories")
        result.addAll(firestoreRepository.categories.value)
        return result
    }

    fun changeTaskStatus(task: Task) {
        viewModelScope.launch {
            task.isCompleted = !task.isCompleted
            firestoreRepository.updateTask(task)
            tasks.emit(tasks.value.sortedBy { it.isCompleted })
```

```
        }
    }
}
```

Файл FirestoreRepository.kt

```
package com.makelick.anytime.model

import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import com.makelick.anytime.model.entity.Task
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class FirestoreRepository @Inject constructor(
    userId: String
) {

    private val userDocRef =
Firebase.firestore.document("users/$userId")
    private val tasksCollectionRef =
userDocRef.collection("tasks")

    val allTasks = MutableStateFlow<List<Task>>(emptyList())
    val categories = MutableStateFlow<List<String>>(emptyList())

    init {
        tasksCollectionRef.addSnapshotListener { value, error ->
            if (error != null) return@addSnapshotListener
            allTasks.value = value?.toObjects(Task::class.java)
?: emptyList()
        }

        userDocRef.addSnapshotListener { value, error ->
            if (error != null) return@addSnapshotListener

            if (value?.exists() == true) {
                categories.value =
                    (value.get("categories") as? List<*>)?.map {
it.toString() } ?: emptyList()
            } else {
                userDocRef.set(mapOf("categories" to
listOf("Personal", "Work")))
            }
        }
    }

    suspend fun addTask(task: Task): Result<Unit> {
        return try {
            task.id = tasksCollectionRef.document().id
            updateTask(task)
            Result.success(Unit)
```

```
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun updateTask(task: Task): Result<Unit> {
        return try {

tasksCollectionRef.document(task.id.toString()).set(task).await(
)
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun updateCategories(newCategories: List<String>):
Result<Unit> {
        return try {
            userDocRef.update("categories",
newCategories).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    suspend fun deleteTask(taskId: String) {
        tasksCollectionRef.document(taskId).delete().await()
    }
}
```

Файл FocusFragment.kt

```
package com.makelick.anytime.view.focus

import android.Manifest
import android.content.Intent
import android.os.Build
import android.os.Bundle
import android.view.View
import androidx.core.content.PermissionChecker
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import com.makelick.anytime.R
import com.makelick.anytime.databinding.FragmentFocusBinding
import
com.makelick.anytime.model.TimerRepository.Companion.SECOND
import com.makelick.anytime.model.entity.PomodoroMode
import com.makelick.anytime.view.BaseFragment
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import java.util.Locale

@AndroidEntryPoint
```

```
class FocusFragment :
BaseFragment<FragmentFocusBinding>(FragmentFocusBinding::inflate
) {

    private val viewModel: FocusViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        observeViewModel()
        askNotificationPermission()
    }

    private fun askNotificationPermission() {
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            if (PermissionChecker.checkSelfPermission(
                    requireContext(),
                    Manifest.permission.POST_NOTIFICATIONS
                ) != PermissionChecker.PERMISSION_GRANTED
            ) {
                @Suppress("DEPRECATION")

requestPermissions(arrayOf(Manifest.permission.POST_NOTIFICATION
S), 1)
            }
        }
    }

    private fun setupUI() {

        binding.playButton.setOnClickListener {
            if (viewModel.isTimerRunning.value) {
                context?.stopService(Intent(context,
TimerService::class.java))
                viewModel.pauseTimer()

binding.iconPlay.setImageResource(R.drawable.ic_play)
            } else {
                context?.startService(Intent(context,
TimerService::class.java))

binding.iconPlay.setImageResource(R.drawable.ic_pause)
            }
        }

        binding.restartButton.setOnClickListener {
            stopTimer()

binding.iconPlay.setImageResource(R.drawable.ic_play)
        }
```

```
        binding.nextButton.setOnClickListener {
            viewModel.nextMode()
            stopTimer()
            context?.startService(Intent(context,
TimerService::class.java))

binding.iconPlay.setImageResource(R.drawable.ic_pause)
        }
    }

    private fun stopTimer() {
        context?.stopService(Intent(context,
TimerService::class.java))
        viewModel.stopTimer()
    }

    private fun observeViewModel() {

        lifecycleScope.launch {
            viewModel.isTimerRunning.collect {
                if (it) {

binding.iconPlay.setImageResource(R.drawable.ic_pause)
                } else {

binding.iconPlay.setImageResource(R.drawable.ic_play)
                }
            }
        }

        lifecycleScope.launch {
            viewModel.currentTime.collect {
                binding.time.text = getStringTime(it)
            }
        }

        lifecycleScope.launch {
            viewModel.timerMode.collect {
                with(binding) {
                    title.text = it.title
                    hint.text = getHint(it)


timeCard.setCardBackgroundColor(getTimerColor(it))

playButton.setCardBackgroundColor(getTimerColor(it))

                    if (it == PomodoroMode.POMODORO) {
                        countOfBreaks.text = getString(
                            R.string.focus_count_of_breaks,
                            viewModel.timerBreaksCount.value
                        )
```

```
                        countOfBreaks.visibility = View.VISIBLE
                    } else {
                        countOfBreaks.visibility = View.GONE
                    }
                }
            }
        }

    private fun getTimerColor(mode: PomodoroMode): Int {
        return when (mode) {
            PomodoroMode.POMODORO ->
resources.getColor(R.color.primary, null)
            PomodoroMode.SHORT_BREAK ->
resources.getColor(R.color.secondary, null)
            PomodoroMode.LONG_BREAK ->
resources.getColor(R.color.accent, null)
        }
    }

    private fun getHint(mode: PomodoroMode): String {
        return when (mode) {
            PomodoroMode.POMODORO ->
getString(R.string.focus_hint_pomodoro)
            PomodoroMode.SHORT_BREAK ->
getString(R.string.focus_hint_short_break)
            PomodoroMode.LONG_BREAK ->
getString(R.string.focus_hint_long_break)
        }
    }

    private fun getStringTime(time: Long): String {
        val minutes = (time / SECOND) / 60
        val seconds = (time / SECOND) % 60
        return String.format(Locale.getDefault(), "%02d:%02d",
minutes, seconds)
    }
}
```

Файл FocusViewModel.kt

```
package com.makelick.anytime.view.focus

import androidx.lifecycle.ViewModel
import com.makelick.anytime.model.TimerRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

@HiltViewModel
class FocusViewModel @Inject constructor(
    private val timerRepository: TimerRepository
) : ViewModel() {

    val timerMode = timerRepository.timerMode
    val timerBreaksCount = timerRepository.timerBreaksCount
```

```
    val isTimerRunning  = timerRepository.isTimerRunning
    val currentTime = timerRepository.currentTime

    fun pauseTimer() {
        timerRepository.pauseTimer()
    }

    fun stopTimer() {
        timerRepository.stopTimer()
    }

    fun nextMode() {
        timerRepository.nextMode()
    }

}
```

Файл TimerService.kt

```
package com.makelick.anytime.view.focus

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.Service
import android.content.Intent
import androidx.core.app.NotificationCompat
import androidx.navigation.NavDeepLinkBuilder
import com.makelick.anytime.R
import com.makelick.anytime.model.TimerRepository
import
com.makelick.anytime.model.TimerRepository.Companion.SECOND
import com.makelick.anytime.model.entity.PomodoroMode
import com.makelick.anytime.view.MainActivity
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import java.util.Locale
import javax.inject.Inject

@AndroidEntryPoint
class TimerService : Service() {

    @Inject
    lateinit var timerRepository: TimerRepository

    override fun onStartCommand(intent: Intent?, flags: Int,
startId: Int): Int {


timerRepository.startTimer(timerRepository.timerMode.value.timeI
nMillis)
        createNotificationChannel()
```

```
        CoroutineScope(Dispatchers.Main).launch {
            timerRepository.currentTime.collect {
                if (it > 0) {
                    val notification = createNotification(

getString(R.string.timer_notification_content,
getStringTime(it))
                    )
                    startForeground(1, notification)
                } else {
                    val notification = createNotification(

getString(R.string.notification_timer_finished),
                        true
                    )

getSystemService(NotificationManager::class.java).notify(2,
notification)
                    timerRepository.nextMode()
                    timerRepository.stopTimer()
                    delay(SECOND)

timerRepository.startTimer(timerRepository.timerMode.value.timeI
nMillis)
                }
            }
        }

        return super.onStartCommand(intent, flags, startId)
    }

    override fun onBind(p0: Intent?) = null

    private fun getStringTime(time: Long): String {
        val minutes = (time / SECOND) / 60
        val seconds = (time / SECOND) % 60
        return String.format(Locale.getDefault(),"%02d:%02d",
minutes, seconds)
    }

    private fun createNotification(
        content: String,
        isFinal: Boolean = false
    ): Notification {
        val pendingIntent = NavDeepLinkBuilder(this)
            .setComponentName(MainActivity::class.java)
            .setGraph(R.navigation.graph)
            .setDestination(R.id.focusFragment)
            .createPendingIntent()


        return NotificationCompat.Builder(this,
NOTIFICATIONS_CHANNEL_NAME)
```

```
.setContentTitle(timerRepository.timerMode.value.title)
            .setContentText(content)
            .setSmallIcon(R.drawable.ic_focus)

.setForegroundServiceBehavior(NotificationCompat.FOREGROUND_SERV
ICE_IMMEDIATE)
            .setContentIntent(pendingIntent)
            .setSilent(!isFinal)
            .setOngoing(!isFinal)
            .build()
    }

    private fun createNotificationChannel() {
        val channel = NotificationChannel(
            NOTIFICATIONS_CHANNEL_NAME,
            PomodoroMode.POMODORO.title,
            NotificationManager.IMPORTANCE_HIGH
        ).apply {
            description =
getString(R.string.timer_notification_channel_description)
        }


getSystemService(NotificationManager::class.java).createNotifica
tionChannel(channel)
    }

    companion object {
        const val NOTIFICATIONS_CHANNEL_NAME = "TIMER_CHANNEL"
    }
}
```

Файл TimerRepository.kt

```
package com.makelick.anytime.model

import android.os.CountDownTimer
import com.makelick.anytime.model.entity.PomodoroMode
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class TimerRepository @Inject constructor(
    private val dataStoreRepository: DataStoreRepository
) {

    private var timer: CountDownTimer? = null

    val timerMode = MutableStateFlow(PomodoroMode.POMODORO)
```

```
    val timerBreaksCount = MutableStateFlow(0)
    val isTimerRunning = MutableStateFlow(timer != null)
    val currentTime = MutableStateFlow<Long>(0)


    init {
        CoroutineScope(Dispatchers.IO).launch {
            timerMode.emit(
                getModeByTitle(

dataStoreRepository.getFromDataStore(DataStoreRepository.KEY_TIM
ER_MODE)
                    .first() ?: PomodoroMode.POMODORO.title)
            )
            currentTime.value = timerMode.value.timeInMillis
        }
        CoroutineScope(Dispatchers.IO).launch {
            timerBreaksCount.emit(

dataStoreRepository.getFromDataStore(DataStoreRepository.KEY_TIM
ER_BREAKS_COUNT)
                    .first() ?: 0
            )
        }
    }

    fun startTimer(timeInMillis: Long) {
        isTimerRunning.value = true
        timer = object : CountDownTimer(timeInMillis, SECOND) {
            override fun onTick(millisUntilFinished: Long) {
                currentTime.value = millisUntilFinished
            }

            override fun onFinish() {
                currentTime.value = 0
            }
        }.start()
    }

    fun pauseTimer() {
        isTimerRunning.value = false
        timer?.cancel()
        timer = null
    }

    fun stopTimer() {
        isTimerRunning.value = false
        timer?.cancel()
        timer = null
        currentTime.value = timerMode.value.timeInMillis
    }

    fun nextMode() {
```

```
        CoroutineScope(Dispatchers.IO).launch {
            if (timerMode.value == PomodoroMode.POMODORO) {
                if (timerBreaksCount.value == COUNT_OF_BREAKS) {
                    dataStoreRepository.saveToDataStore(
                        DataStoreRepository.KEY_TIMER_MODE,
                        PomodoroMode.LONG_BREAK.title
                    )
                    timerMode.value = PomodoroMode.LONG_BREAK

                    dataStoreRepository.saveToDataStore(

DataStoreRepository.KEY_TIMER_BREAKS_COUNT,
                        0
                    )
                    timerBreaksCount.value = 0
                } else {
                    dataStoreRepository.saveToDataStore(
                        DataStoreRepository.KEY_TIMER_MODE,
                        PomodoroMode.SHORT_BREAK.title
                    )
                    timerMode.value = PomodoroMode.SHORT_BREAK

                    dataStoreRepository.saveToDataStore(

DataStoreRepository.KEY_TIMER_BREAKS_COUNT,
                        timerBreaksCount.value + 1
                    )
                    timerBreaksCount.value =
timerBreaksCount.value + 1
                }
            } else {
                dataStoreRepository.saveToDataStore(
                    DataStoreRepository.KEY_TIMER_MODE,
                    PomodoroMode.POMODORO.title
                )
                timerMode.value = PomodoroMode.POMODORO
            }
            currentTime.value = timerMode.value.timeInMillis
        }
    }

    private fun getModeByTitle(title: String): PomodoroMode {
        return when (title) {
            PomodoroMode.POMODORO.title -> PomodoroMode.POMODORO
            PomodoroMode.SHORT_BREAK.title ->
PomodoroMode.SHORT_BREAK
            PomodoroMode.LONG_BREAK.title ->
PomodoroMode.LONG_BREAK
            else -> PomodoroMode.POMODORO
        }
    }

    companion object {
```

```
        private const val COUNT_OF_BREAKS = 4
        const val SECOND = 1_000L
        private const val MINUTE = 60 * SECOND
        const val POMODORO_TIME = 25 * MINUTE
        const val SHORT_BREAK_TIME = 5 * MINUTE
        const val LONG_BREAK_TIME = 15 * MINUTE
    }
}
```

Файл DataStoreRepository.kt

```
package com.makelick.anytime.model

import android.content.Context
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.core.intPreferencesKey
import androidx.datastore.preferences.core.stringPreferencesKey
import androidx.datastore.preferences.preferencesDataStore
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import javax.inject.Inject
import javax.inject.Singleton


@Singleton
class DataStoreRepository @Inject
constructor(@ApplicationContext private val context: Context) {

    private val Context.dataStore: DataStore<Preferences> by
preferencesDataStore(name = "AnyTimeDataStore")

    suspend fun <T> saveToDataStore(key: Preferences.Key<T>,
value: T) {
        context.dataStore.edit { preferences ->
            preferences[key] = value
        }
    }

    fun <T> getFromDataStore(key: Preferences.Key<T>): Flow<T?>
{
        val data = context.dataStore.data.map { preferences ->
            preferences[key]
        }
        return data
    }

    companion object {
        val KEY_TIMER_MODE = stringPreferencesKey("mode")
        val KEY_TIMER_BREAKS_COUNT =
intPreferencesKey("breaks_count")
    }
```

```
}
```