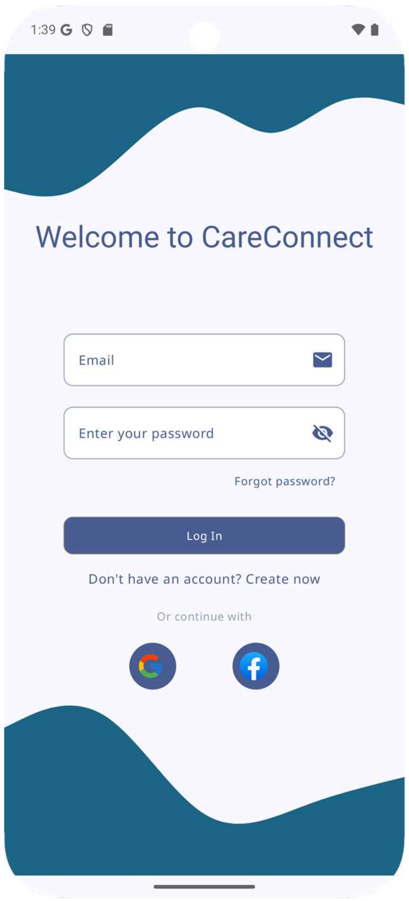
CareConnect

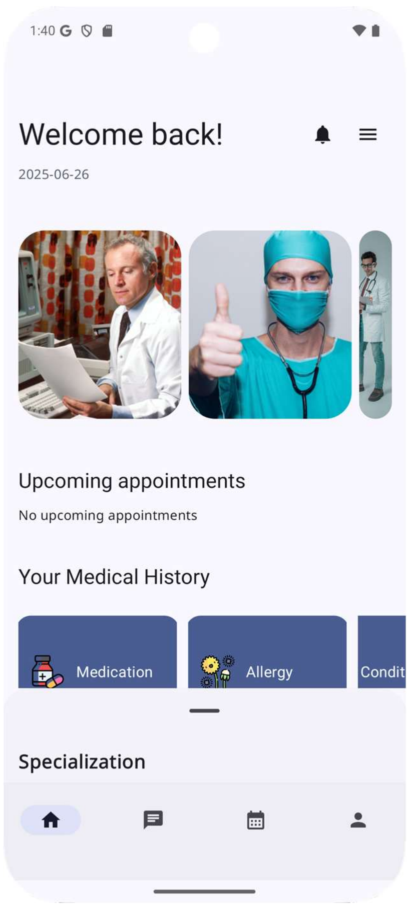
Team contribute: Quang - 50%, Nerike – 50%

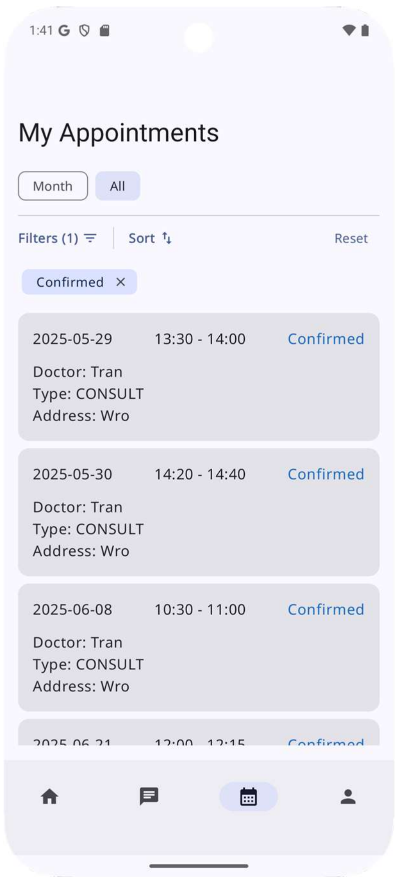
Github link: https://github.com/quangptt0910/CareConnect.git

Introduction

An E-clinic app for android, this app is targeting for the hospital as the **admin** will be the manager/director of the hospital. The app provides an ability for the **doctors** to communicate with **patients** and be a safe place to store all their data to view it easier and quicker







2025-05-29 13:30 - 14:00 Confirmed
Doctor: Tran
Type: CONSULT
Address: Wro

2025-05-30 14:20 - 14:40 Confirmed
Doctor: Tran
Type: CONSULT
Address: Wro

2025-06-08 10:30 - 11:00 Confirmed
Doctor: Tran
Type: CONSULT
Address: Wro

Technologies

This application is created with:

- Android studio
- Kotlin 2.1
- Jetpack composables + Material3
- Firebase
    - Firebase Authentication
    - Firestore Database
    - Firebase Storage
    - Firebase Cloud Messaging
    - Firebase Functions

Features

1. **Authentication**
    - Admin account is created in backend database
    - Doctor is created only with Admin side using firebase functions
    - Patient can register using Email/Password Authentication or Google Authentication (both method)
2. **Patient**
    - Able to see and filter doctors by specialization
    - Able to view doctors's availability, timeslots and book(request) appointments
    - Able to view past/upcoming appointments, filter by status of appointments
    - Real-time chat with doctor, able to send files
    - Able to view prescriptions, medical reports, images,...
3. **Doctor**
    - Able to modify schedules and timeslots
    - Response to request appointments from patients
    - Able to view past/upcoming appointments, filter by status of appointments
4. **Admin**
    - Able to create account for doctor (using firebase functions)
    - See list of doctors, patients and appointments
5. **Notifications**
    - Sent to doctor when the patient request an appointment
    - Sent to patient after the doctor response and change information of the appointment
    - When user(patient or doctor) receives new message from chat

Setup

1. Clone this repository
2. Create a Firebase project with Firebase console
3. Follow these steps to register the app in your Firebase project and add the Firebase Android configuration file
4. Create a Cloud Firestore database in your Firebase project
5. Enable Email/Password Authentication in your Firebase project
6. Enable Google login Authentication in your Firebase project
7. Setup the firestore, fcm, firebase storage and firebase functions
8. Run the app using Android Studio Meerkat+ on a device/emulator with API level 26 or above

Database

Firebase Cloud Firestore

Collections

- **admins**: Responsible for managing the entire system, including adding and removing doctors, modifying doctor profiles, updating their availability schedules, and managing patient/user data.
- **doctors**: A medical professional who can update their profile information, set their availability for appointments, view scheduled consultations, and interact with patients via chat during e-consultations.
    - patient_list/{patientId}: Store the patient as id with time added.
    - schedules/{yyyy-mm-dd}: doctor schedule and working day + timeslot, id is date.
    - tasks/{taskId}: doctor simple task where have name and see whether is checked or not.
- **patients**: A general user who can register for an account, browse available doctors, book appointments, attend e-consultations via chat, view past consultations in their medical history, and upload/download medical documents such as prescriptions.
    - medications / allergies / surgeries: patient medical history section collection
    - medicalReport / Prescription: report and prescription after appointments with doctors.
- **appointments**: Appointments have doctor/patient id and name, with time, address, status and type.
- **chatrooms**: Store all the chat between doctors and patients
    - messages: have the messages in the chatroom

- **notification_triggers**: notification send to doctor when an appointment is requested by patient and send to patient when its status changed (PENDING -> CONFIRMED/CANCELED)
- **scheduled_notifications**: remind about the appointment
- **chat_notifications**:
- **user_tokens**: have the device and token of user for fcm and notifications.

Key parts of the code

- Application have 3 side: Admin, doctor and patient app.

CareConnectApp and CareConnectNavHost are shared between 3 apps:

```kotlin
fun CareConnectApp(
...
) {
    RequestNotificationPermission() // Requests notification permission if needed.

    val context = LocalContext.current // Provides access to Android context.
    val scope = rememberCoroutineScope() // Coroutine scope tied to this composable's lifecycle.
    val snackbarHostState = remember { SnackbarHostState() } // State for managing snackbar visibility and content.

    // State to hold data extracted from a notification intent.
    var notificationData by remember { mutableStateOf<NotificationData?>(null) }

..                }
                }
            }
    }

    // Lambda to simplify showing snackbar messages.
    val showSnackbar: (SnackBarMessage) -> Unit = { snackBarMsg ->
        val resolvedMessage = getMessage(snackBarMsg) // Resolve SnackBarMessage to a String.
        scope.launch { snackbarHostState.showSnackbar(resolvedMessage) }
    }

    CareConnectTheme {
        Surface(
            ...
        ) {
            Scaffold(
                ...          }
        }
    }
}

/**
 * A composable function that requests the `POST_NOTIFICATIONS` permission
 * on Android Tiramisu (API 33) and above.
 *
 * It uses the Accompanist Permissions library to handle the permission request flow.
 * The permission request is launched automatically if it's not already granted.
 */
@OptIn(ExperimentalPermissionsApi::class)
@Composable
fun RequestNotificationPermission() {
    // Notification permission is only needed for Android 13 (Tiramisu) and higher.
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        // State for managing the POST_NOTIFICATIONS permission.
        val permissionState = rememberPermissionState(permission = Manifest.permission.POST_NOTIFICATIONS)

        // Effect to launch the permission request if not granted.
        // Runs once when the composable enters the composition.
        LaunchedEffect(Unit) {
            if (!permissionState.status.isGranted) {
                permissionState.launchPermissionRequest()
            }
        }
    }
}

/**
 * Composable function that defines the navigation graph for the CareConnect application.
 *
 * It uses a [NavHost] to manage navigation between different screens (composables)
 * based on defined routes.
 *
 * @param modifier Optional [Modifier] for this composable.
 * @param navController The [NavHostController] used for navigation.
 * @param startDestination The route of the initial screen to be displayed. Defaults to [SPLASH_ROUTE].
 * @param showSnackBar A lambda function that screens can call to display a [SnackBarMessage].
 * @param snackbarHostState The [SnackbarHostState] to be used by the [SnackbarHost],
 *                          potentially allowing screens to directly interact with it if needed.
 * @param notificationData Optional [NotificationData] that might have been received from an
 *                         incoming notification. This can be passed to relevant screens.
 */
@Composable
fun CareConnectNavHost(
    …
) {
    NavHost(
        modifier = modifier,
        navController = navController,
        startDestination = startDestination
    ) {
        // Defines the splash screen.
        composable(SPLASH_ROUTE) {
            …
        }

        // Defines the login screen.
        composable(LOGIN_ROUTE) {
            …
        }

        // Defines the sign-up screen.
        composable(SIGNUP_ROUTE) {
            …
        }

        // Defines the profile information screen.
        composable(PROFILE_ROUTE) {
            …
        }

        // Defines the settings screen.
        composable(SETTINGS_ROUTE) {
            …
        }

        // Defines the patient-specific part of the application.
        composable(PATIENT_APP){
            ...
        }

        // Defines the admin-specific part of the application.
        composable(ADMIN_APP) {
            ...
        }

        // Defines the doctor-specific part of the application.
        composable(DOCTOR_APP) {
            ...

        }
    }
}
```

Then each of app will have their own role app with navHost

For the chatscreen:

Used firebase to store the messages and we are displaying them in chatitems specifically to each author. You are able to send an image or a document from your phone and it will display through a launcher when you click on it. You can also get referred to a different doctor from your current one to start a new chat with them.

```
package com.example.careconnect.screens.patient.chat

import android.app.NotificationManager
import android.content.Context
import android.content.Intent
import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.shape.RoundedCornerShape
```

```kotlin
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.automirrored.filled.OpenInNew
import androidx.compose.material.icons.automirrored.filled.Send
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Close
import androidx.compose.material.icons.filled.Description
import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.DropdownMenu
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TextField
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.constraintlayout.compose.Dimension
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.lifecycle.viewmodel.compose.viewModel
import coil.compose.AsyncImage
import com.example.careconnect.dataclass.Doctor
import com.example.careconnect.dataclass.Patient
import com.example.careconnect.dataclass.Role
import com.example.careconnect.dataclass.chat.ChatRoom
import com.example.careconnect.dataclass.chat.Message
import com.example.careconnect.screens.patient.home.HomeUiState
import com.example.careconnect.ui.theme.CareConnectTheme
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

/**
```

```
 * Data class representing a selected media item with its URI, type, and
optional name.
 *
 * @property uri The content URI of the selected media.
 * @property type The type of the media (IMAGE or DOCUMENT).
 * @property name The optional name of the media, used primarily for documents.
 */
data class SelectedMedia(
    val uri: Uri,
    val type: MediaType,
    val name: String = ""
)

/**
 * Enum representing supported media types for sending in chat.
 */
enum class MediaType {
    IMAGE, DOCUMENT
}

/**
 * Composable that displays the chat screen UI for a conversation between a
patient and a doctor.
 *
 * It handles loading and displaying doctor, patient, and chat room data,
manages notifications,
 * and integrates chat message list and message input UI.
 *
 * @param viewModel The [ChatViewModel] managing chat state and business logic.
 * @param chatId The unique identifier for the chat room.
 * @param patientId The ID of the patient participant in the chat.
 * @param doctorId The ID of the doctor participant in the chat.
 * @param openChatScreen Lambda function to open another chat screen with
provided chatId, patientId, and doctorId.
 * @param goBack Lambda function to handle back navigation.
 */
@Composable
fun ChatScreen(
    viewModel: ChatViewModel = hiltViewModel(),
    chatId: String,
    patientId: String,
    doctorId: String,
    openChatScreen: (String, String, String) -> Unit,
    goBack: () -> Unit = {}
){
    val context = LocalContext.current

    var doctor by remember { mutableStateOf<Doctor?>(null) }
    var patient by remember { mutableStateOf<Patient?>(null) }

    println("ChatScreen: chatId=$chatId, patientId=$patientId,
doctorId=$doctorId")

    LaunchedEffect(patientId, doctorId, chatId) {
        doctor = viewModel.getDoctor(doctorId)
        println("ChatScreen: doctor=$doctor")
        patient = viewModel.getPatient(patientId)
        println("ChatScreen: patient=$patient")
        viewModel.initialize(chatId, patientId, doctorId)

        val notificationManager =
context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        notificationManager.cancel(chatId.hashCode())
    }

    val chatRoom = viewModel.chatRoom

    // Pass only the necessary data to ChatScreenContent
    if (doctor != null && patient != null && chatRoom != null) {
        ChatScreenContent(
            model = viewModel,
            chatRoom = chatRoom,
            patient = patient!!,
            doctor = doctor!!,
            openChatScreen = openChatScreen,
            goBack = goBack
        )
    } else {
        println("ChatScreen: doctor=$doctor, patient=$patient,
chatRoom=$chatRoom")
    }
}

/**
 * Composable displaying the main content of the chat screen, including the
message list,
 * chat input box with media preview, and optional referral dialog.
 *
 * @param model The [ChatViewModel] containing chat data and actions.
 * @param chatRoom The [ChatRoom] data representing the current chat.
 * @param doctor The [Doctor] participant data.
 * @param patient The [Patient] participant data.
 * @param openChatScreen Lambda to open another chat screen.
 * @param goBack Lambda to navigate back.
 */
@Composable
fun ChatScreenContent(
    model: ChatViewModel,
    chatRoom: ChatRoom,
    doctor: Doctor,
    patient: Patient,
    openChatScreen: (String, String, String) -> Unit,
    goBack: () -> Unit = {}
) {

    val listState = rememberLazyListState()
    val messages by model.messages.collectAsStateWithLifecycle()

    println(" Composable sees ${messages.size} messages")

    LaunchedEffect(messages){
        println(" LaunchedEffect triggered with ${messages.size} messages")

        if (messages.isNotEmpty()){
            println("There are messages")
            listState.animateScrollToItem(messages.lastIndex)
        }
        else (println("No messages"))
    }

    Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {

        val currentUserId = model.me.id
        val chatName = model.chatRoom?.let {
            if (it.doctorId == currentUserId) {
                patient.name
            } else {
                doctor.name
            }
        }

        if (chatName != null) {
            SmallTopAppBarExample(
                name = chatName,
                goBack = goBack
            )
        }

        ConstraintLayout(modifier = Modifier.fillMaxSize()) {
            val (messageListRef, chatBox) = createRefs()

            if (model.showReferralDialog) {
                DoctorReferralDialog(
                    onDismiss = { model.showReferralDialog = false },
                    onDoctorSelected = { selectedDoctor ->
                        model.sendReferralMessage(selectedDoctor)
                        model.showReferralDialog = false
                    },
                    viewModel = model
                )
            }

            // Message List
            LazyColumn(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(top = 100.dp)
                    .constrainAs(messageListRef) {
                        top.linkTo(parent.top)
                        bottom.linkTo(chatBox.top)
                        start.linkTo(parent.start)
                        end.linkTo(parent.end)
                        height = Dimension.fillToConstraints
                    },
                state = listState
            ) {
                items(messages) { message ->
                    ChatItem(
                        message = message,
                        openNewChat = { newChatId, doctorId ->
                            openChatScreen(newChatId, patient.id, doctorId)
                        },
                        handleReferralClick = { referredDoctorId ->
```

```kotlin
model.handleReferralClick(referredDoctorId) }
                    )
                }
            }

            // ChatBox with media preview
            ChatBoxWithPreview(
                modifier = Modifier
                    .fillMaxWidth()
                    .constrainAs(chatBox) {
                        bottom.linkTo(parent.bottom)
                        start.linkTo(parent.start)
                        end.linkTo(parent.end)
                    },
                onSend = { text -> model.sendMessage(text, chatRoom.chatId) },
                onSendWithMedia = { text, media ->
                    when (media.type) {
                        MediaType.IMAGE -> {
                            model.sendImage(
                                media.uri,
                                message = Message(
                                    text = text.ifEmpty { "Image" },
                                    author = model.me,
                                    imageUrl = null
                                ),
                                chatId = chatRoom.chatId
                            )
                        }
                        MediaType.DOCUMENT -> {
                            model.sendDocument(
                                media.uri,
                                message = Message(
                                    text = text.ifEmpty { "Document" },
                                    author = model.me,
                                    documentUrl = null,
                                    documentName = media.name
                                ),
                                chatId = chatRoom.chatId
                            )
                        }
                    }
                },
                viewModel = model
            )
        }
    }
}

/**
 * Composable that provides the chat input box with optional media attachment
preview.
 *
 * Supports sending text messages, images, and documents. Manages media picking
via
 * activity results and dropdown menu for media selection and referrals.
 *
 * @param modifier Modifier to apply to this composable.
 * @param onSend Callback invoked with the typed text when sending a plain
message.
```

```kotlin
 * @param onSendWithMedia Callback invoked with typed text and selected media
when sending media.
 * @param viewModel The [ChatViewModel] used for managing chat state and user
info.
 */
@Composable
fun ChatBoxWithPreview(
    modifier: Modifier = Modifier,
    onSend: (String) -> Unit,
    onSendWithMedia: (String, SelectedMedia) -> Unit,
    viewModel: ChatViewModel = viewModel()
) {
    var expanded by remember { mutableStateOf(false) }
    var text by remember { mutableStateOf("") }
    var selectedMedia by remember { mutableStateOf<SelectedMedia?>(null) }

    val context = LocalContext.current

    // Image launcher
    val imageLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent()
    ) { uri: Uri? ->
        uri?.let {
            selectedMedia = SelectedMedia(
                uri = it,
                type = MediaType.IMAGE
            )
        }
        expanded = false
    }

    // Document launcher
    val documentLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent()
    ) { uri: Uri? ->
        uri?.let {
            val documentName = getDocumentName(context, it) ?: "Document"
            selectedMedia = SelectedMedia(
                uri = it,
                type = MediaType.DOCUMENT,
                name = documentName
            )
        }
        expanded = false
    }

    Column(modifier = modifier) {
        // Media Preview Section
        selectedMedia?.let { media ->
            MediaPreviewCard(
                media = media,
                onRemove = { selectedMedia = null }
            )
        }

        // Chat Input Row
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp)
                .background(MaterialTheme.colorScheme.surface,
RoundedCornerShape(24.dp))
                .padding(horizontal = 8.dp, vertical = 4.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            // Plus (+) button for dropdown menu
            Box {
                IconButton(
                    onClick = { expanded = true }
                ) {
                    Icon(Icons.Filled.Add, contentDescription = "Attachments")
                }

                MinimalDropdownMenu(
                    expanded = expanded,
                    onDismissRequest = { expanded = false },
                    onImageSend = { imageLauncher.launch("image/*") },
                    onDocumentSend = { documentLauncher.launch("application/*")
},
                    onReferralSend = if (viewModel.me.role == Role.DOCTOR) {
                        { viewModel.showReferralDialog = true }
                    } else null
                )
            }

            TextField(
                value = text,
                onValueChange = { text = it },
                modifier = Modifier.weight(1f),
                placeholder = {
                    Text(
                        if (selectedMedia != null) "Add a message (optional)..."
                        else "Type a message..."
                    )
                },
                colors = TextFieldDefaults.colors(
                    unfocusedIndicatorColor = Color.Transparent,
                    focusedIndicatorColor = Color.Transparent
                )
            )

            // Send button
            IconButton(
                onClick = {
                    selectedMedia?.let { media ->
                        onSendWithMedia(text, media)
                        selectedMedia = null
                        text = ""
                    } ?: run {
                        if (text.isNotBlank()) {
                            onSend(text)
                            text = ""
                        }
                    }
                },
                enabled = selectedMedia != null || text.isNotBlank()
            ) {
                Icon(Icons.AutoMirrored.Filled.Send, contentDescription =
```

```kotlin
"Send")
            }
        }
    }
}

/**
 * Displays a preview card for the selected media (image or document) in the
chat input area.
 *
 * @param media The selected media to preview.
 * @param onRemove Callback invoked when the user removes the selected media.
 */
@Composable
fun MediaPreviewCard(
    media: SelectedMedia,
    onRemove: () -> Unit
) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 8.dp, vertical = 4.dp),
        colors = CardDefaults.cardColors(
            containerColor = MaterialTheme.colorScheme.surfaceVariant
        )
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(12.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            when (media.type) {
                MediaType.IMAGE -> {
                    AsyncImage(
                        model = media.uri,
                        contentDescription = "Selected Image",
                        modifier = Modifier
                            .size(60.dp)
                            .clip(RoundedCornerShape(8.dp)),
                        contentScale = ContentScale.Crop
                    )
                    Spacer(modifier = Modifier.width(12.dp))
                    Text(
                        text = "Image selected",
                        modifier = Modifier.weight(1f),
                        style = MaterialTheme.typography.bodyMedium
                    )
                }
                MediaType.DOCUMENT -> {
                    Icon(
                        imageVector = Icons.Default.Description,
                        contentDescription = "Document",
                        modifier = Modifier.size(40.dp),
                        tint = MaterialTheme.colorScheme.primary
                    )
                    Spacer(modifier = Modifier.width(12.dp))
                    Column(modifier = Modifier.weight(1f)) {
                        Text(
                            text = media.name,
                            style = MaterialTheme.typography.bodyMedium,
                            maxLines = 1,
                            overflow = TextOverflow.Ellipsis
                        )
                        Text(
                            text = "Document selected",
                            style = MaterialTheme.typography.bodySmall,
                            color = MaterialTheme.colorScheme.onSurfaceVariant
                        )
                    }
                }
            }

            IconButton(onClick = onRemove) {
                Icon(
                    Icons.Default.Close,
                    contentDescription = "Remove",
                    tint = MaterialTheme.colorScheme.error
                )
            }
        }
    }
}

// Helper function to get document name from URI
/**
 * Retrieves the display name of a document from its [Uri].
 *
 * @param context The context to access content resolver.
 * @param uri The Uri of the document.
 * @return The display name of the document if available, or the last path
segment of the Uri.
 */
fun getDocumentName(context: Context, uri: Uri): String? {
    return try {
        context.contentResolver.query(uri, null, null, null, null)?.use { cursor
->
            val nameIndex =
cursor.getColumnIndex(android.provider.OpenableColumns.DISPLAY_NAME)
            if (nameIndex >= 0 && cursor.moveToFirst()) {
                cursor.getString(nameIndex)
            } else null
        }
    } catch (e: Exception) {
        uri.lastPathSegment
    }
}

/**
 * Displays an individual chat message item in the message list.
 * Handles different message types including text, image, document, and referral
messages.
 *
 * @param message The message data to display.
 * @param openNewChat Callback to open a new chat screen with given chat and
doctor IDs.
 * @param handleReferralClick Suspend function to handle referral clicks,
returning the chat and doctor IDs.
```

```
 */
@Composable
fun ChatItem(
    message: Message,
    openNewChat: (String, String) -> Unit,
    handleReferralClick: suspend (String) -> Pair<String, String>?,
) {

    val coroutineScope = rememberCoroutineScope()

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        horizontalArrangement = if (message.isFromMe) Arrangement.End else
Arrangement.Start,
        verticalAlignment = Alignment.Bottom
    ) {
        if (message.isFromMe) {
            Text(
                text = formatTimestamp(message.timestamp),
                color = Color.Gray,
                style = MaterialTheme.typography.bodySmall,
                modifier = Modifier.padding(end = 4.dp)
            )
        }

        Box(
            modifier = Modifier
                .clip(
                    RoundedCornerShape(
                        topStart = 16.dp,
                        topEnd = 16.dp,
                        bottomStart = if (message.isFromMe) 16.dp else 0.dp,
                        bottomEnd = if (message.isFromMe) 0.dp else 16.dp
                    )
                )
                .background(if (message.isFromMe)
MaterialTheme.colorScheme.primary else MaterialTheme.colorScheme.surfaceVariant)
                .padding(12.dp)
        ) {
            when (message.metadata?.get("messageType")) {
                "referral" -> {
                    ReferralMessageItem(
                        message = message,
                        onReferralClick = { referralDoctorId ->
                            coroutineScope.launch {
                                val result =
handleReferralClick(referralDoctorId)
                                result?.let { (chatId, referredDoctorId) ->
                                    openNewChat(chatId, referredDoctorId)
                                }
                            }
                        }
                    )
                }
                "referral_intro" -> {
                    ReferralIntroMessageItem(message = message)
                }
```

```kotlin
                else -> {
                    Column {
                        if (message.text.isNotEmpty()) {
                            Text(
                                text = message.text,
                                color = if (message.isFromMe) Color.White else
Color.Black
                            )
                        }

                        message.imageUrl?.let { url ->
                            if (message.text.isNotEmpty()) {
                                Spacer(modifier = Modifier.height(8.dp))
                            }
                            ImagePreview(model = url)
                        }

                        message.documentUrl?.let { url ->
                            if (message.text.isNotEmpty() || message.imageUrl !=
null) {
                                Spacer(modifier = Modifier.height(8.dp))
                            }
                            DocumentPreview(
                                documentName = message.documentName ?:
"Document",
                                documentUrl = url,
                                isFromMe = message.isFromMe
                            )
                        }
                    }
                }
            }
        }

        if (!message.isFromMe) {
            Text(
                text = formatTimestamp(message.timestamp),
                color = Color.Gray,
                style = MaterialTheme.typography.bodySmall,
                modifier = Modifier.padding(start = 4.dp)
            )
        }
    }
}

/**
 * Shows an image preview inside the chat message.
 * Clicking the image opens it in an external viewer.
 *
 * @param model The URL or Uri string of the image to display.
 */
@Composable
fun ImagePreview(model : String){
    val context = LocalContext.current

    AsyncImage(
        model = model,
        contentDescription = "Sent Image",
        modifier = Modifier
```

```kotlin
            .size(200.dp)
            .clip(RoundedCornerShape(8.dp))
            .clickable {
                val intent = Intent(Intent.ACTION_VIEW).apply {
                    setDataAndType(Uri.parse(model), "*/*")
                    addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
                }
                context.startActivity(intent)
            },
        contentScale = ContentScale.Crop,
    )
}

/**
 * Shows a preview card for a document attached to a chat message.
 * Includes the document name and a button to open the document.
 *
 * @param documentName The display name of the document.
 * @param documentUrl The URL or Uri string of the document.
 * @param isFromMe Indicates if the document was sent by the current user.
 */
@Composable
fun DocumentPreview(documentName: String, documentUrl: String, isFromMe:
Boolean) {
    val context = LocalContext.current
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .height(80.dp),
        colors = CardDefaults.cardColors(
            containerColor = if (isFromMe)
MaterialTheme.colorScheme.primaryContainer else
MaterialTheme.colorScheme.surface
        )
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .fillMaxSize()
                .padding(horizontal = 12.dp)
        ) {
            Icon(
                imageVector = Icons.Default.Description,
                contentDescription = "Document Icon",
                tint = MaterialTheme.colorScheme.onSurface,
                modifier = Modifier.size(32.dp)
            )
            Spacer(modifier = Modifier.width(12.dp))
            Text(
                text = documentName,
                modifier = Modifier.weight(1f),
                maxLines = 1,
                overflow = TextOverflow.Ellipsis,
                color = MaterialTheme.colorScheme.onSurface
            )
            IconButton(onClick = {
                val intent = Intent(Intent.ACTION_VIEW).apply {
                    setDataAndType(Uri.parse(documentUrl), "*/*")
                    addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
                }
                context.startActivity(intent)
            }) {
                Icon(Icons.AutoMirrored.Filled.OpenInNew, contentDescription =
"Open Document")
            }
        }
    }
}

/**
 * Formats a timestamp (in milliseconds) into a human-readable time string.
 *
 * @param timestamp The timestamp to format.
 * @return A formatted time string (e.g., "02:30 PM").
 */
fun formatTimestamp(timestamp: Long): String {
    val sdf = SimpleDateFormat("hh:mm a", Locale.getDefault())
    return sdf.format(Date(timestamp))
}

/**
 * A small top app bar composable with a title and back navigation button.
 *
 * @param name The title to display in the app bar.
 * @param goBack Callback invoked when the back button is pressed.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SmallTopAppBarExample(
    name: String = "Chat",
    goBack: () -> Unit = {}
) {
    Scaffold(
        topBar = {
            TopAppBar(
                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor = MaterialTheme.colorScheme.primary,
                    titleContentColor = MaterialTheme.colorScheme.primary,
                ),
                title = {
                    Text(
                        text = name,
                        style = MaterialTheme.typography.titleLarge,
                        color = MaterialTheme.colorScheme.onPrimary
                    )
                },
                navigationIcon = {
                    IconButton(onClick = { goBack() }) {
                        Icon(
                            tint = MaterialTheme.colorScheme.onPrimary,
                            imageVector = Icons.AutoMirrored.Filled.ArrowBack,
                            contentDescription = "Localized description"
                        )
                    }
                },
            )
        },
    ){
        Box(modifier = Modifier.padding(it))
    }
}

/**
 * Dialog composable that allows a doctor user to refer the patient to another
doctor.
 * Displays a list of available doctors excluding the current doctor.
 *
 * @param onDismiss Callback invoked when the dialog is dismissed.
 * @param onDoctorSelected Callback invoked when a doctor is selected for
referral.
 * @param viewModel The ChatViewModel used to retrieve the doctors list.
 */
@Composable
fun DoctorReferralDialog(
    onDismiss: () -> Unit,
    onDoctorSelected: (Doctor) -> Unit,
    viewModel: ChatViewModel
) {
    var doctorList by remember { mutableStateOf<List<Doctor>>(emptyList()) }

    LaunchedEffect(Unit) {
        doctorList = viewModel.getAllDoctors()
            .filter { it.id != viewModel.doctorId }
    }

    AlertDialog(
        onDismissRequest = onDismiss,
        title = { Text("Refer to Another Doctor") },
        text = {
            Column {
                doctorList.forEach { doctor ->
                    TextButton(onClick = { onDoctorSelected(doctor) }) {
                        Text("${doctor.name} - ${doctor.specialization}")
                    }
                }
            }
        },
        confirmButton = {},
        dismissButton = {
            TextButton(onClick = onDismiss) { Text("Cancel") }
        }
    )
}

/**
 * A minimal dropdown menu for selecting media attachment options in the chat
input.
 * Options include sending images, documents, and optionally referrals.
 *
 * @param expanded Whether the dropdown menu is expanded or not.
 * @param onDismissRequest Callback invoked to dismiss the dropdown menu.
 * @param onImageSend Callback invoked when the "Send image" option is selected.
 * @param onDocumentSend Callback invoked when the "Send document" option is
selected.
 * @param onReferralSend Optional callback invoked when the "Send referral"
option is selected.
 */
```

```kotlin
@Composable
fun MinimalDropdownMenu(
    expanded: Boolean,
    onDismissRequest: () -> Unit,
    onImageSend: () -> Unit,
    onDocumentSend: () -> Unit,
    onReferralSend: (() -> Unit)? = null
) {
    Box(
        modifier = Modifier.padding(16.dp)
    ) {
        DropdownMenu(
            expanded = expanded,
            onDismissRequest = onDismissRequest
        ) {
            DropdownMenuItem(
                text = { Text("Send image") },
                onClick = onImageSend
            )
            DropdownMenuItem(
                text = { Text("Send document") },
                onClick = onDocumentSend
            )
            onReferralSend?.let {
                DropdownMenuItem(
                    text = { Text("Send referral") },
                    onClick = it
                )
            }
        }
    }
}

/**
 * Preview composable for ChatScreenContent showcasing chat UI with dummy data.
 */
@Preview
@Composable
fun ChatScreenPreview() {
    CareConnectTheme {
        val uiState = HomeUiState()
        ChatScreenContent(
            model = viewModel(),
            chatRoom = ChatRoom(),
            doctor = Doctor(),
            patient = Patient(),
            openChatScreen = { doctorId, patientId, chatId ->
                println("Opening chat with doctor ID: $doctorId and chat ID:
$chatId")
            }
        )
    }
}
```

Appointment booking:

Here the patient is able to select a date and a timeslot from a selected doctor and they can then book the appointment. The data will then be displayed in the home screen and appointment screen via firebase.

```kotlin
package com.example.careconnect.screens.patient.appointment

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.ExperimentalLayoutApi
import androidx.compose.foundation.layout.FlowRow
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.Button
import androidx.compose.material3.DatePicker
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.FilterChip
import androidx.compose.material3.FilterChipDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SelectableDates
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.material3.rememberDatePickerState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.scale
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.careconnect.common.LoadingIndicator
import com.example.careconnect.dataclass.Doctor
import com.example.careconnect.dataclass.SnackBarMessage
import com.example.careconnect.dataclass.TimeSlot
import com.example.careconnect.ui.theme.CareConnectTheme
import java.time.Instant
import java.time.LocalDate
import java.time.LocalTime
import java.time.ZoneId
```

```kotlin
/**
 * Main Composable to display the Book Appointment screen.
 *
 * @param doctorId The unique identifier of the doctor for whom the appointment
is being booked.
 * @param viewModel The ViewModel managing appointment data and UI state.
 * @param showSnackBar A lambda callback to display snack bar messages.
 * @param goBack A lambda callback to navigate back from the screen.
 */
@Composable
fun BookAppointmentScreen(
    doctorId: String,
    viewModel: BookAppointmentViewModel = hiltViewModel(),
    showSnackBar: (SnackBarMessage) -> Unit,
    goBack: () -> Unit = {}
){

    LaunchedEffect(doctorId) {
        viewModel.setDoctorId(doctorId)
    }

    val doctor by viewModel.doctor.collectAsStateWithLifecycle()
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()

    BookAppointmentScreenContent(
        doctor = doctor,
        uiState = uiState,
        onDateSelected = { date ->
            viewModel.onDateSelected(date, showSnackBar)
        },
        onTimeSelected = { timeSlot ->
            viewModel.onTimeSelected(timeSlot)
        },
        onBookAppointment = {
            viewModel.bookAppointment(showSnackBar)
        },
        goBack = goBack
    )
}

/**
 * The UI content of the Book Appointment screen, displaying doctor info,
 * date picker, available time slots, and booking button.
 *
 * @param doctor The [Doctor] object whose appointment is being booked.
 * @param uiState The current UI state of the booking screen.
 * @param onDateSelected Callback when a new date is selected.
 * @param onTimeSelected Callback when a time slot is selected.
 * @param onBookAppointment Callback to trigger booking action.
 * @param goBack Callback to navigate back.
 */
@Composable
fun BookAppointmentScreenContent(
    doctor: Doctor? = Doctor(),
    uiState: BookAppointmentUiState,
    onDateSelected: (LocalDate) -> Unit,
    onTimeSelected: (TimeSlot) -> Unit,
    onBookAppointment: () -> Unit,
    goBack: () -> Unit = {}
) {
    Scaffold(
        topBar = {
            BookAppointmentTopBar(
                goBack = goBack
            )
        }
    ) { paddingValues ->
        Column(
            modifier =
Modifier.padding(paddingValues).verticalScroll(rememberScrollState()),
            horizontalAlignment = Alignment.CenterHorizontally,
        ) {
            doctor?.let {
                Text(
                    text = "Dr. ${it.name} ${it.surname}",
                    style = MaterialTheme.typography.headlineSmall,
                    modifier = Modifier
                        .align(Alignment.Start)
                        .padding(start = 20.dp, top = 16.dp)
                )

                Text(
                    text = it.specialization,
                    style = MaterialTheme.typography.bodyLarge,
                    color = MaterialTheme.colorScheme.onSurfaceVariant,
                    modifier = Modifier
                        .align(Alignment.Start)
                        .padding(start = 20.dp, bottom = 16.dp)
                )

                InlineDatePicker { selectedDate ->
                    onDateSelected(Instant.ofEpochMilli(selectedDate)
                        .atZone(ZoneId.systemDefault())
                        .toLocalDate())
                }

                // Time slots
                Column(modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 20.dp)) {
                    Text(
                        text = "Available Time Slots",
                        style = MaterialTheme.typography.titleMedium,
                        modifier = Modifier.padding(bottom = 8.dp)
                    )

                    when {
                        uiState.isLoading -> LoadingIndicator()
                        uiState.availableSlots.isEmpty() -> NoSlotsMessage()
                        else -> TimeSelectionSection(
                            slots = uiState.availableSlots,
                            selectedTimeSlot = uiState.selectedTimeSlot,
                            selectedDate = uiState.selectedDate,
                            onTimeSelected = onTimeSelected
                        )
                    }
                }
                Spacer(modifier = Modifier.height(16.dp))

                Button(onClick = onBookAppointment) {
                    Text("Book an Appointment",
                        style = MaterialTheme.typography.titleLarge)
                }
            }

        }



    }
}

/**
 * Displays a section with selectable time slots as chips.
 *
 * @param slots List of available [TimeSlot]s.
 * @param selectedTimeSlot Currently selected time slot.
 * @param selectedDate The selected date.
 * @param onTimeSelected Callback triggered when a time slot is selected.
 */
@Composable
private fun TimeSelectionSection(
    slots: List<TimeSlot>,
    selectedTimeSlot: TimeSlot?,
    selectedDate: LocalDate,
    onTimeSelected: (TimeSlot) -> Unit
) {
    TimeSelectionChips(
        availableTimeSlots = slots,
        onTimeSelected = onTimeSelected,
        selectedDate = selectedDate,
        selectedTimeSlot = selectedTimeSlot,
    )
}

/**
 * Shows a message when there are no available time slots.
 */
@Composable
private fun NoSlotsMessage() {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 16.dp),
        contentAlignment = Alignment.Center
    ) {
        Text(
            text = "No available time slots for this date",
            style = MaterialTheme.typography.bodyMedium,
            color = MaterialTheme.colorScheme.onSurfaceVariant
        )
    }
}

/**
```

```
 * Top app bar for the Book Appointment screen, includes a back button and
title.
 *
 * @param goBack Callback triggered when the back button is pressed.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BookAppointmentTopBar(
    goBack: () -> Unit
) {

            TopAppBar(
                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor = MaterialTheme.colorScheme.primary,
                    titleContentColor = MaterialTheme.colorScheme.onPrimary,
                ),
                title = {
                    Text(
                        "Request Appointment",
                        style = MaterialTheme.typography.titleLarge
                    )
                },
                navigationIcon = {
                    IconButton(onClick = { /* do something */ }) {
                        Icon(
                            tint = MaterialTheme.colorScheme.onPrimary,
                            imageVector = Icons.AutoMirrored.Filled.ArrowBack,
                            contentDescription = "Localized description"
                        )
                    }
                },
            )
}

/**
 * Inline date picker Composable to select appointment dates.
 * Restricts selection to today and future dates, within 2 years.
 *
 * @param onDateSelected Callback returning the selected date in milliseconds.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun InlineDatePicker(onDateSelected: (Long) -> Unit) {
    val selectableDates = object : SelectableDates {
        override fun isSelectableDate(utcTimeMillis: Long): Boolean {
            val selectedDate = Instant.ofEpochMilli(utcTimeMillis)
                .atZone(ZoneId.systemDefault())
                .toLocalDate()
            return !selectedDate.isBefore(LocalDate.now())
        }

        override fun isSelectableYear(year: Int): Boolean {
            val currentYear = LocalDate.now().year
            return year >= currentYear && year <= currentYear + 2
        }
    }
    val datePickerState = rememberDatePickerState(
        initialSelectedDateMillis = System.currentTimeMillis(),
        yearRange = IntRange(LocalDate.now().year, LocalDate.now().year + 2),
        selectableDates = selectableDates
    )

    LaunchedEffect(datePickerState.selectedDateMillis) {
        datePickerState.selectedDateMillis?.let { selectedDate ->
            onDateSelected(selectedDate)
        }
    }
    Column(
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Box(
            modifier = Modifier
                .size(width = 500.dp, height = 500.dp) // Adjust size
                .scale(0.9f) // Scale down the DatePicker
        ) {
            DatePicker(state = datePickerState)
        }
        Spacer(modifier = Modifier.height(16.dp))

    }
}

/**
 * Displays a list of selectable chips representing available time slots.
 *
 * Disabled if the slot is in the past or unavailable.
 *
 * @param availableTimeSlots List of time slots.
 * @param selectedTimeSlot Currently selected time slot.
 * @param selectedDate The date for which slots are shown.
 * @param onTimeSelected Callback when a time slot is selected.
 */
@OptIn(ExperimentalMaterial3Api::class, ExperimentalLayoutApi::class)
@Composable
fun TimeSelectionChips(
    availableTimeSlots: List<TimeSlot>,
    selectedTimeSlot: TimeSlot?,
    selectedDate: LocalDate?,
    onTimeSelected: (TimeSlot) -> Unit
) {
    val currentTime = LocalTime.now()
    val currentDate = LocalDate.now()

    FlowRow(
        modifier = Modifier
            .fillMaxWidth()
            .padding(start = 20.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp),
        maxItemsInEachRow = 4 // Limit to 4 chips per row
    ) {
        availableTimeSlots.forEach { slot ->
            val timeRange = "${slot.startTime} - ${slot.endTime}"
            val isSelected = selectedTimeSlot == slot
            val isPastTime = selectedDate == currentDate &&
LocalTime.parse(slot.startTime) < currentTime

            val isAvailable = slot.available && !isPastTime
            Box(
                modifier = Modifier
                    .width(80.dp)
            ) {
                FilterChip(
                    selected = isSelected,
                    onClick = { if (isAvailable) onTimeSelected(slot) },
                    enabled = isAvailable,
                    label = {
                        Text(
                            text = timeRange,
                            style = MaterialTheme.typography.bodySmall,
                            color = when {
                                !isAvailable ->
MaterialTheme.colorScheme.onSurfaceVariant
                                isSelected ->
MaterialTheme.colorScheme.onPrimary
                                else -> MaterialTheme.colorScheme.onSurface
                            }
                        )
                    },
                    colors = FilterChipDefaults.filterChipColors(
                        labelColor = when {
                            !isAvailable ->
MaterialTheme.colorScheme.onSurfaceVariant
                            isSelected -> MaterialTheme.colorScheme.onPrimary
                            else -> MaterialTheme.colorScheme.onSurface
                        },
                        disabledContainerColor =
MaterialTheme.colorScheme.surfaceVariant,
                        disabledLabelColor =
MaterialTheme.colorScheme.onSurfaceVariant
                    ),
                    modifier = Modifier.fillMaxWidth()
                )
            }

        }
    }
}

/**
 * Preview of the Book Appointment screen content for design and testing
purposes.
 */
@Preview
@Composable
fun BookAppointmentScreenPreview() {
    CareConnectTheme {
        val uiState = BookAppointmentUiState(
            availableSlots = listOf(
                TimeSlot(
                    startTime = "10:00",
                    endTime = "11:00",
                    available = true,
                ),
                TimeSlot(
                    startTime = "11:00",
                    endTime = "12:00",
                    available = true,
                ),
                TimeSlot(
                    startTime = "12:00",
                    endTime = "13:00",
                    available = false,
                )
            )
        )
        BookAppointmentScreenContent(
            doctor = Doctor(
                name = "John",
                surname = "Doe",
                address = "123 Main St",
                specialization = "Family Medicine",
            ),
            uiState = uiState,
            onDateSelected = {},
            onTimeSelected = {},
            onBookAppointment = {}
        )
    }
}
```

Adding the doctor screen:

In here the admin is able to enter some parameters for the doctor and create the new user. They will then also be able to set their schedule for them for any amount of time. We store the information in firebase by usage of cloud functions to create the new user.

```
package com.example.careconnect.screens.admin.doctormanage

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowForward
import androidx.compose.material.icons.filled.Check
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.ExposedDropdownMenuBox
import androidx.compose.material3.ExposedDropdownMenuDefaults
import androidx.compose.material3.Icon
```

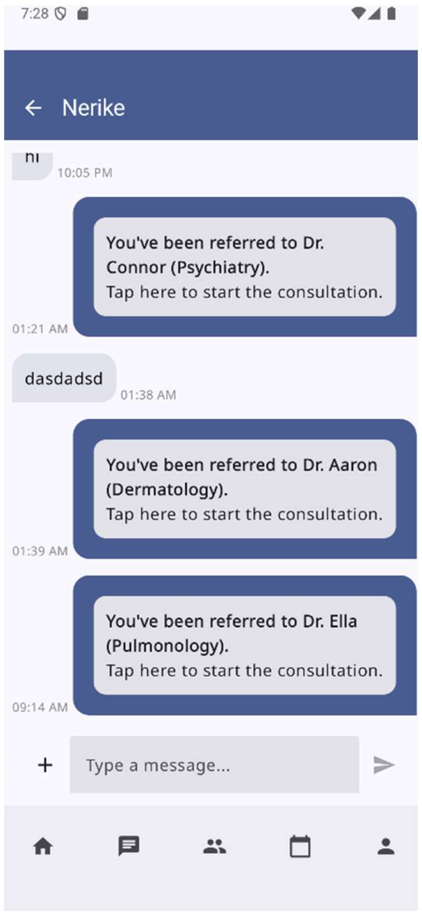
```kotlin
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.careconnect.R
import com.example.careconnect.dataclass.SnackBarMessage
import com.example.careconnect.dataclass.Specialization
import com.example.careconnect.ui.theme.CareConnectTheme

/**
 * Screen to add a new doctor by inputting personal and professional details.
 *
 * Handles navigation to the doctor schedule screen after successful creation.
 *
 * @param openDoctorScheduleScreen Lambda invoked with the created doctor's ID
to navigate to the schedule setup.
 * @param showSnackBar Lambda to show feedback messages as snack bars.
 * @param viewModel ViewModel responsible for managing doctor creation state.
 */
@Composable
fun AddDoctorScreen(
    openDoctorScheduleScreen: (doctorId: String) -> Unit,
    showSnackBar: (SnackBarMessage) -> Unit,
    viewModel: AddDoctorViewModel = hiltViewModel()
){
    val doctorId by viewModel.newDoctorId.collectAsStateWithLifecycle()

    LaunchedEffect(doctorId) {
        doctorId?.let {
            openDoctorScheduleScreen(it)
        }
    }
    AddDoctorScreenContent(
        createDoctorInfo = viewModel::createDoctorInfo,
        showSnackBar = showSnackBar
    )
}


/**
 * UI content for adding doctor information.
 *
 * Presents a form with fields for doctor's name, surname, email,
specialization, experience, address,
 * phone number, and password. Contains a stepper indicator for progress.
 *
 * @param createDoctorInfo Callback invoked with doctor info and a snack bar
callback when user submits.
 * @param showSnackBar Callback to display snack bar messages, default empty.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddDoctorScreenContent(
    createDoctorInfo: (String, String, String, String, String, String, String,
String, (SnackBarMessage) -> Unit) -> Unit,
    showSnackBar: (SnackBarMessage) -> Unit = {}
) {

    var name by remember { mutableStateOf("") }
    var surname by remember { mutableStateOf("") }
    var specialization by remember { mutableStateOf("") }
    var address by remember { mutableStateOf("") }
    var phone by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var experience by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }

    Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {

        AdminTopAppBar(
            label = stringResource(R.string.add_doctor),
            onBack = {}
        )

        Column(modifier = Modifier.padding(top = 80.dp)) {

            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {

            }
            Column(modifier = Modifier
                .fillMaxSize()
                .padding(16.dp)) {
                // Progress Stepper
                StepperIndicator(currentStep = 1)

                Spacer(modifier = Modifier.height(24.dp))

                // Form for Personal Info
                Text(
                    "Step 1: Personal Information",
                    style = MaterialTheme.typography.headlineMedium
                )
                Spacer(modifier = Modifier.height(16.dp))

                var specializationExpanded by remember { mutableStateOf(false) }
                val specializationOptions = Specialization.all()
                var selectedSpecialization by remember {
mutableStateOf<Specialization?>(null) }


                /**
                 * TextFields for doctor information
                 * Validation in viewModel
                 * TODO(): Create drop down menu for specialization - instead of
typing
                 * TODO(): Generate a random password that is strong
                 * TODO(): Send the password to the email (of the doctor)
                 */
                CustomTextField(label = stringResource(R.string.name), value =
name, onValueChange = { name = it })
                CustomTextField(label = stringResource(R.string.surname), value
= surname, onValueChange = { surname = it })
                CustomTextField(label = stringResource(R.string.email), value =
email, onValueChange = { email = it })
                ExposedDropdownMenuBox(
                    expanded = specializationExpanded,
                    onExpandedChange = { specializationExpanded =
!specializationExpanded }
                ) {
                    OutlinedTextField(
                        value = selectedSpecialization?.displayName() ?: "",
                        onValueChange = {},
                        readOnly = true,
                        label = { Text("Specialization") },
                        trailingIcon = {
ExposedDropdownMenuDefaults.TrailingIcon(expanded = specializationExpanded) },
                        modifier = Modifier
                            .menuAnchor()
                            .fillMaxWidth()
                    )

                    ExposedDropdownMenu(
                        expanded = specializationExpanded,
                        onDismissRequest = { specializationExpanded = false }
                    ) {
                        specializationOptions.forEach { spec ->
                            DropdownMenuItem(
                                text = { Text(spec.displayName()) },
                                onClick = {
                                    selectedSpecialization = spec
                                    specializationExpanded = false
                                }
                            )
                        }
                    }
                }
                CustomTextField(
                    label = stringResource(R.string.experience),
                    value = experience,
                    onValueChange = { if (it.all { ch -> ch.isDigit() })
experience = it },
                    keyboardType = KeyboardType.Number
                )
                CustomTextField(label = stringResource(R.string.address), value
= address, onValueChange = { address = it })
                CustomTextField(
                    label = stringResource(R.string.phone),
                    value = phone,
                    onValueChange = {
                        if (it.length <= 9 && it.all { ch -> ch.isDigit() })
phone = it
                    },
                    keyboardType = KeyboardType.Phone
                )
                CustomTextField(label = stringResource(R.string.password), value
= password, onValueChange = { password = it })

                Spacer(modifier = Modifier.height(24.dp))

                val isFormValid = name.isNotBlank()
                        && surname.isNotBlank()
                        && email.isNotBlank()
                        && selectedSpecialization != null
                        && experience.isNotBlank()
                        && address.isNotBlank()
                        && phone.length == 9
                        && password.isNotBlank()

                // Next Button
                IconButton(
                    onClick = {
                        selectedSpecialization?.let {
                            createDoctorInfo(
                                name,
                                surname,
                                email,
                                phone,
                                address,
                                it.name, // Enum name
                                experience,
                                password,
                                showSnackBar
                            )
                        }
                    },
                    modifier = Modifier.align(Alignment.End),
                    enabled = isFormValid // ✅ disables button when form is
incomplete
                ) {
                    Icon(Icons.AutoMirrored.Filled.ArrowForward,
contentDescription = "Next Step")
                }
            }
        }
    }
}

// Progress Stepper UI
/**
 * Displays a horizontal stepper indicator showing progress in doctor creation.
 *
 * @param currentStep Current step number (1-based index).
 */
@Composable
fun StepperIndicator(currentStep: Int) {
    val steps = listOf("Info", "Schedule")

    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        steps.forEachIndexed { index, step ->
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Box(
                    modifier = Modifier
                        .size(40.dp)
                        .padding(4.dp),
                    contentAlignment = Alignment.Center
                ) {
                    Surface(
                        shape = CircleShape,
                        color = if (index + 1 <= currentStep)
MaterialTheme.colorScheme.primary else Color.Gray,
                        modifier = Modifier.size(40.dp)
                    ) {
                        Box(contentAlignment = Alignment.Center) {
                            if (index + 1 < currentStep) {
                                Icon(Icons.Default.Check, contentDescription =
"Completed", tint = Color.White)
                            } else {
                                Text(text = "${index + 1}", color = Color.White,
fontSize = 18.sp)
                            }
                        }

                    }
                }
                Text(text = step, fontSize = 14.sp)
            }
        }
    }
}

// Custom TextField
/**
 * A custom styled outlined text field used in doctor form inputs.
 *
 * @param label Label text describing the field.
 * @param value Current value of the text field.
 * @param onValueChange Callback triggered when the user changes the input text.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CustomTextField(label: String, value: String, onValueChange: (String) ->
Unit,keyboardType: KeyboardType = KeyboardType.Text) {
    OutlinedTextField(
        value = value,
        onValueChange = onValueChange,
        label = { Text(label) },
        modifier = Modifier.fillMaxWidth(),
        keyboardOptions = KeyboardOptions.Default.copy(keyboardType =
keyboardType)
    )
}


@Preview
@Composable
fun AddDoctorScreenPreview() {
    CareConnectTheme {
        AddDoctorScreenContent(
            createDoctorInfo = { _, _, _, _, _, _, _, _, _->},
            showSnackBar = {}
        )
    }
}
```

Feature in progress

- Admin:
  - Send tasks, email (with friendly UI) to doctors
  - Create fcm for doctors or patients
  - Able to modify doctor's data
- App:
  - Search or filter by address for doctors, appointments or consultation
  - Review about the doctors
- Authentication:
  - Verify email when changed, send email to doctor and patient when their account created.
  - Reset password

Doctor Screens:

A medical professional who can update their profile information, set their availability for appointments, view scheduled consultations, and interact with patients via chat during e-consultations.


7:28

Nerike

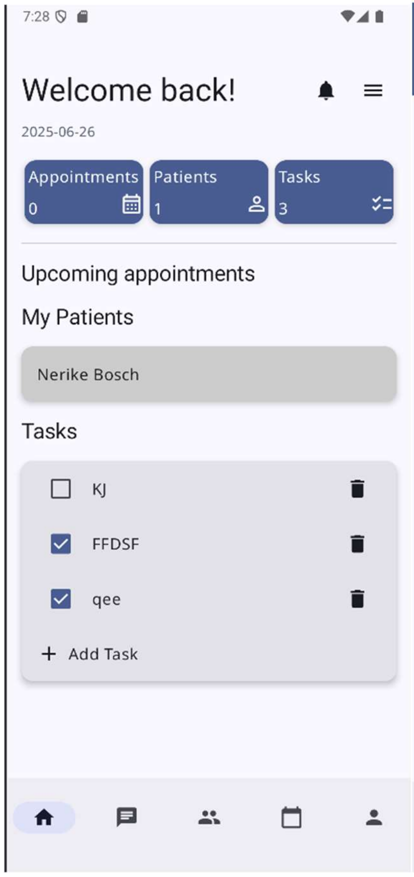
hi 10:05 PM

You've been referred to Dr. Connor (Psychiatry).
Tap here to start the consultation.
01:21 AM

dasdadsd 01:38 AM

You've been referred to Dr. Aaron (Dermatology).
Tap here to start the consultation.
01:39 AM

You've been referred to Dr. Ella (Pulmonology).
Tap here to start the consultation.
09:14 AM


7:28

Welcome back!

2025-06-26

Appointments	Patients	Tasks
0	1	3

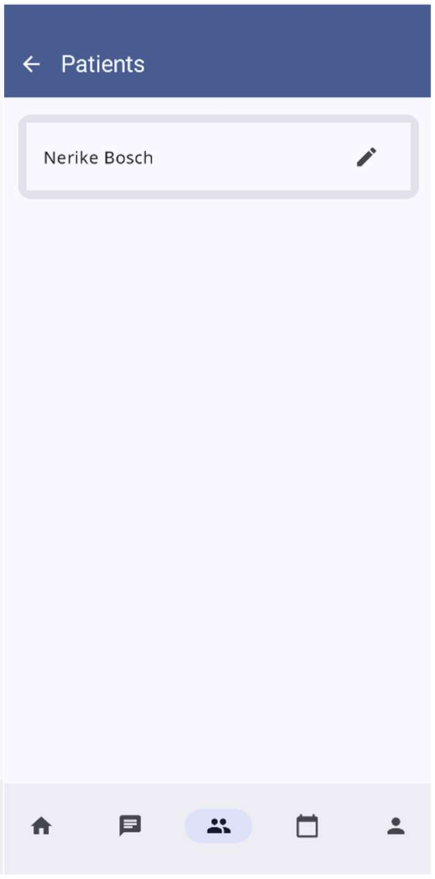
Upcoming appointments

My Patients

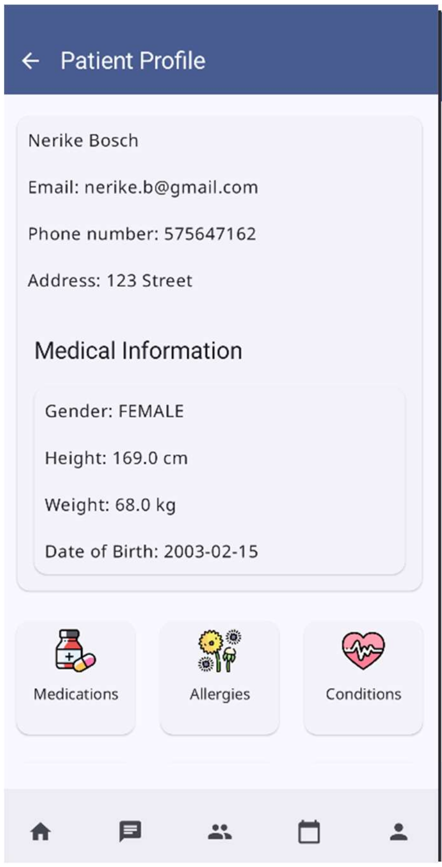
Nerike Bosch

Tasks

- [ ] KJ
- [x] FFDSF
- [x] qee


Patients

Nerike Bosch



Nerike Bosch

Email: nerike.b@gmail.com

Phone number: 575647162

Address: 123 Street

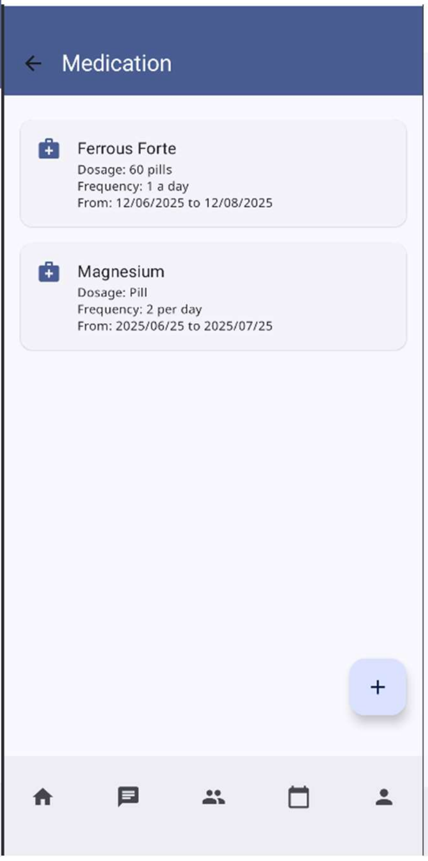
Medical Information

Gender: FEMALE

Height: 169.0 cm

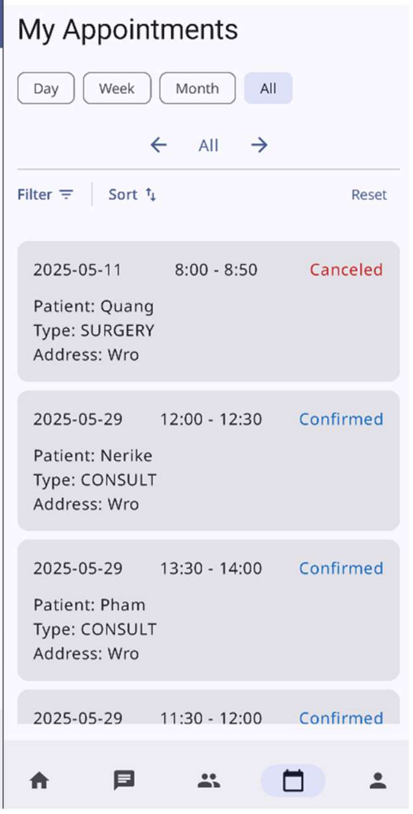
Weight: 68.0 kg

Date of Birth: 2003-02-15



Ferrous Forte
Dosage: 60 pills
Frequency: 1 a day
From: 12/06/2025 to 12/08/2025

Magnesium
Dosage: Pill
Frequency: 2 per day
From: 2025/06/25 to 2025/07/25

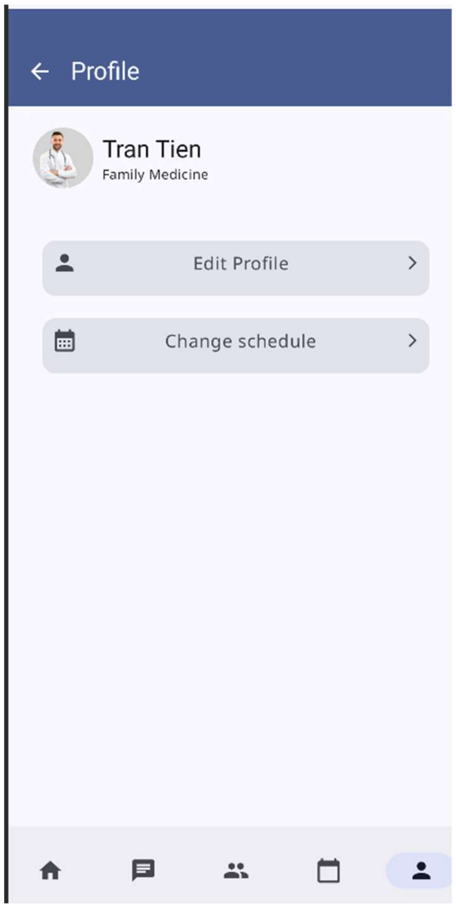


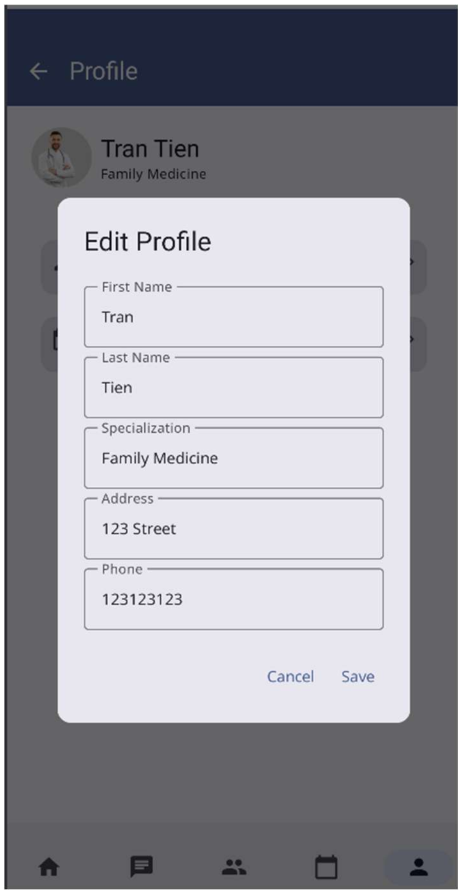
2025-05-11 8:00 - 8:50 Canceled
Patient: Quang
Type: SURGERY
Address: Wro

2025-05-29 12:00 - 12:30 Confirmed
Patient: Nerike
Type: CONSULT
Address: Wro

2025-05-29 13:30 - 14:00 Confirmed
Patient: Pham
Type: CONSULT
Address: Wro

2025-05-29 11:30 - 12:00 Confirmed

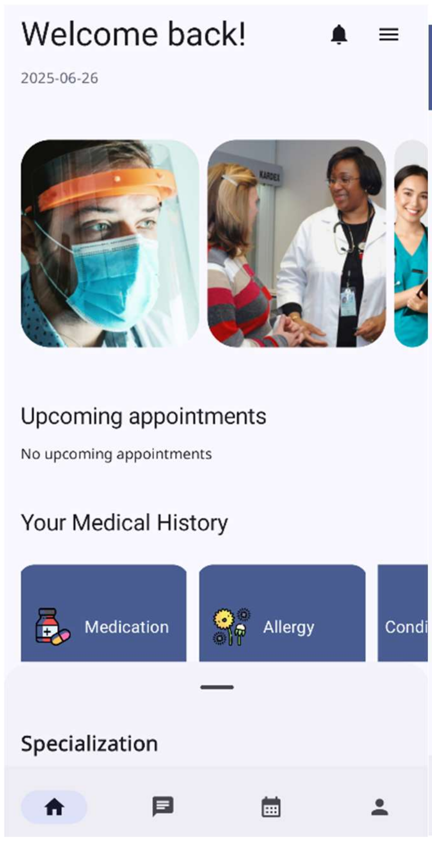


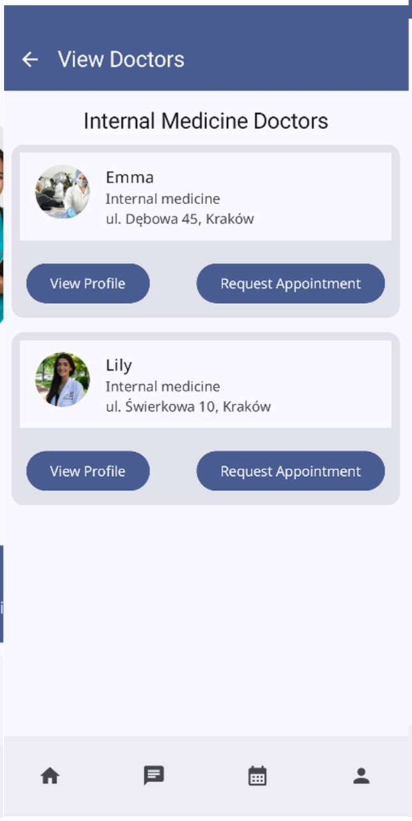


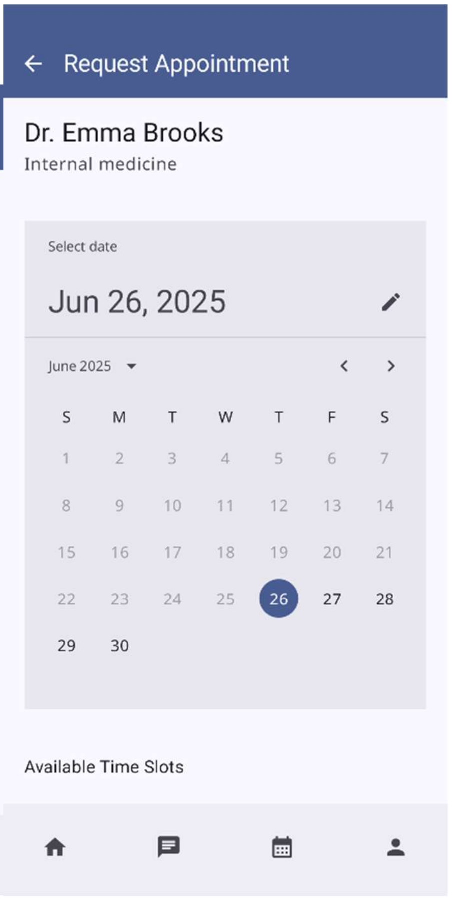
First Name: Tran
Last Name: Tien
Specialization: Family Medicine
Address: 123 Street
Phone: 123123123

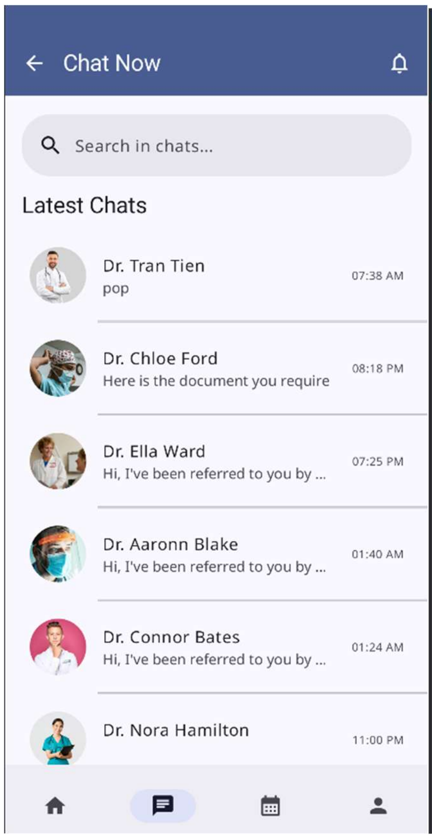
Patient Screens:

A general user who can register for an account, browse available doctors, book appointments, attend e-consultations via chat, view past consultations in their medical history, and upload/download medical documents such as prescriptions


Welcome back!
2025-06-26
Upcoming appointments
No upcoming appointments
Your Medical History
Medication
Allergy
Condi
Specialization


View Doctors
Internal Medicine Doctors
Emma
Internal medicine
ul. Dębowa 45, Kraków
View Profile
Request Appointment
Lily
Internal medicine
ul. Świerkowa 10, Kraków
View Profile
Request Appointment


Request Appointment
Dr. Emma Brooks
Internal medicine
Select date
Jun 26, 2025
June 2025
S M T W T F S
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Available Time Slots


Chat Now

Search in chats...

Latest Chats

Dr. Tran Tien
pop
07:38 AM

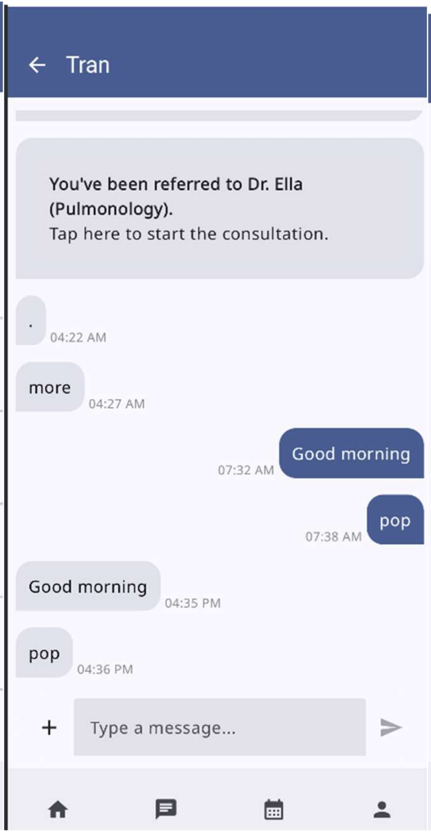
Dr. Chloe Ford
Here is the document you require
08:18 PM

Dr. Ella Ward
Hi, I've been referred to you by ...
07:25 PM

Dr. Aaronn Blake
Hi, I've been referred to you by ...
01:40 AM

Dr. Connor Bates
Hi, I've been referred to you by ...
01:24 AM

Dr. Nora Hamilton
11:00 PM


Tran

You've been referred to Dr. Ella (Pulmonology).
Tap here to start the consultation.

.
04:22 AM

more
04:27 AM

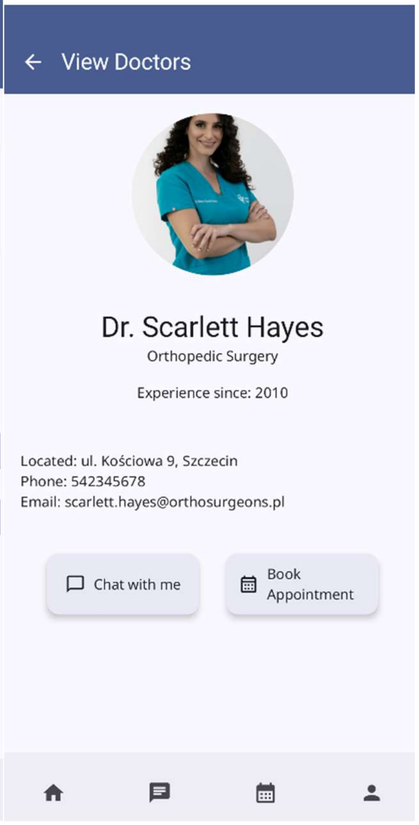
Good morning
07:32 AM

pop
07:38 AM

Good morning
04:35 PM

pop
04:36 PM

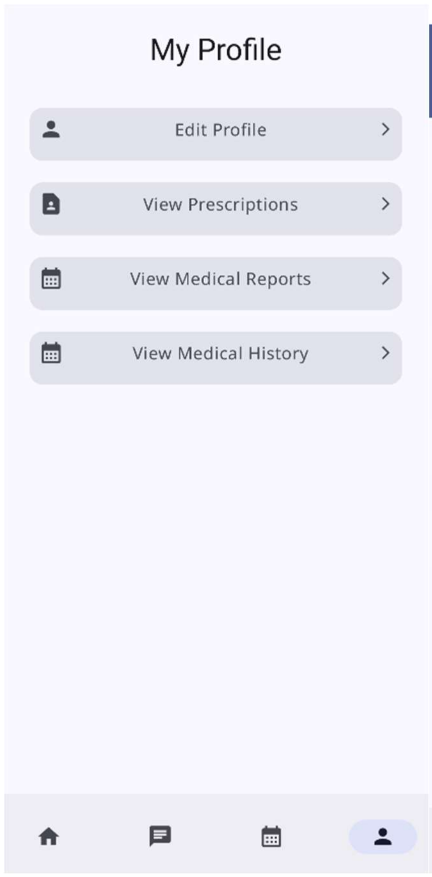
Type a message...


View Doctors

Dr. Scarlett Hayes
Orthopedic Surgery

Experience since: 2010

Located: ul. Kościowa 9, Szczecin
Phone: 542345678
Email: scarlett.hayes@orthosurgeons.pl



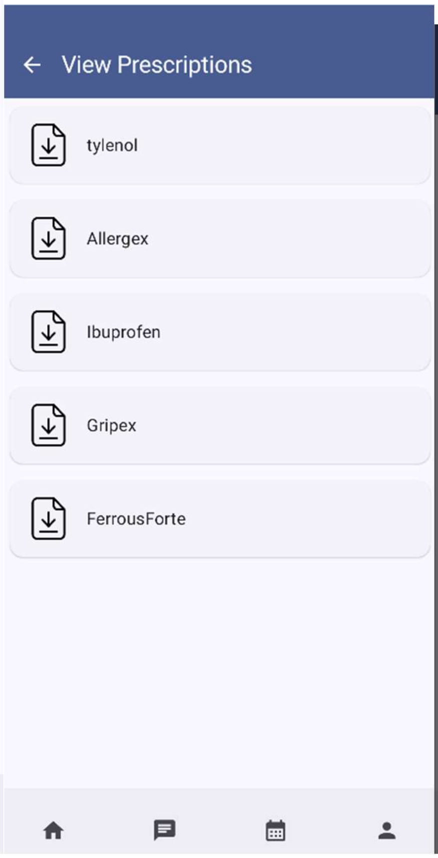
My Profile

Edit Profile

View Prescriptions

View Medical Reports

View Medical History



View Prescriptions

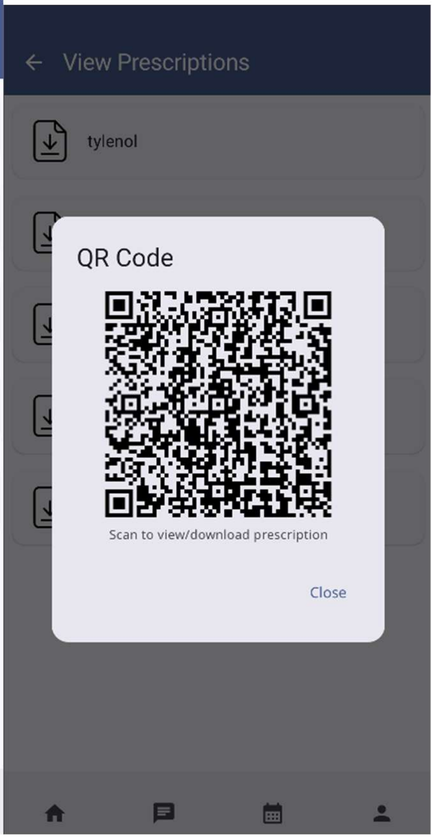
tylenol

Allergex

Ibuprofen

Gripex

FerrousForte



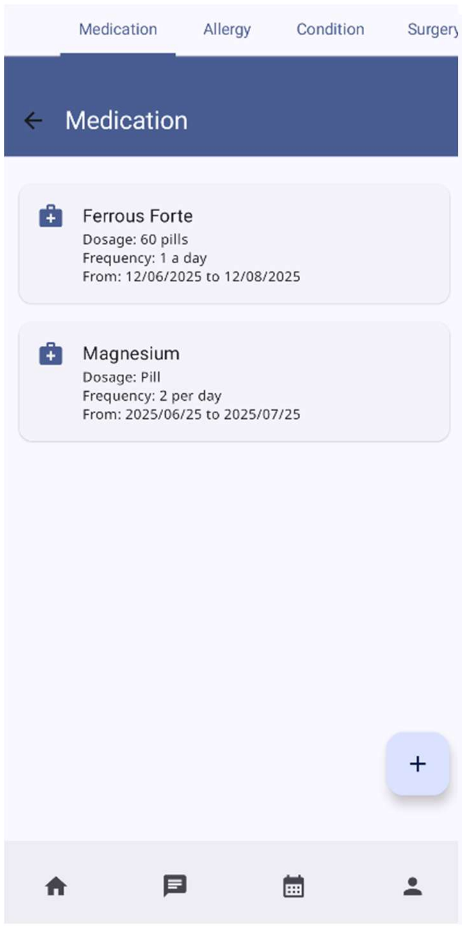
View Prescriptions

tylenol

QR Code

Scan to view/download prescription

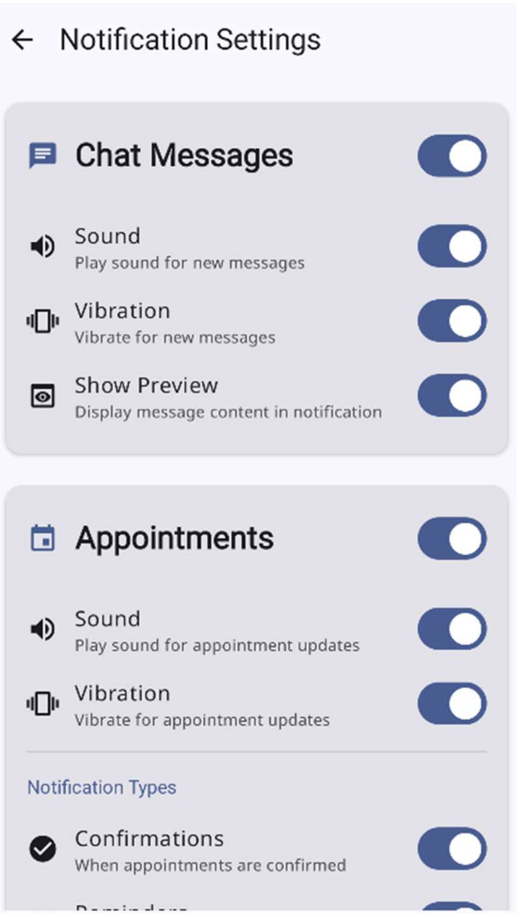
Close


Medication | Allergy | Condition | Surgery

Medication

Ferrous Forte
Dosage: 60 pills
Frequency: 1 a day
From: 12/06/2025 to 12/08/2025

Magnesium
Dosage: Pill
Frequency: 2 per day
From: 2025/06/25 to 2025/07/25


Notification Settings

Chat Messages

Sound
Play sound for new messages

Vibration
Vibrate for new messages

Show Preview
Display message content in notification

Appointments

Sound
Play sound for appointment updates

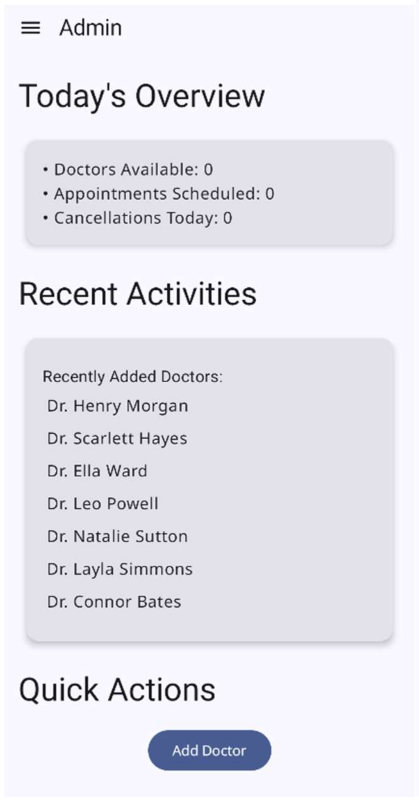
Vibration
Vibrate for appointment updates

Notification Types

Confirmations
When appointments are confirmed

Admin Screens:

Responsible for managing the entire system, including adding and removing doctors, modifying doctor profiles, updating their availability schedules, and managing patient/user data.


Admin

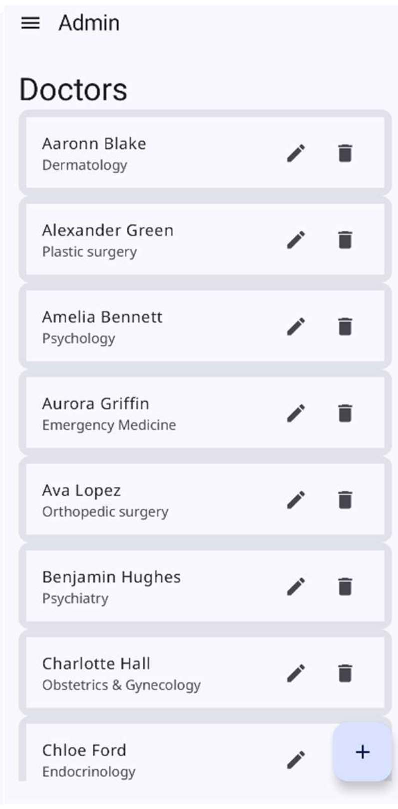
Today's Overview

- Doctors Available: 0
- Appointments Scheduled: 0
- Cancellations Today: 0

Recent Activities

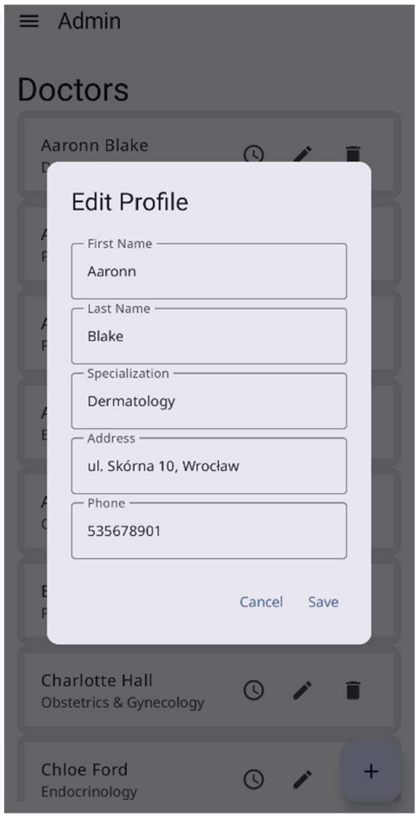
Recently Added Doctors:
Dr. Henry Morgan
Dr. Scarlett Hayes
Dr. Ella Ward
Dr. Leo Powell
Dr. Natalie Sutton
Dr. Layla Simmons
Dr. Connor Bates

Quick Actions


Admin

Doctors

Aaronn Blake	Dermatology
Alexander Green	Plastic surgery
Amelia Bennett	Psychology
Aurora Griffin	Emergency Medicine
Ava Lopez	Orthopedic surgery
Benjamin Hughes	Psychiatry
Charlotte Hall	Obstetrics & Gynecology
Chloe Ford	Endocrinology


Admin

Doctors

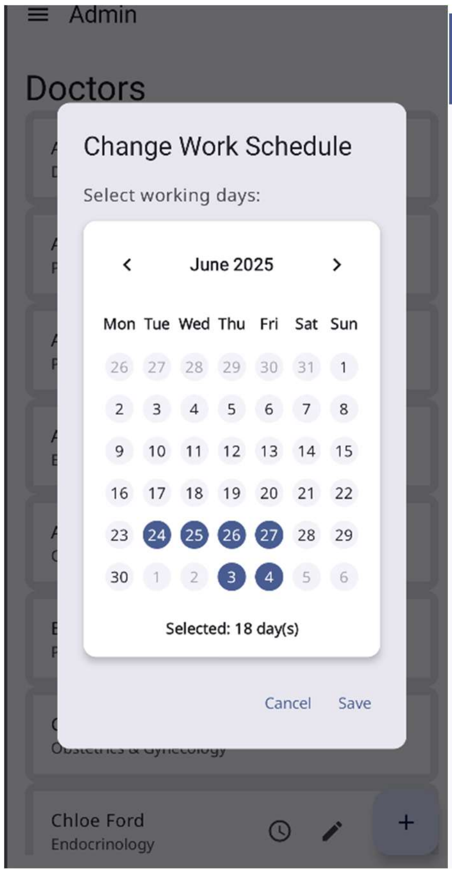
Aaronn Blake

Edit Profile

First Name	Aaronn
Last Name	Blake
Specialization	Dermatology
Address	ul. Skórna 10, Wrocław
Phone	535678901

Charlotte Hall
Obstetrics & Gynecology

Chloe Ford
Endocrinology



Admin

Doctors

Change Work Schedule

Select working days:

June 2025

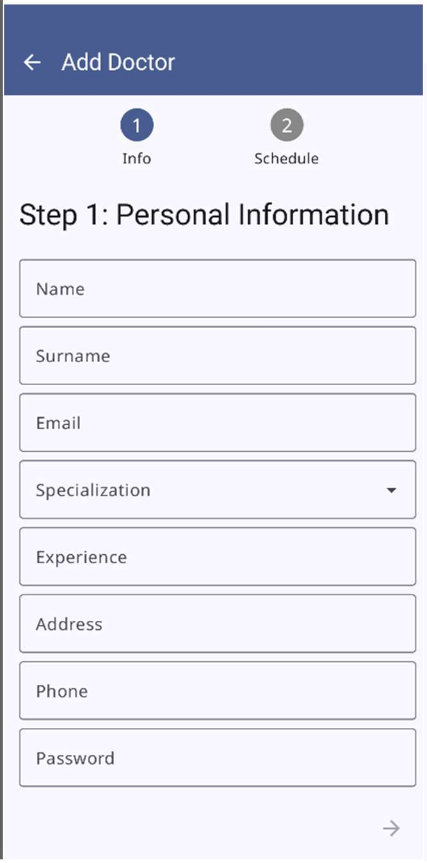
Mon	Tue	Wed	Thu	Fri	Sat	Sun
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

Selected: 18 day(s)

Cancel Save

Obstetrics & Gynecology

Chloe Ford
Endocrinology



Add Doctor

1 Info 2 Schedule

Step 1: Personal Information

Name

Surname

Email

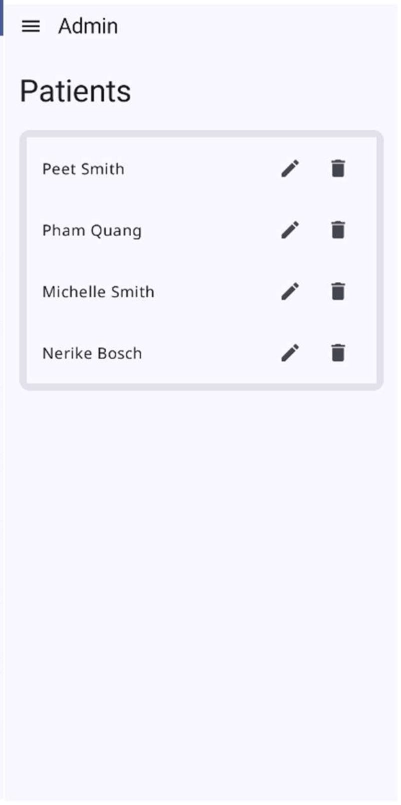
Specialization

Experience

Address

Phone

Password



Admin

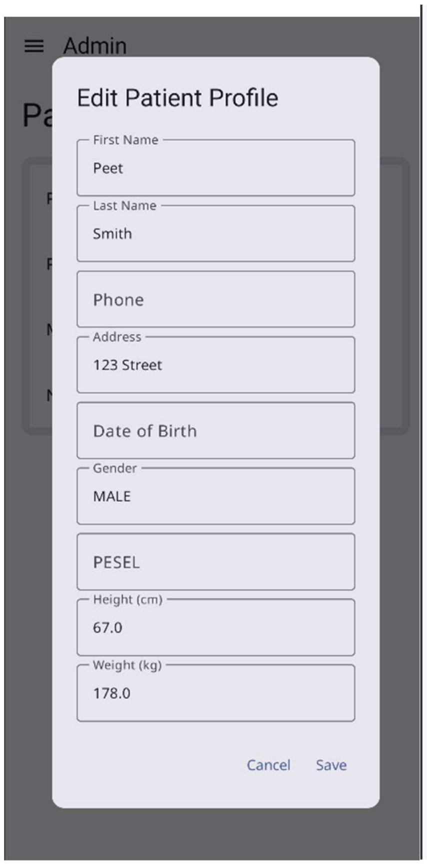
Patients

Peet Smith

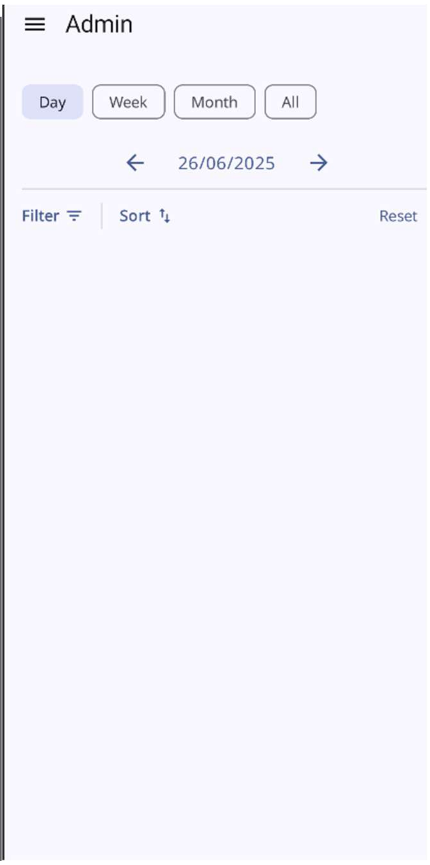
Pham Quang

Michelle Smith

Nerike Bosch



First Name: Peet
Last Name: Smith
Phone
Address: 123 Street
Date of Birth
Gender: MALE
PESEL
Height (cm): 67.0
Weight (kg): 178.0



Authors

- Nerike Bosch, Quang Pham