Psyche Assistant

Bilag 2: Kildekode indeks

Indholdsfortegnelse

1. Forord:

Indeksering af koden blev udført den 28. August, 2024.

Alt kode udviklet i forbindelse med projektet findes i dette dokument, inklusiv modeller, hjælpe funktioner, visninger, osv., for både platform-specifik og platform-agnostisk kode, samt for for front- og back-end-delene.
Derudover er indholdet fra opsætning- og konfigurations filer også inkluderet.

Bemærk, at sektions inddelingen og overskrifter, overordnet set afspejler filstrukturen, dog med undtagelse af, at den agnostiske kode kommer før den platform-specifikke kode – derfor er der også ”tomme” overskrifter (jf. afsnit 2, 2.1, 2.1.1, osv).

Endelig er det også værd at bemærke, at så vidt muligt at kode der er specifikt til det forudgående proof-of-concept projekt udeladt, så modeller til f.eks. Survey vil mangle, da disse ikke skal indgå i vurderingen af projektet. Såfremt det vurderes, at noget af den ældre kode fortsat er relevant for den nye kode, og derfor bør inkluderes for overbliks skyld, men ikke indgå i en evt. vurdering, markeres sektionen med **rød overskrift**

2. Front-end

2.1 commonMain

2.1.1. composeResources

2.1.1.1. values

2.1.1.1.1. strings.xml

```
<resources>
    <string name="title">Psyche Assistant</string>
    <string name="home">Home</string>
    <string name="survey">Survey</string>
    <string name="survey_history">Survey History</string>
    <string name="question_of_template">Question %1$d of %2$d</string>
    <string name="not_at_all">Not at all</string>
    <string name="very_much">Very much so</string>
    <string name="selected_score">Selected score: %1$d</string>
    <string name="previous">Previous</string>
    <string name="next">Next</string>
    <string name="cancel">Cancel</string>
    <string name="submit">Submit</string>
    <string name="sum_score">Total score: </string>
    <string name="language">Language: </string>
    <string name="completed">Survey completed!</string>
    <string name="final_score">You scored a total of %1$d on the Mental Fatigue Scale.</string>
    <string name="symptoms">General Symptoms: </string>
    <string name="recommendations">General Recommendations: </string>
    <string name="low_fatigue_symptoms">&#8226; Occasional, mild difficulties with concentration and memory.\
n&#8226; Slightly increased sensitivity to stress, but manageable.\n&#8226; Normal functioning in daily activities with
minimal impact.</string>
    <string name="low_fatigue_recommendations">&#8226; Be aware of early signs of fatigue but understand that these
symptoms are minor.\n&#8226; Ensure to maintain a healthy lifestyle, including regular sleep, exercise, and balanced
diet.\n&#8226; Regular relaxation and stress management techniques may help in preventing escalation of
symptoms.</string>
    <string name="moderate_fatigue_symptoms">&#8226; Noticeable difficulties in concentration and memory,
affecting some daily tasks.\n&#8226; Increased sensitivity to stress, leading to irritability and frustration.\n&#8226;
Feeling mentally drained after routine tasks that did not previously cause fatigue.</string>
    <string name="moderate_fatigue_recommendations">&#8226; Consider implementing structured rest periods during
the day to alleviate fatigue.\n&#8226; Prioritize tasks and activities to manage energy levels more effectively.\
n&#8226; Engage in activities that promote relaxation and reduce stress, such as mindfulness, yoga, or gentle exercise.\
n&#8226; It may be beneficial to seek advice from a healthcare professional to manage symptoms.</string>
    <string name="high_fatigue_symptoms">&#8226; Significant concentration and memory problems, making daily
tasks challenging.\n&#8226; High sensitivity to stress, leading to frequent irritability, frustration, and possibly mood
swings.\n&#8226; Feeling mentally exhausted most of the time, with a noticeable impact on work, social life, and
personal relationships.</string>
    <string name="high_fatigue_recommendations">&#8226; Strongly consider seeking medical advice to explore
potential underlying causes and appropriate treatments.\n&#8226; Implement regular, substantial rest periods and avoid
overexertion.\n&#8226; Cognitive-behavioral strategies might be useful to manage symptoms and improve daily
functioning.\n&#8226; It is important to communicate with employers, friends, and family about the fatigue to seek
their support and understanding.</string>
    <string name="very_high_fatigue_symptoms">&#8226; Severe concentration and memory issues, significantly
impairing the ability to perform everyday tasks.\n&#8226; Extreme sensitivity to stress, leading to frequent emotional
outbursts, anxiety, or depression.\n&#8226; Persistent mental exhaustion, often resulting in a reduced ability to engage
in social or professional activities.</string>
    <string name="very_high_fatigue_recommendations">&#8226; Immediate consultation with a healthcare
professional is critical to address these severe symptoms.\n&#8226; Possible need for medical intervention, including
medication and/or therapy.\n&#8226; Consider a comprehensive approach to treatment, which may involve lifestyle
changes, mental health support, and possibly occupational therapy.\n&#8226; Ensuring a strong support network of
family and friends can help in managing the condition more effectively.\n&#8226; Work on gradual reintroduction to
normal activities as the condition improves, under professional guidance.</string>
    <string name="general_recommendations">&#8226; Regular Check-ins: Monitor symptoms regularly to understand
if they are getting better or worse.\n&#8226; Healthy Habits: Maintain a routine that includes sufficient sleep, nutritious
meals, and physical activity.\n&#8226; Mindfulness and Relaxation: Engage in activities that promote mental
relaxation, such as meditation, breathing exercises, or hobbies.\n&#8226; Professional Support: Do not hesitate to seek
professional help if symptoms are affecting daily life. Early intervention can prevent further deterioration.</string>
    <string name="low">Low</string>
    <string name="moderate">Moderate</string>
    <string name="high">High</string>
    <string name="very_high">Very high</string>
    <string name="danish">Danish</string>
    <string name="english">English</string>
    <string name="previous_results">Previous Results</string>
    <string name="date">Date</string>
    <string name="score">Score</string>
    <string name="delete">Delete</string>
    <string name="no_connection">Error: Could not connect to server</string>
    <string name="okay">OK</string>

    <!-- Settings -->
    <string name="settings">Settings</string>
    <string name="auth_token">Auth Token: %1$s</string>
    <string name="no_token">No Token</string>
    <string name="group">Group: %1$s</string>
    <string name="no_group">No Group</string>
    <string name="logged_in_as">Logged in as %1$s</string>
    <string name="email">Email</string>
    <string name="password">Password</string>
    <string name="unknown_error">Unknown error</string>
    <string name="user_management">User Management</string>
    <string name="group_management">Group Management</string>
    <string name="sign_in_for_group">Sign in to create or join a group</string>
    <string name="failed_to_kick_member">Failed to kick member</string>
    <string name="kick_member">Kick member</string>
    <string name="create_group">Create group</string>
    <string name="join_group">Join group</string>
    <string name="leave_group">Leave group</string>
    <string name="disband_group">Disband group</string>
    <string name="group_code">Group code</string>
    <string name="no_code_or_auth">Group code cannot be empty or you are not authenticated</string>
    <string name="no_members">No members found..</string>
    <string name="no_email_or_pass">Email and password cannot be empty</string>
    <string name="login">Log in</string>
    <string name="register">Register</string>
    <string name="logout">Log out</string>
    <string name="delete_user">Delete user</string>
    <string name="member">Member:</string>
    <string name="member_label">Member: %1$s</string>
    <string name="members_amount">Member amount: %1$s</string>
    <string name="energy_expenditure">Energy Expenditure</string>
    <string name="day_and_week">day/week</string>
    <string name="energy_expenditure_label">Energy Expenditure: %1$d</string>
    <string name="you">You</string>
    <string name="confirm_delete">Confirm deletion</string>
    <string name="delete_confirm_intent">Are you sure you wish to delete your user? This action cannot be
undone.</string>
    <string name="confirm_disband">Confirm disbanding</string>
    <string name="disband_confirm_intent">Are you sure you wish to disband your group? This action cannot be
undone.</string>
    <string name="disband">Disband</string>

    <!-- Activities -->
    <string name="activities">Activities</string>
    <string name="activity">Activity</string>
    <string name="create_activity">Create activity</string>
    <string name="failed_delete_activity">Failed to delete activity</string>
    <string name="failed_complete_activity">Failed to complete activity</string>
    <string name="complete">Complete</string>
    <string name="select_date">Select date</string>
    <string name="selected_date">Date:</string>
    <string name="select_energy_cost">Energy Cost (0 to 10)</string>
    <string name="energy">Energy Cost:</string>
    <string name="energy_cost">Cost: %1$s</string>
    <string name="energy_consumption">Energy consumption</string>
    <string name="today">Today: %1$s</string>
    <string name="past_week">Weekly average: %1$s</string>
    <string name="previous_day">Previous</string>
    <string name="next_day">Next</string>
    <string name="description">Description</string>
    <string name="missing_group_warning">Please log in and create or join a group before creating an activity.</string>
    <string name="weekly_overview">Weekly overview</string>
    <string name="done">Done</string>
</resources>
```

2.1.1.2. values-da

2.1.1.2.1 strings.xml

```
<resources>
    <string name="title">Psyke Assistent</string>
    <string name="home">Hjem</string>
    <string name="survey">Undersøgelser</string>
    <string name="survey_history">Undersøgelser Historik</string>
    <string name="question_of_template">Spørgsmål %1$d af %2$d</string>
    <string name="not_at_all">Slet ikke</string>
    <string name="very_much">I høj grad</string>
    <string name="selected_score">Valgt score: %1$d</string>
    <string name="previous">Forrige</string>
    <string name="next">Næste</string>
    <string name="cancel">Annuller</string>
    <string name="submit">Indsend</string>
    <string name="sum_score">Samlet score: %1$d</string>
    <string name="language">Sprog: </string>
    <string name="completed">Spørgeskema fuldført!</string>
    <string name="final_score">Du scorede samlet set %1$d point på Hjernetrætheds skalaen (Mental Fatigue
Scale).</string>
    <string name="symptoms">Generelle Symptomer:</string>
    <string name="recommendations">Generelle Anbefalinger:</string>
    <string name="low_fatigue_symptoms">&#8226; Lejlighedsvise, milde koncentrations- og hukommelsesproblemer.\
n&#8226; Let øget følsomhed over for stress, men håndterbart.\n&#8226; Normal funktion i daglige aktiviteter med
minimal påvirkning.</string>
    <string name="low_fatigue_recommendations">&#8226; Vær opmærksom på tidlige tegn på træthed, men forstå at
disse symptomer er mindre.\n&#8226; Sørg for at opretholde en sund livsstil, herunder regelmæssig søvn, motion og en
afbalanceret kost.\n&#8226; Regelmæssig afslapning og stresshåndteringsteknikker kan hjælpe med at forhindre
forværring af symptomerne.</string>
    <string name="moderate_fatigue_symptoms">&#8226; Mærkbare koncentrations- og hukommelsesproblemer, der
påvirker nogle daglige opgaver.\n&#8226; Øget følsomhed over for stress, hvilket fører til irritabilitet og frustration.\
n&#8226; Føler sig mentalt udmattet efter rutineopgaver, der tidligere ikke forårsagede træthed.</string>
    <string name="moderate_fatigue_recommendations">&#8226; Overvej at implementere strukturerede hvileperioder i
løbet af dagen for at lindre træthed.\n&#8226; Prioriter opgaver og aktiviteter for at håndtere energiniveauer mere
effektivt.\n&#8226; Deltag i aktiviteter, der fremmer afslapning og reducerer stress, såsom mindfulness, yoga eller let
motion.\n&#8226; Det kan være gavnligt at søge råd hos en sundhedsprofessionel for at håndtere
symptomerne.</string>
    <string name="high_fatigue_symptoms">&#8226; Betydelige koncentrations- og hukommelsesproblemer, der gør
daglige opgaver udfordrende.\n&#8226; Høj følsomhed over for stress, hvilket fører til hyppig irritabilitet, frustration
og muligvis humørsvingninger.\n&#8226; Føler sig mentalt udmattet det meste af tiden, med en mærkbar påvirkning på
arbejde, socialt liv og personlige relationer.</string>
    <string name="high_fatigue_recommendations">&#8226; Overvej stærkt at søge lægehjælp for at undersøge
potentielle underliggende årsager og passende behandlinger.\n&#8226; Implementer regelmæssige, betydelige
hvileperioder og undgå overanstrengelse.\n&#8226; Kognitiv adfærdsterapi kan være nyttig til at håndtere
symptomerne og forbedre daglig funktion.\n&#8226; Det er vigtigt at kommunikere med arbejdsgivere, venner og
familie om træthed for at søge deres støtte og forståelse.</string>
    <string name="very_high_fatigue_symptoms">&#8226; Alvorlige koncentrations- og hukommelsesproblemer, der
markant hæmmer evnen til at udføre daglige opgaver.\n&#8226; Ekstrem følsomhed over for stress, hvilket fører til
hyppige følelsesmæssige udbrud, angst eller depression.\n&#8226; Vedvarende mental udmattelse, ofte resulterende i
en reduceret evne til at deltage i sociale eller professionelle aktiviteter.</string>
    <string name="very_high_fatigue_recommendations">&#8226; Øjeblikkelig konsultation med en
sundhedsprofessionel er kritisk for at tackle disse alvorlige symptomer.\n&#8226; Mulig behov for medicinsk
```

```
intervention, herunder medicin og/eller terapi.\n&#8226; Overvej en omfattende tilgang til behandling, som kan
involvere livsstilsændringer, mental sundhedsstøtte og muligvis ergoterapi.\n&#8226; At sikre et stærkt støttenetværk af
familie og venner kan hjælpe med at håndtere tilstanden mere effektivt.\n&#8226; Arbejd på gradvis genindførelse af
normale aktiviteter, efterhånden som tilstanden forbedres, under professionel vejledning.</string>
    <string name="general_recommendations">&#8226; Regelmæssige tjek: Overvåg symptomerne regelmæssigt for at
forstå, om de bliver bedre eller værre.\n&#8226; Sunde vaner: Oprethold en rutine, der inkluderer tilstrækkelig søvn,
nærende måltider og fysisk aktivitet.\n&#8226; Mindfulness og afslapning: Deltag i aktiviteter, der fremmer mental
afslapning, såsom meditation, vejrtrækningsøvelser eller hobbyer.\n&#8226; Professionel støtte: Tøv ikke med at søge
professionel hjælp, hvis symptomerne påvirker dagligdagen. Tidlig indgriben kan forhindre yderligere
forværring.</string>
    <string name="low">Lav</string>
    <string name="moderate">Moderat</string>
    <string name="high">Høj</string>
    <string name="very_high">Meget høj</string>
    <string name="danish">Dansk</string>
    <string name="english">Engelsk</string>
    <string name="previous_results">Tidligere resultater</string>
    <string name="date">Dato</string>
    <string name="score">Score</string>
    <string name="delete">Slet</string>
    <string name="no_connection">Fejl: Kunne ikke forbinde til server</string>
    <string name="okay">OK</string>

    <!-- Settings -->
    <string name="settings">Indstillinger</string>
    <string name="auth_token">Auth Token: %1$s</string>
    <string name="no_token">Ingen Token</string>
    <string name="group">Gruppe: %1$s</string>
    <string name="no_group">Ingen gruppe</string>
    <string name="logged_in_as">Logget ind som %1$s</string>
    <string name="email">Email</string>
    <string name="password">Kodeord</string>
    <string name="unknown_error">Ukendt fejl</string>
    <string name="user_management">Brugerhåndtering</string>
    <string name="group_management">Gruppehåndtering</string>
    <string name="sign_in_for_group">Log ind for at oprette eller deltage i en gruppe</string>
    <string name="failed_to_kick_member">Kunne ikke fjerne medlem</string>
    <string name="kick_member">Fjern medlem</string>
    <string name="create_group">Opret gruppe</string>
    <string name="join_group">Deltag i gruppe</string>
    <string name="leave_group">Forlad gruppe</string>
    <string name="disband_group">Opløs gruppe</string>
    <string name="group_code">Gruppe kode</string>
    <string name="no_code_or_auth">Gruppe koden må ikke være tom, eller du er ikke logget ind</string>
    <string name="no_members">Der blev ikke fundet nogen medlemmer..</string>
    <string name="no_email_or_pass">Email og kodeord må ikke være tomme</string>
    <string name="login">Log ind</string>
    <string name="register">Registrer</string>
    <string name="logout">Log ud</string>
    <string name="delete_user">Slet bruger</string>
    <string name="member">Medlem</string>
    <string name="member_label">Medlem: %1$s</string>
    <string name="members_amount">Antal medlemmer: %1$s</string>
    <string name="energy_expenditure">Energiforbrug</string>
    <string name="day_and_week">dag/uge</string>
    <string name="energy_expenditure_label">Energiforbrug: %1$d</string>
    <string name="you">Dig</string>
    <string name="confirm_delete">Bekræft sletning</string>
    <string name="delete_confirm_intent">Er du sikker på, du ønsker at slette din bruger? Dette kan ikke
omgøres.</string>
    <string name="confirm_disband">Bekræft opløsning</string>
    <string name="disband_confirm_intent">Er du sikker på, du ønsker at opløse din gruppe? Dette kan ikke
omgøres.</string>
    <string name="disband">Opløs</string>

    <!-- Activities -->
    <string name="activities">Aktiviteter</string>
    <string name="activity">Aktivitet</string>
    <string name="create_activity">Opret aktivitet</string>
    <string name="failed_create_activity">Kunne ikke slette aktivitet</string>
    <string name="failed_complete_activity">Kunne ikke fuldføre aktivitet</string>
    <string name="complete">Fuldfør</string>
    <string name="select_date">Vælg dato</string>
    <string name="selected_date">Dato: %1$s</string>
    <string name="select_energy_cost">Energiomkostning (0 til 10)</string>
    <string name="energy">Energiomkostning</string>
    <string name="energy_cost">Omkostning: %1$s</string>
    <string name="energy_consumption">Energiforbrug</string>
    <string name="today">I dag: %1$s</string>
    <string name="past_week">Ugentlig gennemsnit: %1$s</string>
    <string name="description">Beskrivelse</string>
    <string name="missing_group_warning">Venligst log ind og opret eller deltag i en gruppe, før du opretter
aktiviteter.</string>
    <string name="weekly_overview">Ugentlig oversigt</string>
    <string name="done">Fuldfør</string>
</resources>
```

2.1.2. kotlin

2.1.2.1. Composable

2.1.2.1.1. Common

2.1.2.1.1.1. ErrorScreen.kt

```
package org.psyche.assistant.Composable.Common

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier

/**
 * (DEPRECATED!) Error screen
 * Common composable to show full-screen error message (taken as a parameter).
 * As of writing not used.
 * @param message
 */
@Composable
fun ErrorScreen(message: String) {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Text(message)
    }
}
```

2.1.2.1.1.2. LoadingScreen.kt

```
package org.psyche.assistant.Composable.Common

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier

/**
 * Loading screen
 * Common composable used to display a loading screen if the app has not finished loading critical data.
 * Primarily used during initial start-up when App is gathering data from external database.
 */
@Composable
fun LoadingScreen() {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        CircularProgressIndicator()
    }
}
```

2.1.2.1.2. Components

2.1.2.1.2.1. DatePickerComponent.kt

```
package org.psyche.assistant.Composable.Components

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.Button
import androidx.compose.material.TabRowDefaults
import androidx.compose.material.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.dp
import kotlinx.datetime.LocalDate
import network.chaintech.kmp_date_time_picker.ui.datepicker.WheelDatePickerView
import network.chaintech.kmp_date_time_picker.utils.DateTimePickerView
import org.jetbrains.compose.resources.stringResource
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.select_date
import psycheassistant.composeapp.generated.resources.selected_date
```

```
/**
 * Date picker component
 * Used to display a date-picker alert-type dialogue.
 * This is instantiated using the library kmp_date_time_picker (WheelDatePickerView),
 * passing the functions to be used for selecting the chosen date/dismissing the component, as well as the initial date
(typically the current date).
 * @param initialDate
 * @param onDateSelected
 * @param onDismiss
 */
@Composable
fun DatePickerComponent(
    initialDate: LocalDate,
    onDateSelected: (LocalDate) -> Unit,
    onDismiss: () -> Unit
) {
    var showDatePicker by remember { mutableStateOf(false) }
    var selectedDate by remember { mutableStateOf(initialDate) }

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = stringResource(Res.string.selected_date, selectedDate.toString())
        )
        Button(
            onClick = { showDatePicker = true },

        ) {
            Text(text = stringResource(Res.string.select_date))
        }

        if (showDatePicker) {
            WheelDatePickerView(
                showDatePicker = showDatePicker,
                rowCount = 5,
                height = 200.dp,
                title = stringResource(Res.string.select_date),
                doneLabel = stringResource(Res.string.done),
                dateTimePickerView = DateTimePickerView.DIALOG_VIEW,
                dragHandle = {
                    TabRowDefaults.Divider(
                        modifier = Modifier.padding(top = 8.dp).width(15.dp).clip(CircleShape),
                        thickness = 4.dp,
                    )
                },
                onDoneClick = {
                    selectedDate = it
                    onDateSelected(it)
                    showDatePicker = false
                },
                onDismiss = {
                    onDismiss()
                    showDatePicker = false
                }
        )
      }
    }
}
```

2.1.2.1.2.2. EnergyExpenditureChart.kt

```
package org.psyche.assistant.Composable.Components

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.Canvas
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.material.Divider
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.geometry.Size
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.datetime.DatePeriod
import kotlinx.datetime.LocalDate
import kotlinx.datetime.minus
import network.chaintech.kmp_date_time_picker.utils.now
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Model.Activity.Activity
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.weekly_overview

/**
 * Energy expenditure chart
 * Component used to visualize a list of activities, sent in data as String/List key-value pairs.
 * with keys being unique dates, and the lists being activities occuring on that date.
 *
 * @param data
 */
@Composable
fun EnergyExpenditureChart(data: Map<String, List<Activity>>) {
    if (data.isEmpty()) {
        return
    }

    val textStyle = TextStyle(color = Color.Gray, fontSize = 10.sp)
    val padding = 15.dp

    // Generate a list of dates, starting from today, going X days back, essentially using a for-loop like structure to do so.
    val today = LocalDate.now()
    val dateRange = (0 until 7).map { daysBack ->
        today.minus(DatePeriod(days = daysBack))
    }.reversed().map { it.toString() }

    // Generate the daily expenditure map by associating each date generated earlier, summing the energyCost of the list
```

```
of activities in the passed data, for that date.
    val dailyExpenditure = dateRange.associateWith { date ->
        data[date]?.sumOf { it.energyCost.toDouble() } ?: 0.0
    }

    Column(
        modifier = Modifier
            .border(BorderStroke(1.dp, Color.Gray))
            .fillMaxWidth()
            .padding(bottom = 16.dp)

    ) {
        Text(
            text = stringResource(Res.string.weekly_overview),
            style = MaterialTheme.typography.h4,
            modifier = Modifier
                .padding(padding)
                .fillMaxWidth()
        )

        Divider()

        Box(
            modifier = Modifier
                .fillMaxWidth()
                .padding(padding)
                .height(150.dp)
        ) {
            Canvas(modifier = Modifier.fillMaxSize()) {
                val width = size.width / 7f // Fixed number of bars (7)
                val maxEnergy = dailyExpenditure.values.maxOrNull() ?: 1.0

                // For each date-energyExpenditure pair (toList), we draw a rectangle, using the relative energy per day
divided by the absolute max of all dates
                dailyExpenditure.toList().forEachIndexed { index, (_, energy) ->
                    val barHeight = (energy / maxEnergy) * size.height

                    drawRect(
                        color = Color.Green,
                        topLeft = Offset(x = index * width, y = size.height - barHeight.toFloat()),
                        size = Size(width - 4.dp.toPx(), barHeight.toFloat())
                    )
                }
            }
        }

        Divider()

        // Adding a row beneath the Canvas, were we explicitly write out the date and energy expenditure for each bar
generated above.
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = padding),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically,
        ) {
            dailyExpenditure.keys.forEach { date ->
                val energy = dailyExpenditure[date] ?: 0.0

                Text(
                    text = date.split("-").let {
                        "${it[2]}/${it[1]}\n${energy}"
                    },
                    style = textStyle,
                    modifier = Modifier
                        .weight(1f),
                    textAlign = TextAlign.Center

                )
            }
        }
    }
}
```

2.1.2.1.3. Dialogs

2.1.2.1.3.1. ActivityCreateDialog.kt

```
package org.psyche.assistant.Composable.Dialogs

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.launch
import kotlinx.datetime.LocalDate
import network.chaintech.kmp_date_time_picker.utils.now
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Components.DatePickerComponent
import psycheassistant.composeapp.generated.resources.*

/**
 * Activity create dialog
 * A dialog used for creating new activities, passing the functions needed to submit the new activity/dismiss the dialog
 * Using coroutine to asynchronously attempt to create the activity using onSubmit, as well as the variables
 * @param onDismiss
 * @param onSubmit
 */
@Composable
fun ActivityCreateDialog(
    onDismiss: () -> Unit,
    onSubmit: (LocalDate, Float, String) -> Unit
) {
    val coroutineScope = rememberCoroutineScope()
    var selectedDate by remember { mutableStateOf(LocalDate.now()) }
    var energyCost by remember { mutableStateOf(0f) }
    var description by remember { mutableStateOf("") }
var errorMessage by remember { mutableStateOf("") }
var isSubmitting by remember { mutableStateOf(false) }

fun handleSubmit() {
    isSubmitting = true
    if (description.trim().isEmpty()) {
        errorMessage = "Description cannot be empty"
        isSubmitting = false
        return
    }
    coroutineScope.launch {
        try {
            onSubmit(selectedDate, energyCost, description)
            onDismiss()
        } catch (e: Exception) {
            errorMessage = e.message ?: "An error occurred"
        } finally {
            isSubmitting = false
        }
    }
}

AlertDialog(
    onDismissRequest = { onDismiss() },
    title = { Text(stringResource(Res.string.create_activity)) },
    text = {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(5.dp)
        ) {
            // Datepicker component, includes button to activate the component.
            DatePickerComponent(
                initialDate = selectedDate,
                onDateSelected = { date -> selectedDate = date },
                onDismiss = { }
            )

            Divider(
                modifier = Modifier
                    .padding(vertical = 15.dp),
                thickness = 4.dp
            )

            // Energy Cost Slider
            Text(stringResource(Res.string.select_energy_cost))
            Slider(
                value = energyCost,
                onValueChange = { energyCost = it },
                valueRange = 0f..10f,
                steps = 9
            )
            Text(stringResource(Res.string.energy_cost, energyCost.toInt()))

            Divider(
                modifier = Modifier
                    .padding(vertical = 15.dp),
                thickness = 4.dp
            )

            Text(stringResource(Res.string.description))
            TextField(
                value = description,
                onValueChange = { description = it },
                modifier = Modifier.fillMaxWidth(),
                placeholder = { Text("...") }
            )

            if (errorMessage.isNotEmpty()) {
                Text(
                    text = errorMessage,
                    color = Color.Red,
                    style = MaterialTheme.typography.body2
                )
            }
        }
    },
    confirmButton = {
        Button(onClick = { handleSubmit() }, enabled = !isSubmitting) {
            Text(stringResource(Res.string.create_activity))
        }
    },
    dismissButton = {
        Button(onClick = { onDismiss() }) {
            Text(stringResource(Res.string.cancel))
        }
    }
  )
}
```

2.1.2.1.3.2. ConfirmDeleteDialog.kt

```
package org.psyche.assistant.Composable.Dialogs

import androidx.compose.material.AlertDialog
import androidx.compose.material.Text
import androidx.compose.material.TextButton
import androidx.compose.runtime.Composable

@Composable
fun ConfirmDeleteDialog(
    title: String,
    message: String,
    confirmButtonText: String,
    dismissButtonText: String,
    onConfirm: () -> Unit,
    onDismiss: () -> Unit
) {
    AlertDialog(
        onDismissRequest = { onDismiss() },
        title = { Text(text = title) },
        text = { Text(text = message) },
        confirmButton = {
            TextButton(
                onClick = {
```

```
                onConfirm()
                onDismiss()
            }
        ) {
            Text(text = confirmButtonText)
        }
    },
    dismissButton = {
        TextButton(onClick = { onDismiss() }) {
            Text(text = dismissButtonText)
        }
    }
  )
}
```

2.1.2.1.4. Items

2.1.2.1.4.1. ActivityItem.kt

```
package org.psyche.assistant.Composable.Items

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Check
import androidx.compose.material.icons.filled.Delete
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.drawBehind
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Model.Activity.Activity
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.complete
import psycheassistant.composeapp.generated.resources.delete

/**
 * Activity item
 * Composable used to display a single Activity record, passed to it as a parameter.
 * Note this exists as part of a table structure, and is therefore arranged in rows.
 * @param activity
 * @param onCompleteClick
 * @param onDeleteClick
 */
@Composable
fun ActivityItem(
    activity: Activity,
    onCompleteClick: (Activity) -> Unit,
    onDeleteClick: (Activity) -> Unit,
    ) {

    val isCompleted = activity.completed

    Row(
        modifier = Modifier
            .padding(vertical = 8.dp)
            // Append a second modifier in order to strike through the entire row, if the activity is completed.
            .then(
                if (activity.completed) Modifier.drawBehind {
                    val strokeWidth = 2.dp.toPx()
                    val yOffset = size.height / 2
                    drawLine(
                        color = Color.Gray,
                        start = androidx.compose.ui.geometry.Offset(0f, yOffset),
                        end = androidx.compose.ui.geometry.Offset(size.width, yOffset),
                        strokeWidth = strokeWidth
                    )
                } else Modifier
            ),
        horizontalArrangement = Arrangement.SpaceBetween,
        verticalAlignment = Alignment.CenterVertically,

        )
    {
        Text(
            text = activity.description,
            style = MaterialTheme.typography.body1,
            modifier = Modifier
                    .weight(2f)
        )
        Text(
            text = activity.energyCost.toString(),
            style = MaterialTheme.typography.body2,
            modifier = Modifier
                .weight(1f)
        )
        IconButton(
            onClick = {
                if (!isCompleted) {
                    onCompleteClick(activity)
                }
            },
            enabled = !isCompleted
        ) {
            Icon(Icons.Filled.Check, contentDescription = stringResource(Res.string.complete))
        }
        IconButton(onClick = { onDeleteClick(activity) }) {
            Icon(Icons.Filled.Delete, contentDescription = stringResource(Res.string.delete))
        }
    }
}
```

2.1.2.1.4.2. ActivityItemTable.kt

```
package org.psyche.assistant.Composable.Items

import androidx.compose.foundation.layout.*
import androidx.compose.material.Divider
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Model.Activity.Activity
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.activity
import psycheassistant.composeapp.generated.resources.energy

/**
 * Activity item table
 * Table stored as an item to be called with a list of activities. This in turn calls items for each individual activity.
 * This is done for the sake of readability.
 * @param activities
 * @param onCompleteClick
 * @param onDeleteClick
 * @param modifier
 */
@Composable
fun ActivityItemTable(
    activities: List<Activity>,
    onCompleteClick: (Activity) -> Unit,
    onDeleteClick: (Activity) -> Unit,
    modifier: Modifier = Modifier
) {
    Column(modifier = modifier) {
        Row(
            modifier = Modifier.fillMaxWidth().padding(vertical = 8.dp),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = stringResource(Res.string.activity),
                style = MaterialTheme.typography.subtitle1,
                modifier = Modifier.weight(1f)
            )
            Text(
                text = stringResource(Res.string.energy),
                style = MaterialTheme.typography.subtitle1,
                modifier = Modifier.weight(1f)
            )
            // Spaces for buttons
            Spacer(modifier = Modifier.width(40.dp))
            Spacer(modifier = Modifier.width(40.dp))
        }
        Divider()

        activities.forEach { activity ->
            ActivityItem(
                activity = activity,
            onCompleteClick = onCompleteClick,
            onDeleteClick = onDeleteClick,
        )
        Divider()
      }
    }
}
```

2.1.2.1.4.3. UserItem.kt

```
package org.psyche.assistant.Composable.Items

import androidx.compose.foundation.layout.*
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Sections.EnergyOverviewSection
import org.psyche.assistant.Helper.LevelColor
import org.psyche.assistant.Model.User.User
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.kick_member
import psycheassistant.composeapp.generated.resources.you

/**
 * User item
 * Composable used to display single User records, passed along as a parameter
 * Note it also accepts the function used to kick a user from the group (passed from GroupManagementPage > 
UserItemTable > UserItem),
 * as well as a boolean value to indicate whether the user logged in to the app, is the one being iterated upon 
(isCurrentUser),
 * to clarify to the user which user they are in the resulting table.
 * @param user
 * @param onKickClick Function
 * @param showKickButton
 * @param isCurrentUser
 * @param modifier
 */
@Composable
fun UserItem(
    user: User,
    onKickClick: (User) -> Unit,
    showKickButton: Boolean,
    isCurrentUser: Boolean,
    modifier: Modifier = Modifier
) {


    val textColor = LevelColor.getColor(user.energyExpenditure.toDouble())
    val userIdentifier = if (isCurrentUser) stringResource(Res.string.you) else user.email
    Row(
        modifier = modifier
            .padding(vertical = 8.dp),
        horizontalArrangement = Arrangement.SpaceBetween,
        verticalAlignment = Alignment.CenterVertically
    ) {
        Text(
            text = userIdentifier,
            style =  MaterialTheme.typography.body1.copy(color = textColor),
            modifier = Modifier
                .weight(2f)
        )

        EnergyOverviewSection(simple=true, specificUser = user)
        if (showKickButton) {
            IconButton(onClick = { onKickClick(user) }) {
                Icon(Icons.Filled.Delete, contentDescription = stringResource(Res.string.kick_member))
            }
        } else {
            Spacer(modifier = Modifier.width(40.dp))
        }
    }
}
```

2.1.2.1.4.4. UserItemTable.kt

```
package org.psyche.assistant.Composable.Items

import androidx.compose.foundation.layout.*
import androidx.compose.material.Divider
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Model.User.User
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.day_and_week
import psycheassistant.composeapp.generated.resources.energy_expenditure
import psycheassistant.composeapp.generated.resources.member

/**
 * User item table
 * Composable used to display a list of users in a table structure, calling UserItem on each individual user in order to
popualte the rows.
 * Note here there are additional parameters to control visibility of Kick-functionality on each user, avoiding showing
the option to others
 * than the group owner (and on the group owner themselves, if so).
 * @param users
 * @param onKickClick
 * @param showKickButton
 * @param currentUserId
 * @param ownerId
 * @param modifier
 */
@Composable
fun UserItemTable(
    users: List<User>,
    onKickClick: (User) -> Unit,
    showKickButton: Boolean,
    currentUserId: Int?,
    ownerId: Int?,
    modifier: Modifier = Modifier
) {
    Column(modifier = modifier) {
        Row(
            modifier = Modifier.fillMaxWidth().padding(vertical = 8.dp),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = stringResource(Res.string.member),
                style = MaterialTheme.typography.subtitle2,
                modifier = Modifier.weight(2f)
            )
            Column(
                modifier = Modifier.weight(1f)
            ) {
                Text(
                    text = stringResource(Res.string.energy_expenditure),
                    style = MaterialTheme.typography.subtitle2,

                )
                Text(
                    text = stringResource(Res.string.day_and_week),
                    style = MaterialTheme.typography.subtitle2,
                )
            }
            if (showKickButton) {
                Spacer(modifier = Modifier.width(40.dp)) // Space for the Kick button
            }
        }
        Divider()

        users.forEach { user ->
            UserItem(
                user = user,
                onKickClick = onKickClick,
                showKickButton = showKickButton && user.id != ownerId,
                isCurrentUser = user.id == currentUserId,
                modifier = Modifier.fillMaxWidth()
            )
            Divider()
        }
    }
}
```

2.1.2.1.5. Main

2.1.2.1.5.1. PsycheAssistantApp.kt

```kotlin
package org.psyche.assistant.Composable.Main

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.List
import androidx.compose.material.icons.twotone.CheckCircle
import androidx.compose.material.icons.twotone.Settings
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.painter.Painter
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.sp
import org.jetbrains.compose.resources.painterResource
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Common.LoadingScreen
import org.psyche.assistant.Composable.LocalAuthToken
import org.psyche.assistant.Composable.LocalGroup
import org.psyche.assistant.Composable.LocalUser
import org.psyche.assistant.Composable.Pages.AccountManagementPage
import org.psyche.assistant.Composable.Pages.ActivityPage
import org.psyche.assistant.Composable.Pages.MainPage
import org.psyche.assistant.Composable.Pages.SurveyPage
import org.psyche.assistant.Controller.GroupController
import org.psyche.assistant.Controller.UserController
import org.psyche.assistant.Helper.CustomTheme
import org.psyche.assistant.Model.Group.Group
import org.psyche.assistant.Model.User.User
import org.psyche.assistant.Storage.AuthStorage
import psycheassistant.composeapp.generated.resources.*

/**
 * Psyche assistant app
 * The main entry point for the application.
 * Mainly responsible for establishing the apps theme, navigation bar/function, as well as gathering initial data.
 */
@Composable
fun PsycheAssistantApp() {
    var currentScreen by remember { mutableStateOf("main") }
    var authToken = remember { mutableStateOf(AuthStorage.getAuthToken()) }
    val userState = remember { mutableStateOf<User?>(null) }
    val groupState = remember { mutableStateOf<Group?>(null) }


    var isLoading by remember { mutableStateOf(true) }
    var isError by remember { mutableStateOf(false) }
    var errorMessage by remember { mutableStateOf("") }

    // Asynchronous operation that monitors the given value for changes, executing the function when change occurs.
    LaunchedEffect(authToken.value) {
        if (authToken.value != null) {
            try {
                val token = authToken.value!!
                val userDetails = UserController().getUserProfile(token)
                userState.value = userDetails
                val groupDetails = userDetails?.groupId?.let { GroupController().getGroupDetails(it) }
                groupState.value = groupDetails
                isLoading = false
            } catch (e: Exception) {
                isLoading = false
                isError = true
                errorMessage = e.message.toString()
            }
        } else {
            isLoading = false
        }
    }

    MaterialTheme(
        colors = CustomTheme.psycheColors()
    ) {
        // pass critical values to the rest of the composable tree. I.e. share data with other pages/composable functions.
        CompositionLocalProvider(
            LocalAuthToken provides authToken,
            LocalUser provides userState,
            LocalGroup provides groupState
        ) {
            Scaffold(
                // Add a top bar which will act as a "Home" button, due to space constraint in the bottom bar.
                topBar = {
                    Box(
                        modifier = Modifier
                            .clickable { currentScreen = "main" } // Make the entire TopAppBar clickable
                            .fillMaxWidth()
                    ) {
                        TopAppBar(
                            title = {
                                Text(
                                    text = stringResource(Res.string.home),
                                    maxLines = 1,
                                    overflow = TextOverflow.Ellipsis,
                                )
                            },
                            navigationIcon = {
                                val icon: Painter = painterResource(Res.drawable.psyche_assistant_icon)
                                Icon(
                                    painter = icon,
                                    contentDescription = stringResource(Res.string.home),
                                    tint = Color.Unspecified
                                )
                            },
                        )
                    }
                },
                // Add bottom bar for general navigation.
                bottomBar = {
                    BottomNavigation {
```

```
            BottomNavigationItem(
                icon = { Icon(Icons.Filled.List, contentDescription = null) },
                label = {
                    Text(
                        stringResource(Res.string.survey),
                        style = TextStyle(fontSize = 11.sp),
                        maxLines = 1,
                        overflow = TextOverflow.Ellipsis
                    )
                },
                selected = currentScreen == "survey",
                onClick = { currentScreen = "survey" }
            )

            BottomNavigationItem(
                icon = { Icon(Icons.TwoTone.CheckCircle, contentDescription = null) },
                label = {
                    Text(
                        stringResource(Res.string.activities),
                        style = TextStyle(fontSize = 11.sp),
                        maxLines = 1,
                        overflow = TextOverflow.Ellipsis
                    )
                },
                selected = currentScreen == "activities",
                onClick = { currentScreen = "activities" }
            )
            BottomNavigationItem(
                icon = { Icon(Icons.TwoTone.Settings, contentDescription = null) },
                label = {
                    Text(
                        stringResource(Res.string.settings),
                        style = TextStyle(fontSize = 11.sp),
                        maxLines = 1,
                        overflow = TextOverflow.Ellipsis
                    )
                },
                selected = currentScreen == "settings",
                onClick = { currentScreen = "settings" }
            )

        }
    },
    // Main component for holding the actual content of the application, i.e. the various pages "inside" the layout.
    content = { innerPadding ->
        if (isLoading) {
            LoadingScreen()
        } else {
            Column(
                modifier = Modifier
                    .padding(innerPadding)
                    .fillMaxSize()
            ) {
                when (currentScreen) {
                    "main" -> MainPage()
                    "survey" -> SurveyPage(onBack = { currentScreen = "main" })
                    "settings" -> AccountManagementPage()
                    "activities" -> ActivityPage()
                }
            }
        }
    }
)

// Error dialog, used to explicitly but non-intrusively inform the user about an issue during app initialization.
if (isError) {
    AlertDialog(
        onDismissRequest = { isError = false },
        title = { Text(stringResource(Res.string.no_connection)) },
        text = { Text(errorMessage) },
        confirmButton = {
            Button(
                onClick = { isError = false }
            ) {
                Text("OK")
            }
        }
    )
}
        }
    }
}
```

2.1.2.1.6. Pages

2.1.2.1.6.1. AccountManagementPage.kt

```
package org.psyche.assistant.Composable.Pages

import androidx.compose.foundation.layout.*
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.LocalAuthToken
import org.psyche.assistant.Composable.Sections.GroupManagementSection
import org.psyche.assistant.Composable.Sections.UserManagementSection
import psycheassistant.composeapp.generated.resources.Res
import psycheassistant.composeapp.generated.resources.group_management
import psycheassistant.composeapp.generated.resources.sign_in_for_group
import psycheassistant.composeapp.generated.resources.user_management

/**
 * Account management page
 * Used to display various account-related functions, such as user registration/log in, group registration/joining,
 * group administration, as well as group member specific data.
 * Note these are handled by subsequent sections (UserManagementSection, GroupMangementSection).
 */
@Composable
fun AccountManagementPage() {
    val authToken = LocalAuthToken.current

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Text(stringResource(Res.string.user_management), style = MaterialTheme.typography.h5)
        Spacer(modifier = Modifier.height(8.dp))
        UserManagementSection()

        Spacer(modifier = Modifier.height(32.dp))
        if (authToken.value != null) {

            Text(stringResource(Res.string.group_management), style = MaterialTheme.typography.h5)
            Spacer(modifier = Modifier.height(8.dp))
            GroupManagementSection()
        } else {
            Text(stringResource(Res.string.sign_in_for_group))
        }
    }
}
```

2.1.2.1.6.2. ActivityPage.kt

```
package org.psyche.assistant.Composable.Pages

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material.AlertDialog
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.launch
import kotlinx.datetime.DatePeriod
import kotlinx.datetime.LocalDate
import kotlinx.datetime.minus
import kotlinx.datetime.plus
import network.chaintech.kmp_date_time_picker.utils.now
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Dialogs.ActivityCreateDialog
import org.psyche.assistant.Composable.Items.ActivityItemTable
import org.psyche.assistant.Composable.LocalAuthToken
import org.psyche.assistant.Composable.LocalGroup
import org.psyche.assistant.Controller.ActivityController
import org.psyche.assistant.Model.Activity.Activity
import psycheassistant.composeapp.generated.resources.*

/**
 * Activity page
 * Used to administrate activities, ensuring the list of activities are updated when needed,
 * that they can be created/completed/deleted, and that the user can paginate through dates to see activities from other
days.
 * Note that if they choose to complete activities from a different date, the activation counts for the day they were
completed,
 * not the deadline they had.
 */
@Composable
fun ActivityPage() {
    val authToken = LocalAuthToken.current
    val group = LocalGroup.current
    val activityController = ActivityController()
    val coroutineScope = rememberCoroutineScope()

    var currentDate by remember { mutableStateOf(LocalDate.now())}
    var activities by remember { mutableStateOf<List<Activity>>(emptyList())}

    var isWarningDialogOpen by remember { mutableStateOf(false) }
    var isDialogOpen by remember { mutableStateOf(false) }
    var isLoading by remember { mutableStateOf(false) }

    var errorMessage by remember { mutableStateOf("") }
    var unknownError = stringResource(Res.string.unknown_error)
    var failedDeleteError = stringResource(Res.string.failed_delete_activity)

    LaunchedEffect(authToken.value, activities, currentDate) {
        if (authToken.value != null) {
            isLoading = true
            try {
                activities = activityController.getActivityForToday(group.value!!.id, currentDate)
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    /**
     * Complete activity
     * Launches asynchronous operation to mark the activity as completed by the current user (i.e. matching the
authToken)
     * @param activity
     */
    fun completeActivity(activity: Activity) {
        coroutineScope.launch {
            try {
                isLoading = true
                val result = activityController.completeActivity(authToken.value!!, activity.id)
                if (result != null) {
                    activities = activities.filter { it.id != activity.id }
                } else {
                    errorMessage = "Herp"
                }
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    /**
     * Delete activity
     * Launches asynchronous operation to delete the activity.
     * @param activity
     */
    fun deleteActivity(activity: Activity) {
        coroutineScope.launch {
            try {
                isLoading = true
                val activityId = activity.id
                val result = activityController.deleteActivity(authToken.value!!, activity.id)
                if (result) {
                    activities = activities.filter { it.id != activityId }
                } else {
                    errorMessage = failedDeleteError
                }
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Row(
            horizontalArrangement = Arrangement.SpaceBetween,
            modifier = Modifier.fillMaxWidth()
        ) {

            Button(onClick = { currentDate = currentDate.minus(DatePeriod(days = 1)) }) {
                Text("<")
            }

            Text(currentDate.toString(), modifier = Modifier.align(Alignment.CenterVertically))
            Button(onClick = { currentDate = currentDate.plus(DatePeriod(days = 1)) }) {
                Text(">")
            }
        }

        Button(onClick = {
            if (group.value == null) {
                isWarningDialogOpen = true
            } else {
                isDialogOpen = true
            }
        }) {
            Text(stringResource(Res.string.create_activity))
        }

        // Add a scrollable column for the activities
        LazyColumn(
            modifier = Modifier
                .fillMaxSize()
                .padding(8.dp)
        ) {
            if (activities.isNotEmpty()) {
                item {
                    ActivityItemTable(
                        activities = activities,
                        onCompleteClick = { activity -> completeActivity(activity) },
                        onDeleteClick = { activity -> deleteActivity(activity) }
                    )
                }
            }
        }
    }

    // Call the Create Activity dialog when the boolean changes, and change it back once the create dialog is dismissed.
    if (isDialogOpen) {
        ActivityCreateDialog(
            onDismiss = { isDialogOpen = false },
            onSubmit = { date, cost, desc ->
                coroutineScope.launch {
                    try {
                        activityController.createActivity(authToken.value!!, date.toString(), desc, cost.toInt())
                        activities = activityController.getActivityForToday(group.value!!.id, LocalDate.now())
                    } catch (e: Exception) {
                        errorMessage = e.message ?: unknownError
                    }
                }
            }
        )
    }
    if (isWarningDialogOpen) {
        AlertDialog(
            onDismissRequest = { isWarningDialogOpen = false },
            title = { Text(stringResource(Res.string.no_group)) },
            text = { Text(stringResource(Res.string.missing_group_warning)) },
            confirmButton = {
                Button(onClick = { isWarningDialogOpen = false }) {
                    Text(stringResource(Res.string.okay))
                }
            }
        )
    }
}
```

2.1.2.1.6.3. MainPage.kt

```
package org.psyche.assistant.Composable.Pages

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import org.psyche.assistant.Composable.Sections.EnergyOverviewSection
import org.psyche.assistant.Composable.Sections.SurveyHistorySection
import org.psyche.assistant.Model.SurveyRepository

/**
 * the Main page ("home") now displays a visualization of the users energy expenditure for the day/average past week
 * as well as a bar graph for daily energy expendtiure for each day for the past week.
 *
 * DEPRECATED:
 * Main composable. Page meant to hold generalized information on how to use and interact with the app.
 * Using main-page for this as the target usergroup won't necessarily be all too tech-savvy, in addition to
 * potentially exceedingly mentally fatigued.
 */
@Composable
fun MainPage() {
    // Retrieve surveys from SQL Delight local storage.
    val db = SurveyRepository()
    var surveys by remember { mutableStateOf(db.selectAllSurveys()) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
            .verticalScroll(rememberScrollState()),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        EnergyOverviewSection()

        Spacer(modifier = Modifier.height(32.dp))
        SurveyHistorySection(
            surveys = surveys,
            onDeleteSurvey = { surveyId ->
                db.deleteSurveyById(surveyId)
                surveys = db.selectAllSurveys()
            }
        )

    }

}
```

2.1.2.1.7. Sections

2.1.2.1.7.1. EnergyOverviewSection.kt

```
package org.psyche.assistant.Composable.Sections

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.material.Divider
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import kotlinx.datetime.DatePeriod
import kotlinx.datetime.LocalDate
import kotlinx.datetime.minus
import network.chaintech.kmp_date_time_picker.utils.now
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Components.EnergyExpenditureChart
import org.psyche.assistant.Composable.LocalAuthToken
import org.psyche.assistant.Composable.LocalGroup
import org.psyche.assistant.Composable.LocalUser
import org.psyche.assistant.Controller.ActivityController
import org.psyche.assistant.Model.Activity.Activity
import org.psyche.assistant.Model.User.User
import psycheassistant.composeapp.generated.resources.*
import roundToDecimals

/**
 * Energy overview section
 * Used to display user-specific energy expenditure values, if specified.
 * It can be used to display a simplified summation of daily/weekly avg. expenditure, or a more detailed
 * version including graph visualization, determined by whether the function is explicitly called with "simple=true".
 * If it's called with a specificUser designated, this overwrites the logged-in user, used when generating list-views of
group members.
 * @param simple
 * @param specificUser
 */
@Composable
fun EnergyOverviewSection(simple: Boolean = false, specificUser: User? = null) {
    // Values to fetch relevant completed activities, and show today's energy expenditure.
    val authToken = LocalAuthToken.current
    val group = LocalGroup.current
    val activityController = ActivityController()
    val unknownError = stringResource(Res.string.unknown_error)

    // User is converted to variable so it can be reassigned, based on whether a specific user is passed to the function.
    var user = LocalUser.current
    var weeklyActivities by remember { mutableStateOf<Map<String, List<Activity>>>(emptyMap()) }
    var energyExpenditure by remember { mutableStateOf(0.0) }
    var energyExpenditureToday by remember { mutableStateOf(0.0) }
    var errorMessage by remember { mutableStateOf("") }
    var isLoading by remember { mutableStateOf(false) }

    if (specificUser != null)
    {
        user = remember {mutableStateOf(specificUser)}
    }
    LaunchedEffect(authToken) {
        if (authToken.value != null) {
            try {
                var activities = activityController.getHandledActivityByPeriod(
                    group.value!!.id,
                    LocalDate.now(),
                    LocalDate.now().minus(DatePeriod(days = 7)
                ))

                activities.forEach { activity ->
                    energyExpenditureToday += if (activity.handledOn == LocalDate.now() && activity.completed &&
activity.handledById == user.value?.id) activity.energyCost else 0
                    energyExpenditure += if (activity.completed && activity.handledById == user.value?.id)
activity.energyCost else 0
                }
                weeklyActivities = activities.filter { it.completed && it.handledById == user.value?.id}.groupBy
{it.handledOn.toString()}

            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    if (simple) {
        Text(
            text = "${energyExpenditureToday} / ${(energyExpenditure / 7).roundToDecimals(1)}"
        )
    }
    else {
        Box(
            modifier = Modifier
                .border(BorderStroke(1.dp, Color.Gray))
                .fillMaxWidth()
                .padding(bottom = 16.dp)
        ) {
            Column(
                modifier = Modifier
                    .padding(16.dp)
            ) {
                Text(
                    stringResource(Res.string.energy_consumption),
                    style = MaterialTheme.typography.h4
                )
                Divider(
                    modifier = Modifier
                        .height(4.dp)
                )
                Text(
                    stringResource(Res.string.today, energyExpenditureToday)
                )
                Text(
                    stringResource(Res.string.past_week, (energyExpenditure / 7).roundToDecimals(1))
                )
                EnergyExpenditureChart(data = weeklyActivities)
            }

        }
    }
}
```

2.1.2.1.7.2. GroupManagementSection.kt

```
package org.psyche.assistant.Composable.Sections

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.launch
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Dialogs.ConfirmDeleteDialog
import org.psyche.assistant.Composable.Items.UserItemTable
import org.psyche.assistant.Composable.LocalAuthToken
import org.psyche.assistant.Composable.LocalGroup
import org.psyche.assistant.Composable.LocalUser
import org.psyche.assistant.Controller.GroupController
import org.psyche.assistant.Controller.UserController
import org.psyche.assistant.Model.User.User
import psycheassistant.composeapp.generated.resources.*

/**
 * Group management section
 * Used for group-specific administrative tasks and overview. I.e. creating or joining a group,
 * as well as seeing an overview of the groups members, and being able to kick them (if the user is the owner of the
group).
 *
 */
@Composable
fun GroupManagementSection() {
    val coroutineScope = rememberCoroutineScope()
    val groupController = GroupController()
    val userController = UserController()

    var authToken = LocalAuthToken.current
    var group = LocalGroup.current
    var user = LocalUser.current
    var code by remember { mutableStateOf("")}
    var members by remember { mutableStateOf<List<User>>(emptyList())}

    var showConfirmationDialog by remember { mutableStateOf(false) }
    var isOwner by remember { mutableStateOf(false)}
    var isLoading by remember { mutableStateOf(false) }
    var errorMessage by remember { mutableStateOf("") }

    var unknownError = stringResource(Res.string.unknown_error)
    var failedToKick = stringResource(Res.string.failed_to_kick_member)
    var noCodeOrAuth = stringResource(Res.string.no_code_or_auth)
    var membersAmount = stringResource(Res.string.members_amount, members.size)


    LaunchedEffect(authToken, group.value, members) {
        if (authToken.value != null) {
            isLoading = true
            try {
                val groupDetails = groupController.getGroupDetails(group.value!!.id)
                group.value = groupDetails
                var groupOwner = userController.getUserById(groupDetails?.ownerId!!)
                isOwner = groupOwner?.id == user.value?.id
                members = groupDetails.userIds
                    .mapNotNull { userId -> userController.getUserById(userId) }
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    fun kickMember(user: User) {
        coroutineScope.launch {
            try {
                isLoading = true
                val result = groupController.kickMember(authToken.value!!, group.value!!.id, user.id)
                if (result != null) {
                    members = members.filter { it.id != user.id }
                } else {
                    errorMessage = failedToKick
                }
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    val disbandGroup = {
        coroutineScope.launch {
            try {
                isLoading = true
                val result = groupController.disbandGroup(authToken.value!!, group.value!!.id)
                if (result) {
                    group.value = null
                    members = emptyList()
                } else {
                    errorMessage = "Failed to disband group."
                }
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    Column(
        modifier = Modifier
        .fillMaxWidth()
    .padding(8.dp)
  ) {
    Text(stringResource(Res.string.group, group.value?.code ?: stringResource(Res.string.no_group)), style =
MaterialTheme.typography.h6)
    Spacer(modifier = Modifier.height(16.dp))

    if (authToken.value != null && group.value != null) {

      Row {
        Button(onClick = {
          coroutineScope.launch {
            try {
              isLoading = true
              val result = groupController.leaveGroup(authToken.value!!, group.value!!.id)
              if (result) {
                group.value = null
                members = emptyList()
              } else {
                errorMessage = "Testing.."
              }
            } catch (e: Exception) {
              errorMessage = e.message ?: unknownError
            } finally {
              isLoading = false
            }
          }
        }) {
          Text(text = stringResource(Res.string.leave_group))
        }
        Spacer(modifier = Modifier.padding(15.dp))
        if (isOwner) {
          Button(
            onClick = {
              showConfirmationDialog = true
            }, colors = ButtonDefaults.buttonColors(
              backgroundColor = Color.Red,
              contentColor = Color.Black
            )
          ) {
            Text(text = stringResource(Res.string.disband_group))
          }
        }
      }
    }

    if (authToken.value != null && group.value == null) {
      Button(onClick = {
        coroutineScope.launch {
          try {
            isLoading = true
            val groupDetails = groupController.createGroup(authToken.value!!)
            group.value = groupDetails
          } catch (e: Exception) {
            errorMessage = e.message ?: unknownError
          } finally {
            isLoading = false
          }
```

```
                }
            }) {
                Text(text = stringResource(Res.string.create_group))
            }

            Text(stringResource(Res.string.join_group), style = MaterialTheme.typography.h6)
            Spacer(modifier = Modifier.height(8.dp))

            OutlinedTextField(
                value = code,
                onValueChange = { code = it },
                label = { Text(stringResource(Res.string.group_code)) },
                modifier = Modifier.fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(8.dp))

            Button(onClick = {
                if (code.isNotBlank() && authToken.value != null) {
                    isLoading = true
                    coroutineScope.launch {
                        try {
                            val joinedGroup = groupController.joinGroup(authToken.value!!, code)
                            group.value = joinedGroup?.let { groupController.getGroupDetails(it.id) }
                        } catch (e: Exception) {
                            errorMessage = e.message ?: unknownError
                        } finally {
                            isLoading = false
                        }
                    }
                } else {
                    errorMessage = noCodeOrAuth
                }
            }) {
                Text(stringResource(Res.string.join_group))
            }


        }
        Text(text = membersAmount)
        LazyColumn(
            modifier = Modifier
                .fillMaxSize()
                .padding(8.dp)
        ) {
            if (members.isNullOrEmpty()) {
                item {
                    Text(text = stringResource(Res.string.no_members))
                }
            } else {
                item {
                    UserItemTable(
                        users = members,
                        onKickClick = { user -> kickMember(user) },
                        showKickButton = isOwner,
                        currentUserId = user.value?.id,
                        ownerId = group.value?.ownerId
                    )
                }
            }
        }

        Spacer(modifier = Modifier.height(8.dp))

        if (isLoading) {
            CircularProgressIndicator()
        }

        if (errorMessage.isNotEmpty()) {
            Text(text = errorMessage, color = Color.Red)
        }

        if (showConfirmationDialog) {
            ConfirmDeleteDialog(
                title = stringResource(Res.string.confirm_disband),
                message = stringResource(Res.string.disband_confirm_intent),
                confirmButtonText = stringResource(Res.string.disband),
                dismissButtonText = stringResource(Res.string.cancel),
                onConfirm = { disbandGroup() },
                onDismiss = { showConfirmationDialog = false }
            )
        }

    }
}
```

2.1.2.1.7.3. UserManagementSection.kt

```
package org.psyche.assistant.Composable.Sections

import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.launch
import org.jetbrains.compose.resources.stringResource
import org.psyche.assistant.Composable.Dialogs.ConfirmDeleteDialog
import org.psyche.assistant.Composable.LocalAuthToken
import org.psyche.assistant.Composable.LocalGroup
import org.psyche.assistant.Composable.LocalUser
import org.psyche.assistant.Controller.UserController
import psycheassistant.composeapp.generated.resources.*

/**
 * User management section
 * Used to manage user-specific settings. I.e. registering and/or logging in/out of a specific user.
 */
@Composable
fun UserManagementSection() {
    val coroutineScope = rememberCoroutineScope()
    val userController = UserController()

    var authToken = LocalAuthToken.current
    var user = LocalUser.current
    var group = LocalGroup.current
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var isLoading by remember { mutableStateOf(false) }
    var errorMessage by remember { mutableStateOf("") }

    var unknownError = stringResource(Res.string.unknown_error)
    var noEmailOrPass = stringResource(Res.string.no_email_or_pass)
    var showConfirmationDialog by remember { mutableStateOf(false) }

    val deleteUser = {
        isLoading = true
        coroutineScope.launch {
            try {
                userController.deleteUser(authToken.value!!)
                authToken.value = null
                user.value = null
                group.value = null
            } catch (e: Exception) {
                errorMessage = e.message ?: unknownError
            } finally {
                isLoading = false
            }
        }
    }

    Column(
        modifier = Modifier
        .fillMaxWidth()
        .padding(8.dp)
    ) {
        if (authToken.value != null) {
            Text(
                text = stringResource(Res.string.logged_in_as, user.value?.email ?: ""),
                style = MaterialTheme.typography.h6)
            Spacer(modifier = Modifier.height(16.dp))

            Row {
                Button(onClick = {
                    isLoading = true
                    coroutineScope.launch {
                        try {
                            userController.signOutUser()
                            authToken.value = null
                            user.value = null
                            group.value = null
                        } catch (e: Exception) {
                            errorMessage = e.message ?: unknownError
                        } finally {
                            isLoading = false
                        }
                    }
                }) {
                    Text(stringResource(Res.string.logout))
                }
                Spacer(modifier = Modifier.padding(15.dp))
```

```
                Button(
                    onClick = {
                        showConfirmationDialog = true
                    }, colors = ButtonDefaults.buttonColors(
                        backgroundColor = Color.Red,
                        contentColor = Color.Black
                    )
                ) {
                    Text(stringResource(Res.string.delete_user))
                }
            }
        } else {
            OutlinedTextField(
                value = email,
                onValueChange = { email = it },
                label = { Text(stringResource(Res.string.email)) },
                modifier = Modifier.fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(8.dp))

            OutlinedTextField(
                value = password,
                onValueChange = { password = it },
                label = { Text(stringResource(Res.string.password)) },
                modifier = Modifier.fillMaxWidth()
            )

            Spacer(modifier = Modifier.height(8.dp))

            Button(onClick = {
                if (email.isNotBlank() && password.isNotBlank()) {
                    isLoading = true
                    coroutineScope.launch {
                        try {
                            authToken.value = userController.authenticateUser(email, password)
                            user.value = userController.getUserProfile(authToken.value!!)
                        } catch (e: Exception) {
                            errorMessage = e.message ?: unknownError
                        } finally {
                            isLoading = false
                        }
                    }
                } else {
                    errorMessage = noEmailOrPass
                }
            }) {
                Text(stringResource(Res.string.login))
            }

            Spacer(modifier = Modifier.height(8.dp))

            Button(onClick = {
                if (email.isNotBlank() && password.isNotBlank()) {
                    isLoading = true
                    coroutineScope.launch {
                        try {
                            userController.registerNewUser(email, password)
                            authToken.value = userController.authenticateUser(email, password)
                        } catch (e: Exception) {
                            errorMessage = e.message ?: unknownError
                        } finally {
                            isLoading = false
                        }
                    }
                } else {
                    errorMessage = noEmailOrPass
                }
            }) {
                Text(stringResource(Res.string.register))
            }
        }

        Spacer(modifier = Modifier.height(8.dp))

        if (isLoading) {
            CircularProgressIndicator()
        }

        if (errorMessage.isNotEmpty()) {
            Text(text = errorMessage, color = Color.Red)
        }
       }

       if (showConfirmationDialog) {
        ConfirmDeleteDialog(
            title = stringResource(Res.string.confirm_delete),
            message = stringResource(Res.string.delete_confirm_intent),
            confirmButtonText = stringResource(Res.string.delete),
            dismissButtonText = stringResource(Res.string.cancel),
            onConfirm = { deleteUser() },
            onDismiss = { showConfirmationDialog = false }
        )
       }
      }
```

2.1.2.1.8. GlobalState.kt

```
package org.psyche.assistant.Composable

import androidx.compose.runtime.CompositionLocal
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.compositionLocalOf
import org.psyche.assistant.Model.Group.Group
import org.psyche.assistant.Model.User.User

val LocalAuthToken = compositionLocalOf<MutableState<String?>> { error("No token found..") }
val LocalUser = compositionLocalOf<MutableState<User?>> { error("No user found..") }
val LocalGroup = compositionLocalOf<MutableState<Group?>> { error("No group found..") }
```

2.1.2.2. Controller

2.1.2.2.1. ActivityController

```
package org.psyche.assistant.Controller

import kotlinx.datetime.LocalDate
import org.psyche.assistant.Model.Activity.Activity
import org.psyche.assistant.Service.ActivityService

class ActivityController {
    private val activityService = ActivityService()

    suspend fun getHandledActivityByPeriod(groupId: Int, startDate: LocalDate, endDate: LocalDate): List<Activity> {
        return activityService.getHandledActivityByPeriod(groupId, startDate, endDate)
    }

    suspend fun getActivityByPeriod(groupId: Int, startDate: LocalDate, endDate: LocalDate): List<Activity> {
        return activityService.getActivityByPeriod(groupId, startDate, endDate)
    }

    suspend fun getActivityForToday(groupId: Int, today: LocalDate): List<Activity> {
        return activityService.getActivityForToday(groupId, today)
    }

    suspend fun createActivity(authToken: String, deadline: String, description: String, energyCost: Int): Activity? {
        return activityService.createActivity(authToken, deadline, description, energyCost)
    }

    suspend fun completeActivity(authToken: String, activityId: Int): Activity? {
        return activityService.completeActivity(authToken, activityId)
    }

    suspend fun deleteActivity(authToken: String, activityId: Int): Boolean {
        return activityService.deleteActivity(authToken, activityId)
    }

}
```

2.1.2.2.2. GroupController

```
package org.psyche.assistant.Controller

import org.psyche.assistant.Model.Group.Group
import org.psyche.assistant.Service.GroupService

class GroupController {
    private val groupService = GroupService()

    suspend fun getGroupDetails(id: Int): Group? {
        return groupService.getGroupDetails(id)
    }

    suspend fun createGroup(authToken: String): Group? {
        return groupService.createGroup(authToken)
    }
    suspend fun joinGroup(authToken: String, code: String): Group? {
        return groupService.joinGroup(authToken, code)
    }

    suspend fun kickMember(authToken: String, groupId: Int, userId: Int): Group? {
        return groupService.removeUserFromGroup(authToken, groupId, userId)
    }

    suspend fun leaveGroup(authToken: String, groupId: Int): Boolean {
        return groupService.leaveGroup(authToken, groupId)
    }

    suspend fun disbandGroup(authToken: String, groupId: Int): Boolean {
        return groupService.disbandGroup(authToken, groupId)
    }
}
```

2.1.2.2.1. UserController

```
package org.psyche.assistant.Controller

import org.psyche.assistant.Model.User.User
import org.psyche.assistant.Service.UserService


class UserController {
    private val userService = UserService()
    suspend fun getUserProfile(authToken: String): User? {
        return userService.getUserDetails(authToken)
    }

    suspend fun registerNewUser(email: String, password: String): String {
        return userService.registerUser(email, password)
    }

    suspend fun authenticateUser(email: String, password: String): String {
        return userService.loginUser(email, password)
    }

    suspend fun signOutUser() {
        userService.logoutUser()
    }

    suspend fun deleteUser(authToken: String): Boolean {
        return userService.deleteUser(authToken)
    }

    suspend fun getUserById(id: Int): User? {
        return userService.getUserById(id)
    }
}
```

2.1.2.3. Helper

2.1.2.3.1. CustomTheme

```
package org.psyche.assistant.Helper

import androidx.compose.material.Colors
import androidx.compose.material.lightColors
import androidx.compose.ui.graphics.Color

class CustomTheme {
    companion object {
        fun psycheColors(
            primary: Color = Color(0xFF006400),
            onPrimary: Color = Color.White,
            secondary: Color = Color(0xFF444444),
            onSecondary: Color = Color(0xFF00FF00),
            background: Color = Color(0xFF121212),
            onBackground: Color = Color(0xFFB2B2B2),
            surface: Color = Color(0xFF1C1C1C),
            onSurface: Color = Color(0xFF00FF00),
            error: Color = Color(0xFFB00020),
            onError: Color = Color.White,
        ): Colors {
            return lightColors(
                primary = primary,
                onPrimary = onPrimary,
                secondary = secondary,
                onSecondary = onSecondary,
                background = background,
                onBackground = onBackground,
                surface = surface,
                onSurface = onSurface,
                error = error,
                onError = onError,
            )
        }
    }
}
```

2.1.2.3.2. LevelColor

```
package org.psyche.assistant.Helper

import androidx.compose.ui.graphics.Color
import org.psyche.assistant.Model.SurveyModel
import org.psyche.assistant.Survey

/**
 * Store hard-coded colors in separate model,
 * just to avoid interfering with UI models, but ensuring uniformity in text color across platforms.
 */
object LevelColor {
    val blue = Color(0xFF2196F3)
    val green = Color(0xFF4CAF50)
    val yellow = Color(0xFFFFEB3B)
    val red = Color(0xFFF44336)
    fun getColor(score: Double): Color {
        when {
            score <= 10 -> return blue
            score <= 15 -> return green
            score <= 20 -> return yellow
            else -> return red
        }

    }
}
```

2.1.2.3.3. RoundingHelper

```
import kotlin.math.roundToInt

/**
 * Helper function that allows us to format a double to decimal points as needed within Kotlin,
 * without having to rely on platform specificity.
 */
fun Double.roundToDecimals(decimals: Int): Double {
    var dotAt = 1
    repeat(decimals) { dotAt *= 10 }
    val roundedValue = (this * dotAt).roundToInt()
    var result = (roundedValue / dotAt) + (roundedValue % dotAt).toDouble() / dotAt
    return result
}
```

2.1.2.4. Model

2.1.2.4.1. Activity

2.1.2.4.1.1. Activity

```
package org.psyche.assistant.Model.Activity

import kotlinx.datetime.LocalDate
import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable

@Serializable
data class Activity (
    val id: Int,
    val description: String,
    val energyCost: Int,

    // Clarify field name since back-end operates with objects, and front-end with IDs
    @SerialName("group")
    val groupId: Int? = null,

    @SerialName("owner")
    val ownerId: Int? = null,

    @SerialName("handledBy")
    val handledById: Int? = null,
    val deadline: LocalDate,
    val handledOn: LocalDate? = null,
    val completed: Boolean = false
)
```

2.1.2.4.1.2. ActivityRepository

```
package org.psyche.assistant.Model.Activity

import kotlinx.datetime.LocalDate

interface ActivityRepository {
    suspend fun getHandledActivityByPeriod(groupId: Int, startDate: LocalDate, endDate: LocalDate): List<Activity>
    suspend fun getActivityByPeriod(groupId: Int, startDate: LocalDate, endDate: LocalDate): List<Activity>
    suspend fun getActivityForToday(groupId: Int, today: LocalDate): List<Activity>
    suspend fun getActivity(activityId: Int): Activity?
    suspend fun createActivity(authToken: String, deadline: String, description: String, energyCost: Int): Activity?
    suspend fun completeActivity(authToken: String, activityId: Int): Activity?
    suspend fun deleteActivity(authToken: String, activityId: Int): Boolean
}
```

2.1.2.4.2. Group

2.1.2.4.2.1. Group

```
package org.psyche.assistant.Model.Group

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable

@Serializable
data class Group (
    val id: Int,

    // Clarify field name since back-end operates with objects, and front-end with IDs
    @SerialName("users")
    val userIds: List<Int>,
    @SerialName("activities")
    val activityIds: List<Int>,

    @SerialName("owner")
    val ownerId: Int?,

    val code: String,
)
```

2.1.2.4.2.2. GroupRepository

```
package org.psyche.assistant.Model.Group

interface GroupRepository {
    suspend fun createGroup(authToken: String): Group?
    suspend fun joinGroup(authToken: String, code: String): Group?
    suspend fun leaveGroup(authToken: String, groupId: Int): Boolean
    suspend fun disbandGroup(authToken: String, groupId: Int): Boolean
    suspend fun getGroupDetails(id: Int): Group?
    suspend fun removeUserFromGroup(authToken: String, groupId: Int, userId: Int): Group?
}
```

2.1.2.4.3. User

2.1.2.4.3.1. User

```
package org.psyche.assistant.Model.User

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable

@Serializable
data class User (
    val id: Int,
    val email: String,
    val password: String,
    val energyExpenditure: Int,

    // Clarify field name since back-end operates with objects, and front-end with IDs
    @SerialName("group")
    val groupId: Int? = null,
)
```

2.1.2.4.3.2. UserRepository

```
package org.psyche.assistant.Model.User


interface UserRepository {
    suspend fun getUserDetails(authToken: String): User?
    suspend fun registerUser(email: String, password: String): String
    suspend fun loginUser(email: String, password: String): String
    suspend fun logoutUser()
    suspend fun deleteUser(authToken: String): Boolean
    suspend fun getUserById(id: Int): User?
}
```

2.1.2.5. Service

2.1.2.5.1. ActivityService

```
package org.psyche.assistant.Service

import io.ktor.client.*
import io.ktor.client.request.*
import io.ktor.client.statement.*
import io.ktor.http.*
import kotlinx.datetime.LocalDate
import kotlinx.serialization.encodeToString
import kotlinx.serialization.json.Json
import org.psyche.assistant.Model.Activity.Activity
import org.psyche.assistant.Model.Activity.ActivityRepository

/**
 * Activity service
 * Makes use of the ActivityRepository interface to ensure functionality is present.
 * @constructor Create empty Activity service
 */
class ActivityService : ActivityRepository {
    private val client = HttpClient()

    /**
     * Get handled activity by period
     * Note that this looks specifically at when activities were handled; not their deadline.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    override suspend fun getHandledActivityByPeriod(groupId: Int, startDate: LocalDate, endDate: LocalDate):
List<Activity> {
        val url = ServiceBuilder.url("api/activities/group/$groupId/handled/period?startDate=${startDate}&endDate=$
{endDate}")
        val response: HttpResponse = client.get(url) {
            contentType(ContentType.Application.Json)
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<List<Activity>>(responseBody)
            }
            else -> emptyList()
        }
    }

    /**
     * Get activity by period
     * Note this looks on the activities deadline.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    override suspend fun getActivityByPeriod(groupId: Int, startDate: LocalDate, endDate: LocalDate): List<Activity> {
        val url = ServiceBuilder.url("api/activities/group/$groupId/period?startDate=${startDate}&endDate=${endDate}")
        val response: HttpResponse = client.get(url) {
            contentType(ContentType.Application.Json)
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<List<Activity>>(responseBody)
            }
            else -> emptyList()
        }
    }

    /**
     * Get activity for today
     * Gets only activities relevant for today' date.
     * @param groupId
     * @param today
     * @return
     */
    override suspend fun getActivityForToday(groupId: Int, today: LocalDate): List<Activity> {
        val url = ServiceBuilder.url("api/activities/group/$groupId?date=${today}")
        val response: HttpResponse = client.get(url) {
            contentType(ContentType.Application.Json)
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                val activities = Json.decodeFromString<List<Activity>>(responseBody)

                activities.filter { activity ->
                    activity.deadline == today
                }
            }
            else -> emptyList()
        }
    }

    /**
     * Get activity
     * Fetches an activity from API based on the passed ID
     * @param activityId
     * @return
     */
    override suspend fun getActivity(activityId: Int): Activity?{
        val url = ServiceBuilder.url("api/activities/$activityId")
        val response: HttpResponse = client.get(url) {
            contentType(ContentType.Application.Json)
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Activity>(responseBody)
            }
            else -> null
        }
    }

    /**
     * Create activity
     * Creates an activity so long as authToken is valid.
     * @param authToken
     * @param deadline
     * @param description
     * @param energyCost
     * @return
     */
    override suspend fun createActivity(authToken: String, deadline: String, description: String, energyCost: Int):
Activity? {
        val url = ServiceBuilder.url("api/activities/create")
        val requestPayload = Json.encodeToString(mapOf(
            "deadline" to deadline,
            "description" to description,
            "energyCost" to energyCost.toString()
        ))

        val response: HttpResponse = client.post(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
            setBody(requestPayload)
        }

        return when (response.status) {
            HttpStatusCode.Created -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Activity>(responseBody)
            }
            else -> null
        }
    }

    /**
     * Complete activity
     * API call to complete activities, requires valid authToken.
     * @param authToken
     * @param activityId
     * @return
     */
    override suspend fun completeActivity(authToken: String, activityId: Int): Activity? {
        val url = ServiceBuilder.url("api/activities/$activityId/complete")
        val response: HttpResponse = client.post(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Activity>(responseBody)
            }
            else -> null
        }
    }

    /**
     * Delete activity
     * Deletes activity based solely on ID.
     * @param activityId
     * @return
     */
    override suspend fun deleteActivity(authToken: String, activityId: Int): Boolean {
        val url = ServiceBuilder.url("api/activities/$activityId/delete")
        val response: HttpResponse = client.delete(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
        }
        return when (response.status) {
            HttpStatusCode.NoContent -> true
            HttpStatusCode.NotFound -> false
            else -> throw Exception(response.status.toString())
        }
    }
}
```

2.1.2.5.2. ConfigProvider

```
package org.psyche.assistant.Service

/**
 * Expect implementation for using platform-specific configuration file for fetching URL
 */
expect object ConfigProvider {
    val baseUrl: String
}
```

2.1.2.5.3. GroupService

```
package org.psyche.assistant.Service

import io.ktor.client.*
import io.ktor.client.request.*
import io.ktor.client.statement.*
import io.ktor.http.*
import kotlinx.serialization.encodeToString
import kotlinx.serialization.json.Json
import org.psyche.assistant.Model.Group.Group
import org.psyche.assistant.Model.Group.GroupRepository

class GroupService : GroupRepository {
    private val client = HttpClient()

    /**
     * Create group
     * Creates a group via API, given a valid authToken is supplied.
     * @param authToken
     * @return
     */
    override suspend fun createGroup(authToken: String): Group? {
        val url = ServiceBuilder.url("api/groups/register")
        val response: HttpResponse = client.post(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
        }
        return when (response.status) {
            HttpStatusCode.Created -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Group>(responseBody)
            }
            else -> null
        }
    }
```

```kotlin
/**
 * Join group
 * Joins a group designated by the groups 6-character code (viewable by members of that group)
 * so long as there is also a valid authToken.
 * @param authToken
 * @param code
 * @return
 */
override suspend fun joinGroup(authToken: String, code: String): Group? {
    val url = ServiceBuilder.url("api/groups/join")
    val response: HttpResponse = client.post(url) {
        contentType(ContentType.Application.Json)
        header(HttpHeaders.Authorization, "Bearer $authToken")
        setBody(Json.encodeToString(mapOf("code" to code)))
    }
    return when (response.status) {

        HttpStatusCode.OK -> {
            val responseBody = response.bodyAsText()
            Json.decodeFromString<Group>(responseBody)
        }
        else -> null
    }
}

/**
 * Leave group
 * Removes the user from the group, validation of ownership, relations etc., is handled by backend service.
 * @param authToken
 * @param groupId
 * @return
 */
override suspend fun leaveGroup(authToken: String, groupId: Int): Boolean {
    val url = ServiceBuilder.url("api/groups/${groupId}/leave")
    val response: HttpResponse = client.post(url) {
        contentType(ContentType.Application.Json)
        header(HttpHeaders.Authorization, "Bearer $authToken")
    }

    return when (response.status) {
        HttpStatusCode.NoContent -> true
        HttpStatusCode.NotFound -> false
        else -> throw Exception(response.status.toString())
    }
}

/**
 * Disband group
 * Deletes the group and its relations to members and activities; i.e. nullifies the relation - orphans remain.
 * @param authToken
 * @param groupId
 * @return
 */
override suspend fun disbandGroup(authToken: String, groupId: Int): Boolean {
    val url = ServiceBuilder.url("api/groups/${groupId}/disband")
    val response: HttpResponse = client.delete(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
        }

        return when (response.status) {
            HttpStatusCode.NoContent -> true
            HttpStatusCode.NotFound -> false
            else -> throw Exception(response.status.toString())
        }
    }

    /**
     * Get group details
     * General purpose read function for specific group. Note that no personal or critical data is stored, therefore no
authentication is required.
     * @param id
     * @return
     */
    override suspend fun getGroupDetails(id: Int): Group? {
        val url = ServiceBuilder.url("api/groups/$id")
        val response: HttpResponse = client.get(url) {
            contentType(ContentType.Application.Json)
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Group>(responseBody)
            }
            else -> null
        }
    }

    /**
     * Remove user from group
     * Removes the user in question from the group. Note that this can be used either to kick or voluntarily leave the
group
     * included direct reference from unimplemented version to clarify this.
     * @param authToken
     * @param groupId
     * @param userId
     * @return
     */
    override suspend fun removeUserFromGroup(authToken: String, groupId: Int, userId: Int): Group? {
        val url = ServiceBuilder.url("api/groups/$groupId/kick")
        val response: HttpResponse = client.post(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
            setBody(Json.encodeToString(mapOf("userId" to userId)))
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Group>(responseBody)
            }
            else -> null
        }
    }


}
```

2.1.2.5.4. ServiceBuilder

```
package org.psyche.assistant.Service

import io.ktor.client.HttpClient
import io.ktor.client.engine.cio.*
import io.ktor.client.plugins.contentnegotiation.ContentNegotiation
import io.ktor.serialization.kotlinx.json.*
import kotlinx.serialization.json.Json

object ServiceBuilder {
    private val BASE_URL = ConfigProvider.baseUrl

    val client = HttpClient(CIO) {
        install(ContentNegotiation) {
            json(Json {
                prettyPrint = true
                isLenient = true
                ignoreUnknownKeys = true
            })
        }
    }

    /**
     * Helper method to ensure controllers can be URL agnostic.
     */
    fun url(path: String): String = "$BASE_URL/$path"
}
```

2.1.2.5.5. UserService

```
package org.psyche.assistant.Service

import io.ktor.client.*
import io.ktor.client.request.*
import io.ktor.client.statement.*
import io.ktor.http.*
import kotlinx.serialization.json.Json
import org.psyche.assistant.Model.User.User
import org.psyche.assistant.Model.User.UserRepository
import org.psyche.assistant.Storage.AuthStorage

class UserService : UserRepository {
    private val client = HttpClient()

    /**
     * Get user details
     * Fetches the users details, as well as revalidating or refreshing tokens.
     * @param authToken
     * @return
     */
    override suspend fun getUserDetails(authToken: String): User? {
        val url = ServiceBuilder.url("api/users/me")
        var response: HttpResponse = client.get(url) {
            header(HttpHeaders.Authorization, "Bearer $authToken")
        }

        if (response.status == HttpStatusCode.Unauthorized) {
            // Token might be expired, try to refresh it
            val newToken = refreshToken()
            if (newToken != null) {
                AuthStorage.saveAuthToken(newToken)
                response = client.get(url) {
                    header(HttpHeaders.Authorization, "Bearer $newToken")
                }
            } else {
                // Refresh token failed or is not available
                throw Exception("Token refresh failed, user needs to log in again")
            }
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<User>(responseBody)
            }
            else -> null
        }
    }

    /**
     * Register user
     * Register the user using the supplied credentials. Verify that email and password live up to security standards,
before posting to server.
     * @param email
     * @param password
     * @return
     */
    override suspend fun registerUser(email: String, password: String): String {
        validateEmail(email)
        validatePassword(password)

        val url = ServiceBuilder.url("api/users/register")
        val response: HttpResponse = client.post(url) {
            contentType(ContentType.Application.FormUrlEncoded)
            setBody("email=$email&password=$password")
        }
        return when (response.status) {
            HttpStatusCode.Created -> response.bodyAsText()
            else -> throw Exception(response.bodyAsText())
        }
    }

    /**
     * Login user
     * Logs in the user based on supplied credentials, and saves JWT access/refresh tokens.
     * @param email
     * @param password
     * @return
     */
    override suspend fun loginUser(email: String, password: String): String {
        val url = ServiceBuilder.url("api/users/login")
        val response: HttpResponse = client.post(url) {
            contentType(ContentType.Application.FormUrlEncoded)
            setBody("email=$email&password=$password")
        }
        if (response.status == HttpStatusCode.OK) {
            val responseBody = response.bodyAsText()
            val tokens = Json.decodeFromString<Map<String, String>>(responseBody)
            val accessToken = tokens["accessToken"]
            val refreshToken = tokens["refreshToken"]

            if (accessToken != null && refreshToken != null) {
                AuthStorage.saveAuthToken(accessToken)
                AuthStorage.saveRefreshToken(refreshToken)
                return accessToken
            } else {
                throw Exception("Invalid login response")
            }
        } else {
            throw Exception(response.bodyAsText())
        }
    }

    /**
     * Logout user
     * Logs out the user by rescinding identification tokens, meaning the user now must log back in to access external
data.
     */
    override suspend fun logoutUser() {
        AuthStorage.removeAuthToken()
        AuthStorage.removeRefreshToken()
    }

    /**
     * Delete user
     * Removes the user from the database.
     * @param authToken
     * @return
     */
    override suspend fun deleteUser(authToken: String): Boolean {
        val url = ServiceBuilder.url("api/users/delete")
        val response: HttpResponse = client.delete(url) {
            contentType(ContentType.Application.Json)
            header(HttpHeaders.Authorization, "Bearer $authToken")
        }

        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<Boolean>(responseBody)
            }
            else -> false
        }
    }


    override suspend fun getUserById(id: Int): User? {
        val url = ServiceBuilder.url("api/users/$id")
        val response: HttpResponse = client.get(url) {
            contentType(ContentType.Application.Json)
        }
        return when (response.status) {
            HttpStatusCode.OK -> {
                val responseBody = response.bodyAsText()
                Json.decodeFromString<User>(responseBody)
            }
            else -> null
        }
    }

    /**
     * Refresh token
     * Requests a refresh token from the API.
     * @return
     */
    private suspend fun refreshToken(): String? {
        val refreshToken = AuthStorage.getRefreshToken() ?: return null

        val url = ServiceBuilder.url("api/users/refresh-token")
        val response: HttpResponse = client.post(url) {
            header(HttpHeaders.Authorization, "Bearer $refreshToken")
        }

        return if (response.status == HttpStatusCode.OK) {
            val newAccessToken = response.bodyAsText()
            newAccessToken
        } else {
            // Handle refresh token failure
            AuthStorage.removeAuthToken()
            AuthStorage.removeRefreshToken()
            null
        }
    }

    /**
     * Validate email
     * Helper function to validate email format when necessary.
     * @param email
     */
    private fun validateEmail(email: String) {
        val emailRegex = "^[A-Za-z](.*)([@]{1})(.{1,})(\\.)(.{1,})"
        if (!email.matches(emailRegex.toRegex())) {
            throw RuntimeException("Invalid email format")
        }
    }

    /**
     * Validate password
     * Helper function to validate password lives up to certain requirements.
     * @param password
     */
    private fun validatePassword(password: String) {
        if (password.length < 8 ||
            !password.any { it.isUpperCase() } ||
            !password.any { it.isLowerCase() } ||
            !password.any { it.isDigit() } ||
            !password.any { !it.isLetterOrDigit() }
        ) {
            throw RuntimeException("Password must be at least 8 characters long and include uppercase, lowercase, digit,
and special character")
        }
    }
}
```

2.1.2.6. Storage

2.1.2.6.1. AuthStorage

```
package org.psyche.assistant.Storage

/**
 * Auth storage
 * Expect object for platform specific implementation for storing tokens securely on the physical device itself.
 * See PsycheAssistant/composeApp/src/androidMain/kotlin/Storage/AuthStorage.android.kt
 * for Actual implementation.
 * @constructor Create empty Auth storage
 */
expect object AuthStorage {
    fun saveAuthToken(token: String)
    fun getAuthToken(): String?
    fun removeAuthToken()
    fun saveRefreshToken(token: String)
    fun getRefreshToken(): String?
    fun removeRefreshToken()
}
```

2.1.3. resources

2.1.3.1. service.properties

```
BASE_URL=https://192.168.1.145:8443
auth_token=A#1ef!
refresh_token=&#gea1
```

2.2. androidMain

2.2.1. kotlin

2.2.1.1. org.psyche.assistant

2.2.1.1.1. MainActivity.kt

```
package org.psyche.assistant

import android.content.Context
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.tencent.mmkv.MMKV
import org.psyche.assistant.Composable.Main.PsycheAssistantApp

lateinit var appContext: Context

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        appContext = applicationContext
        MMKV.initialize(this)

        setContent {
            PsycheAssistantApp()
        }
    }
}
```

2.2.1.2. Service

2.2.1.2.1. ConfigProvider.android.kt

```
package org.psyche.assistant.Service

import org.psyche.assistant.appContext
import java.io.InputStream
import java.util.*

actual object ConfigProvider {
    private val properties: Properties = Properties().apply {
        val inputStream: InputStream = appContext.assets.open("service.properties")
        load(inputStream)
    }

    actual val baseUrl: String by lazy {
        properties.getProperty("BASE_URL")
    }}
```

2.2.1.3. Storage

2.2.1.3.1. AuthStorage.android.kt

```
package org.psyche.assistant.Storage

import com.tencent.mmkv.MMKV
import org.psyche.assistant.Storage.AuthStorage.properties
import org.psyche.assistant.appContext
import java.io.InputStream
import java.util.*

/**
 * Auth storage
 * Actual implementation for storing tokens safely locally, using keys to encode said token.
 * Uses Memory-Mapped Key-Value for secure storage (as opposed to SharedPreferences)
 * @constructor Create empty Auth storage
 */
actual object AuthStorage {
   private val properties: Properties = Properties().apply {
      val inputStream: InputStream = appContext.assets.open("service.properties")
      load(inputStream)
   }

   val AUTH_TOKEN_KEY: String by lazy {
      properties.getProperty("auth_token")
   }
   val REFRESH_TOKEN_KEY: String by lazy {
      properties.getProperty("refresh_token")
   }

   actual fun saveAuthToken(token: String) {
      MMKV.defaultMMKV().encode(AUTH_TOKEN_KEY, token)
   }

   actual fun getAuthToken(): String? {
      return MMKV.defaultMMKV().decodeString(AUTH_TOKEN_KEY)
   }

   actual fun removeAuthToken() {
      MMKV.defaultMMKV().remove(AUTH_TOKEN_KEY)
   }

   actual fun saveRefreshToken(token: String) {
      MMKV.defaultMMKV().encode(REFRESH_TOKEN_KEY, token)
   }

   actual fun getRefreshToken(): String? {
      return MMKV.defaultMMKV().decodeString(REFRESH_TOKEN_KEY)
   }

   actual fun removeRefreshToken() {
      MMKV.defaultMMKV().remove(REFRESH_TOKEN_KEY)
   }
}
```

2.2.2. res

2.2.2.1. raw

2.2.2.1.1. psyche.crt

```
-----BEGIN CERTIFICATE-----
MIIDfjCCAmagAwIBAgIUWyIrL2HN1U2iIq/DBz+5jXTXWTgwDQYJKoZIhvcNAQEL
BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM
GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAgFw0yNDA4MjYyMzA4NDdaGA8yMTI0
MDgwMjIzMDg0N1owRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUx
ITAfBgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAMyJglTKD8LyCkhAcEfveir9XAlsxRzZLEiKEgTl
RRDbjKDsTGGkspJT4vAks0lLhovH5bcNjHRG6j0QZIyXdyL1eup7Fhw8yPZexkQV
aBSzbSK2/k0ZvJ7Gzg03JmKy9gXfeE1l9ZzR43VeGoW2XjWNoAzyny8Gvxl7U248
/BcY3/Bm19L5uzQfOwIcBAUwLgYXum+kn7hclGVwIBcbuWrkisulfrADc4xviJU2
V643VZjilAKDpCWcUso3UG/UwnwMe3tNOMOqttEZJkxNck+oOYpVU9sipBvkAV2X
wNtVI9Id7LveUNuGgYOtrikRfTZu7oGH/2zVluVhtVjf8/kCAwEAAaNkMGIwHQYD
VR0OBBYEFGj6+L3j/9Oyd2+CTDCNtLYAJJZfMB8GA1UdIwQYMBaAFGj6+L3j/9Oy
d2+CTDCNtLYAJJZfMA8GA1UdEwEB/wQFMAMBAf8wDwYDVR0RBAgwBocEwKgBkTAN
BgkqhkiG9w0BAQsFAAOCAQEAhIHii5rMirahAMoWrmiNqELH+UOBMzCiR+bMU7Pv
ZwXudFBa8mCGXKU9iuTookH8n57YG0sqh3RClSxLZEKDmshUTqIvhmRppqu+Q1aN
UjmzdunI1kBFEPidYHCozhqx66loRf0N2VqJB4f/LXwuqgb3LtT7WDTmcS3WaIQw
AHYteDPLXVUVNMRXWN/bybWxGSLYvzgC5+HQdaFc1azvHZToELp31cAzOmE9vwSI
R36ITz13u4ENPIt1gFi1hjUBimLxNqfOFjNwtzq35w2zQNZNrhLd3y2sSkR8+MxN
UvUVey1kn1qoMK/pNrugb5CUUex/F2OG3myc9ww8BLfXsQ==
-----END CERTIFICATE-----
```

2.2.2.2. xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="@raw/psyche"/>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

2.2.3. AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.Material.Light.NoActionBar"
        android:networkSecurityConfig="@xml/network_security_config">
        <activity
        android:exported="true"
        android:configChanges="orientation|screenSize|screenLayout|keyboardHidden|mnc|colorMode|density|
fontScale|fontWeightAdjustment|keyboard|layoutDirection|locale|mcc|navigation|smallestScreenSize|touchscreen|
uiMode"
        android:name=".MainActivity">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

2.3. build.gradle.kts

```
import org.jetbrains.compose.desktop.application.dsl.TargetFormat
import org.jetbrains.kotlin.gradle.ExperimentalKotlinGradlePluginApi
import org.jetbrains.kotlin.gradle.dsl.JvmTarget

plugins {
    alias(libs.plugins.kotlinMultiplatform)
    alias(libs.plugins.androidApplication)
    alias(libs.plugins.jetbrainsCompose)
    alias(libs.plugins.compose.compiler)
    alias(libs.plugins.sqlDelight)

    // New:
    kotlin("plugin.serialization")
}

kotlin {
    androidTarget {
        compilations.all {
            kotlinOptions {
                jvmTarget = "11"
            }
        }
        // Note: This prevented the use of SQLDelight.
//        @OptIn(ExperimentalKotlinGradlePluginApi::class)
//        compilerOptions {
//            jvmTarget.set(JvmTarget.JVM_11)
//        }
    }

    // Initiate SQLDelight database
    sqldelight {
        databases {
            create("PsycheAssistantDB") {
                packageName = "org.psyche.assistant"
            }
        }
    }

    listOf(
        iosX64(),
        iosArm64(),
        iosSimulatorArm64()
    ).forEach { iosTarget ->
        iosTarget.binaries.framework {
            baseName = "ComposeApp"
            isStatic = true
        }
    }

    sourceSets {
        commonMain.dependencies {
            implementation(compose.runtime)
            implementation(compose.foundation)
            implementation(compose.material)
            implementation(compose.ui)
            implementation(compose.components.resources)
            implementation(compose.components.uiToolingPreview)
            implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.5.0")
            implementation("org.jetbrains.kotlinx:kotlinx-datetime:0.6.0")
            implementation(libs.sqldelight.coroutines)
            implementation("io.ktor:ktor-client-core:2.3.3")
            implementation("io.ktor:ktor-client-content-negotiation:2.3.3")
            implementation("io.ktor:ktor-serialization-kotlinx-json:2.3.3")
            implementation("io.ktor:ktor-client-cio:2.3.3")
            implementation("com.tencent:mmkv:1.2.10")
            implementation("org.jetbrains.kotlinx:kotlinx-datetime:0.4.0")
            implementation("network.chaintech:kmp-date-time-picker:1.0.3")
        }
        androidMain.dependencies {
            implementation(compose.preview)
            implementation(libs.androidx.activity.compose)
            implementation(libs.sqldelight.android)
            implementation("io.ktor:ktor-client-okhttp:2.3.3")
            implementation("com.squareup.okhttp3:okhttp:4.10.0")
        }


        iosMain.dependencies {
            implementation(libs.sqldelight.native)

        }

    }
}

android {
    namespace = "org.psyche.assistant"
    compileSdk = libs.versions.android.compileSdk.get().toInt()

    sourceSets["main"].manifest.srcFile("src/androidMain/AndroidManifest.xml")
    sourceSets["main"].res.srcDirs("src/androidMain/res")
    sourceSets["main"].resources.srcDirs("src/commonMain/resources")
    sourceSets["main"].assets.srcDirs("src/commonMain/resources")
    defaultConfig {
        applicationId = "org.psyche.assistant"
        minSdk = libs.versions.android.minSdk.get().toInt()
        targetSdk = libs.versions.android.targetSdk.get().toInt()
        versionCode = 1
        versionName = "1.0"
    }
    packaging {
        resources {
            excludes += "/META-INF/{AL2.0,LGPL2.1}"
        }
    }
    buildTypes {
        getByName("release") {
            isMinifyEnabled = false
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
    buildFeatures {
        compose = true
    }
    dependencies {
        debugImplementation(compose.uiTooling)
    }
}
dependencies {
    implementation(libs.androidx.material3.android)
    implementation(libs.androidx.foundation.android)
    implementation(libs.androidx.core.i18n)
}
```

2.4. CHANGELOG.md

Bemærk: Relevant vesions nummer for nuværende projekt er fra og med 0.5.0.

```
# Psyche Assistant Changelog
- - -

> ### Author: Mads Søndergaard
> ### Current version: 0.2.2
> ### License: AGPL
- - -

## Version history:
### 0.2.2:
> General
> - Moved sensitive information to separate properties files.
> - Enabled certification for HTTPS; included in GitHub commit for the sake of example, as they are simple self-signed
certs.
> - Added helper method for fetching properties on specific platforms.
```

```
### 0.2.1:
> General
> - Added the final polish needed for complete CRUD implementation, allowing for deletion/disbanding of
users/groups.
> - A bit more refactoring of views for better UI/UX
> - Added documentation to code where relevant/appropriate.

### 0.2.0:
> General
> - Moved to major change as the application now fulfills all major primary milestones
> - Added visualization of energy consumption on a weekly basis
> - Minor adjustment to view definitions to ensure scrolling was available on pages where it would likely be necessary.

### 0.1.6:
> General
> - Worked on finishing touches for activity integration, i.e. fetching completed activities, deleting and completing them
> - Worked on integrating activity integration into new and existing views for better clarity
>   - Note that the User models "energyExpenditure" attribute is now deprecated, as it is instead extrapolated
>     from the summation of activities for the day/period.
> - Slight refactoring of views so that MainPage now has sections for both Energy Consumption and Survey History
> - And the above is more clearly reflected in the code structure.

### 0.1.5:
> General
> - Fleshed out activity-based views, including dialogs for creation, precursor to list views, etc.
> - Refactored project structure, including the structure belonging to the proof-of-concept version of the project.
> - Corrected some of the API communication in terms of how data is processed front-end before being sent to back-end
>   - Note the single-source-of-truth is kept in Backend, but verification is still carried out in front-end


### 0.1.4:
> General
> - Tested various use cases for logical holes, patched in both front- and back-end.
> - Refactored views so file structure is more readable and easier to maintain/edit
> - Improved clarity on UI/UX in terms of user overview for groups, navigation, and the like.
> - Further implemented Activity model and relation to other models (User/Group)

### 0.1.3:
> General
> - Further polish to UI/UX
> - Implemented logic for group owner management of users (remove user)
> - Implemented logic for joining groups
> - Consolidated Group and User setting pages to a single Account settings page.
> - Moved hard-coded string values to resources.
> - Made various adjustments to layout
> - Added error handling on app start-up, in case server is unavailable. Still allowing the user to take surveys if that is
the case.

### 0.1.2:
> General
> - Refactoring Controller/Repository structure so delegation is now:
>   - Controller: Used to control the action between the user and app
>   - Service: Used by the controller to communicate between the app and server
>   - Repository: Used to act as a contract for what functions are available to the Service.
> - Reconfigured model so front-end now deals with ID-based relations. For the sake of ease of use, backend still
operates with object based relations.
```

> - General refactoring and implementation of code to assist in ongoing debugging and testing.
> - Added additional common-screens to help indicate use flow and state (loading screen, error screen)

> Android
> - Configured logo for the app.

### 0.1.1:
> General
> - Migrated to IntelliJ 2024.2 in order to fix issue with debugging suspend functions
>   - This, however, has caused a minor annoyance with physical device screen mirroring not functioning.
>   - [X] Reported to JetBrains.
> - Added base model description for Group entity.
> - Moved User settings to Settings page.

### 0.1.0:
> General
> - Begun development of user integration, using external database (Spring Boot/H2) for general data storage
> - Minor refactoring, but primarily avoiding legacy proof-of-concept code for the time being.

> Android
> - Begun development of security integration, using MMKV for Android specific secure storage.

### 0.0.5:
> General
> - Fine tuned results page for better clarity in terms of result and user understanding.
> - Begun reconfiguring architecture for better clarity, as well as added code documentation

### 0.0.4:
> General
> - Refactored scoring system so it is more in line with logical use-flow; subtracting and adding scores as one might expect.
> - Added result view and called this from the Survey page when Questions have been answered.
> - Closed logical holes in code so the app no longer crashes if the user tries to go outside the range of the question list.
> - Added relevant symptom and awareness info in English and Danish.

### 0.0.3:
> General
> - Worked on translation functionality so contents adheres to device language
>   - But added the option for survey questions to be translateable, as these require separate translator (since they are stored in JSON, not Resources)

### 0.0.2:
> General
> - Various fine tunement of views and logical structure for ease of use
> - Ensured score is updated based on selected value moving forward and backward
> - Ensured value per question is reflected in total sum of values for all questions.
> - Various considerations in terms of:
>   - Translation; dynamic vs static
>   - User interface; readability and clarity
>   - Database; local vs external, on-premise vs cloud.
> - Added helper function for converting float to decimal, as this would otherwise not be available in Kotlin, only natively in Java (Android)

> Android
> - N/A

> iOS

> - N/A

### 0.0.1:
> General
> - Decided on licensing format; AGPL
> - Added serializable Question object for iterating across JSON content.
> - Populated JSON file with placeholder questions.
> - Setup primitive Android view that can display each Question object
>   - And react according to the question-type
> - Added slider for more granular and less taxing indication of severity during assessment.

> Android
> - Added functionaltiy for reading JSON file from local storage.

> iOS
> - N/A

3. Back-end

3.1. main

3.1.1. java

3.1.1.1. com.example.PsycheAssistantAPI

3.1.1.1.1. Controller

3.1.1.1.1.1. ActivityController

```
package com.example.PsycheAssistantAPI.Controller;

import com.example.PsycheAssistantAPI.Helper.AuthHelper;
import com.example.PsycheAssistantAPI.Model.Activity;
import com.example.PsycheAssistantAPI.Model.Group;
import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Security.JwtUtil;
import com.example.PsycheAssistantAPI.Service.ActivityService;
import com.example.PsycheAssistantAPI.Service.GroupService;
import com.example.PsycheAssistantAPI.Service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

/**
 * Controller for API; exposes back-ends for front- to back-end communication used for manipulating Activity model data.
 */
@RestController
@RequestMapping("/api/activities")
public class ActivityController {

    @Autowired
    private ActivityService activityService;
    @Autowired
    private JwtUtil jwtUtil;
    @Autowired
    private AuthHelper authHelper;


    @GetMapping()
    public List<Activity> getAllActivities() {
        return activityService.findAll();
    }

    @GetMapping("/{id}")
    public Activity read(@PathVariable int id) {
        return activityService.findById(id);
    }

    /**
     * Validates user and creates an activity based on the payload contents received.
     * @param authHeader
     * @param payload
     * @return
     */
    @PostMapping("/create")
    public ResponseEntity<Activity> createActivity(
            @RequestHeader("Authorization") String authHeader,
            @RequestBody Map<String, Object> payload) {

        String deadline = (String) payload.get("deadline");
        String description = (String) payload.get("description");
        int energyCost = Integer.parseInt((String) payload.get("energyCost"));

        // Validate user authorization
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        if (userResponse.getStatusCode() != HttpStatus.OK) {
            return new ResponseEntity<>(null, userResponse.getStatusCode());
        }

        User user = userResponse.getBody();
        assert user != null;

        try {
            Activity newActivity = activityService.createActivity(user, deadline, description, energyCost);
            return new ResponseEntity<>(newActivity, HttpStatus.CREATED);
        } catch (IllegalArgumentException e) {
            return new ResponseEntity<>(null, HttpStatus.BAD_REQUEST);
        }
    }

    /**
     * Validates the user and sets the activity to completed by that user
     * @param authHeader
     * @param id
     * @return
     */
    @PostMapping("/{id}/complete")
    public ResponseEntity<Activity> completeActivity(@RequestHeader("Authorization") String authHeader,
@PathVariable int id) {
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        User user = userResponse.getBody();
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }

        try {
            Activity completedActivity = activityService.completeActivity(user, id);
            return new ResponseEntity<>(completedActivity, HttpStatus.OK);
        } catch (IllegalArgumentException e) {
            return new ResponseEntity<>(null, HttpStatus.BAD_REQUEST);
        }

    }

    /**
     * Deletes an activity from the database
     * @param authHeader
     * @param id
     * @return
     */
    @DeleteMapping("/{id}/delete")
    public ResponseEntity<Boolean> deleteActivity(@RequestHeader("Authorization") String authHeader,
@PathVariable int id) {
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        User user = userResponse.getBody();
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }
        boolean success = activityService.deleteActivity(id);
        return success ? new ResponseEntity<>(HttpStatus.NO_CONTENT) : new
ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    /**
     * Gets a range of activities for a group, based on the specific date they were created.
     * @param groupId
     * @param date
     * @return
     */
    @GetMapping("/group/{groupId}")
    public ResponseEntity<List<Activity>> getActivitiesForGroupWithDeadline(
            @PathVariable int groupId,
            @RequestParam("date") String date) {
        List<Activity> activities = activityService.getActivitiesForGroupWithDeadline(groupId, date);
        return new ResponseEntity<>(activities, HttpStatus.OK);
    }

    /**
     * Gets a range of activities where their deadline (creation date) is within a given period.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    @GetMapping("/group/{groupId}/period")
    public ResponseEntity<List<Activity>> getPeriodActivitiesWithDeadline(
            @PathVariable int groupId,
            @RequestParam("startDate") String startDate,
            @RequestParam("endDate") String endDate) {

        List<Activity> activities = activityService.getByPeriod(groupId, startDate, endDate);
        return new ResponseEntity<>(activities, HttpStatus.OK);
    }

    /**
     * Gets a range of activities where their handledOn date is within a given period.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    @GetMapping("/group/{groupId}/handled/period")
    public ResponseEntity<List<Activity>> getPeriodActivitiesWithHandled(
            @PathVariable int groupId,
            @RequestParam("startDate") String startDate,
            @RequestParam("endDate") String endDate) {

        List<Activity> activities = activityService.getHandledByPeriod(groupId, startDate, endDate);
        return new ResponseEntity<>(activities, HttpStatus.OK);
    }

}
```

3.1.1.1.1.2. GroupController

```
package com.example.PsycheAssistantAPI.Controller;

import com.example.PsycheAssistantAPI.Helper.AuthHelper;
import com.example.PsycheAssistantAPI.Model.Group;
import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Security.JwtUtil;
import com.example.PsycheAssistantAPI.Service.GroupService;
import com.example.PsycheAssistantAPI.Service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

/**
 * Controller for API; exposes back-ends for front- to back-end communication used for manipulating Group model data.
 */
@RestController
```

```
@RequestMapping("/api/groups")
public class GroupController {

    @Autowired
    private GroupService groupService;
    @Autowired
    private UserService userService;
    @Autowired
    private JwtUtil jwtUtil;
    @Autowired
    private AuthHelper authHelper;

    @GetMapping()
    public List<Group> getAllGroups() { return groupService.findAll(); }

    @GetMapping("/{id}")
    public Group read(@PathVariable int id) {
        return groupService.findById(id);
    }

    /**
     * Gets all users belonging to the group.
     * @param id
     * @return
     */
    @GetMapping("/{id}/users")
    public List<User> getUsersInGroup(@PathVariable int id) {
        Group group = groupService.findById(id);
        if (group == null) {
            return null;
        }
        return group.getUsers();
    }

    /**
     * Creates a new group.
     * @param authHeader
     * @return
     */
    @PostMapping("/register")
    public ResponseEntity<Group> createGroup(@RequestHeader("Authorization") String authHeader) {
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        if (userResponse.getStatusCode() != HttpStatus.OK) {
            return new ResponseEntity<>(null, userResponse.getStatusCode());
        }

        User user = userResponse.getBody();
        assert user != null;
        Group newGroup = groupService.createGroup(user.getId());
        return new ResponseEntity<>(newGroup, HttpStatus.CREATED);
    }

    /**
     * Joins the validated user to the group matching the supplied code.
     * @param authHeader
     * @param payload
     * @return
     */
    @PostMapping("/join")
    public ResponseEntity<Group> joinGroup(@RequestHeader("Authorization") String authHeader,
                                           @RequestBody Map<String, String> payload) {
        String code = payload.get("code");
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        User user = userResponse.getBody();
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }

        try {
            Group updatedGroup = groupService.joinGroup(code, user);
            return new ResponseEntity<>(updatedGroup, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null, HttpStatus.BAD_REQUEST);
        }
    }

    /**
     * Removes the identified user (via id), if the kicking user is valid and owner.
     * @param authHeader
     * @param id
     * @param payload
     * @return
     */
    @PostMapping("/{id}/kick")
    public ResponseEntity<Group> kickMember(@RequestHeader("Authorization") String authHeader,
                                            @PathVariable int id,
                                            @RequestBody Map<String, Integer> payload) {
        Integer userIdToKick = payload.get("userId");
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        User user = userResponse.getBody();
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }

        try {
            Group group = groupService.findById(id);
            if (group == null) {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }

            if (group.getOwner().getId() != user.getId()) {
                return new ResponseEntity<>(HttpStatus.FORBIDDEN);
            }

            Group updatedGroup = groupService.kickMember(id, userIdToKick);
            return new ResponseEntity<>(updatedGroup, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
    }

    /**
     * Removes the requesting user from the group.
     * @param authHeader
     * @param id
     * @return
     */
    @PostMapping("/{id}/leave")
    public ResponseEntity<Boolean> leaveGroup(@RequestHeader("Authorization") String authHeader,
                                              @PathVariable int id) {
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        User user = userResponse.getBody();
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }

        try {
            Group group = groupService.findById(id);
            if (group == null) {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }

            groupService.kickMember(id, user.getId());
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        } catch (Exception e) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
    }

    /**
     * Deletes the group from the database
     * @param authHeader
     * @param id
     * @return
     */
    @DeleteMapping("/{id}/disband")
    public ResponseEntity<Boolean> disbandGroup(@RequestHeader("Authorization") String authHeader,
                                                @PathVariable int id) {
        ResponseEntity<User> userResponse = authHelper.validateAndGetUser(authHeader);
        User user = userResponse.getBody();
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }

        try {
            groupService.deleteGroup(id);
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        } catch (Exception e) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
    }
}
```

3.1.1.1.1.3. UserController

```
package com.example.PsycheAssistantAPI.Controller;

import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Security.JwtUtil;
import com.example.PsycheAssistantAPI.Service.ActivityService;
import com.example.PsycheAssistantAPI.Service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
/**
 * Controller for API; exposes back-ends for front- to back-end communication used for manipulating User model data.
 */
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;
    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private ActivityService activityService;

    @GetMapping()
    public List<User> getAllUsers() {
        return userService.findAll();
    }

    @GetMapping("/{id}")
    public User read(@PathVariable int id) {
        return userService.findById(id);
    }

    /**
     * Registers the user in the database.
     * @param email
     * @param password
     * @return
     */
    @PostMapping("/register")
    public ResponseEntity<String> registerUser(@RequestParam String email, @RequestParam String password) {
        try {
            userService.registerUser(email, password);
            return ResponseEntity.status(HttpStatus.CREATED).body("User registered successfully");
        } catch (RuntimeException e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
        }
    }

    /**
     * Fetches the currently logged in users information, based on the token.
     * @param authToken
     * @return
     */
    @GetMapping("/me")
    public ResponseEntity<User> getCurrentUser(@RequestHeader("Authorization") String authToken) {
        try {
            String token = authToken.replace("Bearer ", "");
            String email = jwtUtil.extractEmail(token);

            User user = userService.findByEmail(email);

            if (user != null) {
                return ResponseEntity.ok(user);
            } else {
                return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
            }
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
        }
    }

    /**
     * Logs in the user and supplies their device with an acess and refresh token.
     * @param email
     * @param password
     * @return
     */
    @PostMapping("/login")
    public ResponseEntity<Map<String, String>> loginUser(@RequestParam String email, @RequestParam String
password) {
        User user = userService.findByEmail(email);
        if (user != null && new BCryptPasswordEncoder().matches(password, user.getPassword())) {
            String accessToken = jwtUtil.generateToken(email);
            String refreshToken = jwtUtil.generateRefreshToken(email);

            Map<String, String> tokens = new HashMap<>();
            tokens.put("accessToken", accessToken);
            tokens.put("refreshToken", refreshToken);

            return ResponseEntity.ok(tokens);
        }
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }

    /**
     * Refreshes access token based on the refresh token.
     * @param authHeader
     * @return
     */
    @PostMapping("/refresh-token")
    public ResponseEntity<String> refreshToken(@RequestHeader("Authorization") String authHeader) {
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            String refreshToken = authHeader.substring(7);
            try {
                String email = jwtUtil.extractEmail(refreshToken);
                if (jwtUtil.validateRefreshToken(refreshToken, email)) {
                    String newToken = jwtUtil.generateToken(email);
                    return ResponseEntity.ok(newToken);
                } else {
                    return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid or expired refresh token");
                }
            } catch (Exception e) {
```

```
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid refresh token");
        }
    } else {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Missing or invalid Authorization header");
    }
}

/**
 * Deletes the user from the database
 * @param authHeader
 * @return
 */
@DeleteMapping("/delete")
public ResponseEntity<Boolean> deleteUser(@RequestHeader("Authorization") String authHeader) {
    try {
        String token = authHeader.replace("Bearer ", "");
        String email = jwtUtil.extractEmail(token);

        User user = userService.findByEmail(email);

        if (user == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
        }

        boolean success = userService.deleteUser(user);
        return success ? new ResponseEntity<>(HttpStatus.NO_CONTENT) : new
ResponseEntity<>(HttpStatus.NOT_FOUND);
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}
}
```

3.1.1.1.2. Helper

3.1.1.1.2.1. AuthHelper

```
package com.example.PsycheAssistantAPI.Helper;

import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Service.UserService;
import com.example.PsycheAssistantAPI.Security.JwtUtil;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;

/**
 * Helper class that takes care of interacting with jwtUtil to validate users
 */
@Component
public class AuthHelper {

    private final JwtUtil jwtUtil;
    private final UserService userService;
```

```
public AuthHelper(JwtUtil jwtUtil, UserService userService) {
    this.jwtUtil = jwtUtil;
    this.userService = userService;
}

/**
 * Extracts the token from the authHeader received.
 * @param authHeader
 * @return
 * @throws IllegalArgumentException
 */
public String extractToken(String authHeader) throws IllegalArgumentException {
    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        throw new IllegalArgumentException("Invalid authorization header");
    }
    return authHeader.replace("Bearer ", "");
}

/**
 * Based on the token, fetches the email.
 * @param authHeader
 * @return
 * @throws IllegalArgumentException
 */
public String getEmailFromToken(String authHeader) throws IllegalArgumentException {
    String token = extractToken(authHeader);
    return jwtUtil.extractEmail(token);
}

/**
 * Based on the fetched email, finds the specific user.
 * @param authHeader
 * @return
 * @throws IllegalArgumentException
 */
public User getUserFromToken(String authHeader) throws IllegalArgumentException {
    String email = getEmailFromToken(authHeader);
    User user = userService.findByEmail(email);
    if (user == null) {
        throw new IllegalArgumentException("User not found");
    }
    return user;
}

/**
 * Fetches the user from the received authHeader, and returns it if found.
 * @param authHeader
 * @return
 */
public ResponseEntity<User> validateAndGetUser(String authHeader) {
    try {
        User user = getUserFromToken(authHeader);
        return new ResponseEntity<>(user, HttpStatus.OK);
    } catch (IllegalArgumentException e) {
        return new ResponseEntity<>(null, HttpStatus.UNAUTHORIZED);
    }
}
```

```
}
```

3.1.1.1.3. Model

3.1.1.1.3.1. Activity

```
package com.example.PsycheAssistantAPI.Model;

import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.JsonIdentityReference;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.time.LocalDate;
import java.time.LocalDateTime;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "activities")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
public class Activity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String description;
    private int energyCost = 0;

    /**
     * To avoid recursion when calling from front-end, we only use the ID as reference when de/serializing the models as
JSON objects.
     */
    @ManyToOne
    @JoinColumn(name = "group_id")
    @JsonIdentityReference(alwaysAsId = true)
    @ToString.Exclude
    private Group group = null;

    @ManyToOne
    @JsonIdentityReference(alwaysAsId = true)
    @ToString.Exclude
    private User owner;

    @ManyToOne
    @JsonIdentityReference(alwaysAsId = true)
    @ToString.Exclude
    private User handledBy = null;

    private LocalDate deadline;
    private LocalDate handledOn = null;
    private boolean completed = false;

    public Activity(User owner, String description, int energyCost, LocalDate deadline) {
        this.owner = owner;
        this.description = description;
        this.energyCost = energyCost;
        this.deadline = deadline;

        if (owner.getGroup() != null) {
            this.group = owner.getGroup();
        }
    }
}
```

3.1.1.1.3.2. Group

```
package com.example.PsycheAssistantAPI.Model;

import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.JsonIdentityReference;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.ArrayList;
import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "groups")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
public class Group {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    /**
     * To avoid recursion when calling from front-end, we only use the ID as reference when de/serializing the models as
JSON objects.
     */
    @OneToMany(mappedBy = "group")
    @JsonIdentityReference(alwaysAsId = true)
    @ToString.Exclude
    private List<User> users = new ArrayList<>();

    /**
     * Note that we aren't interested in activities no longer tied to a group to be present in the database
     * hence the Cascade and orphanRemoval.
     */
    @OneToMany(mappedBy = "group", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonIdentityReference(alwaysAsId = true)
    @ToString.Exclude
    private List<Activity> activities = new ArrayList<>();

    @OneToOne
    @JsonIdentityReference(alwaysAsId = true)
    @ToString.Exclude
    private User owner;

    private String code;

    public void addActivity(Activity activity) {
        activities.add(activity);
        activity.setGroup(this);
    }

    public void removeActivity(Activity activity) {
        activities.remove(activity);
        activity.setGroup(null);
    }


    public void addUser(User user) {
        users.add(user);
        user.setGroup(this);
    }

    public void removeUser(User user) {
        users.remove(user);
        user.setGroup(null);
    }
}
```

3.1.1.1.3.3. User

```
package com.example.PsycheAssistantAPI.Model;

import com.fasterxml.jackson.annotation.JsonBackReference;
import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.JsonIdentityReference;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;
import org.springframework.lang.Nullable;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(nullable = false, unique = true)
    private String email;
    private String password;
    private int energyExpenditure = 0;

    /**
     * To avoid recursion when calling from front-end, we only use the ID as reference when de/serializing the models as JSON objects.
     */
    @ManyToOne
    @JoinColumn(name = "group_id")
    @JsonIdentityReference(alwaysAsId = true)
    private Group group = null;
}
```

3.1.1.1.4. Repository

3.1.1.1.4.1. ActivityRepository

```
package com.example.PsycheAssistantAPI.Repository;

import com.example.PsycheAssistantAPI.Model.Activity;
import com.example.PsycheAssistantAPI.Model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDate;
import java.util.List;

/**
 * Note JPA implements many of the most basic CRUD functions for communicating with the database.
 * More advanced queries cna be defined individually, as seen below.
 */
public interface ActivityRepository extends JpaRepository<Activity, Integer> {

    /**
     * Gets activities for the group based on the activitity's deadline
     * @param groupId
     * @param date
     * @return
     */
    @Query("SELECT a FROM Activity a WHERE a.group.id = :groupId AND a.deadline = :date")
    List<Activity> findActivitiesForGroupWithDeadline(
            @Param("groupId") int groupId,
            @Param("date") LocalDate date);

    /**
     * Gets activities for a group where the deadline falls between two specific dates.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    @Query("SELECT a FROM Activity a WHERE a.group.id = :groupId AND a.deadline BETWEEN :endDate
AND :startDate")
    List<Activity> findActivitiesForPeriod(
            @Param("groupId") int groupId,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate);

    /**
     * Gets activities from a group where they were handled (i.e. completed) between two specific dates.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    @Query("SELECT a FROM Activity a WHERE a.group.id = :groupId AND a.handledOn BETWEEN :endDate
AND :startDate")
    List<Activity> findHandledActivitiesForPeriod(
            @Param("groupId") int groupId,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate);

    /**
     * Attempts to null user-specific values on activities handled by a given user, i.e. when that user is deleted.
     * Transactional, so if one function fails, the entire function tree fails.
     * @param user
     */
    @Modifying
    @Transactional
    @Query("UPDATE Activity a SET a.handledBy = NULL, a.handledOn = NULL, a.completed = FALSE WHERE
a.handledBy = :user")
    void updateActivitiesHandledByUser(@Param("user") User user);

    /**
     * Transactional function that attempts to remove the completed state and owner of an activity.
     * @param user
     */
    @Modifying
    @Transactional
    @Query("UPDATE Activity a SET a.owner = NULL, a.completed = FALSE WHERE a.owner = :user")
    void updateActivitiesOwnedByUser(@Param("user") User user);
}
```

3.1.1.1.4.2. GroupRepository

```
package com.example.PsycheAssistantAPI.Repository;

import com.example.PsycheAssistantAPI.Model.Group;
import org.springframework.data.jpa.repository.JpaRepository;

/**
```

```
 * Note that JPA repository implements basic CRUD functionality.
 * It is also able to automatically generate simple queries based on naming conventions
 * employed in the function definition.
 */
public interface GroupRepository extends JpaRepository<Group, Integer> {
    Group findByCode(String code);
}
```

3.1.1.1.4.3. UserRepository

```
package com.example.PsycheAssistantAPI.Repository;

import com.example.PsycheAssistantAPI.Model.User;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 * Note that JPA repository implements basic CRUD functionality.
 * It is also able to automatically generate simple queries based on naming conventions
 * employed in the function definition.
 */
public interface UserRepository extends JpaRepository<User, Integer> {
    User findByEmail(String email);

}
```

3.1.1.1.5. Security

3.1.1.1.5.1. CustomUserDetailsService

```
package com.example.PsycheAssistantAPI.Security;
import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    /**
     * Override function so we can get our custom user details.
     * @param username
     * @return
     * @throws UsernameNotFoundException
     */
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(username);

        return org.springframework.security.core.userdetails.User
                .withUsername(user.getEmail())
                .password(user.getPassword())
                .build();
    }
}
```

3.1.1.1.5.2. JwtFilter

```
package com.example.PsycheAssistantAPI.Security;

import io.jsonwebtoken.ExpiredJwtException;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.GenericFilterBean;

import java.io.IOException;

public class JwtFilter extends GenericFilterBean {

    private final JwtUtil jwtUtil;

    public JwtFilter(JwtUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

    /**
     * Override of standard filter function, in order for us to filter based on email and not username.
     * @param request
     * @param response
     * @param chain
     * @throws IOException
     * @throws ServletException
     */
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;
        String authHeader = httpRequest.getHeader("Authorization");

        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            String token = authHeader.substring(7);

            try {
                String username = jwtUtil.extractEmail(token);
                if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
                    if (jwtUtil.validateToken(token, username)) {
                        jwtUtil.setAuthentication(username, httpRequest);
                    }
                }
            } catch (ExpiredJwtException e) {
                // Handle token expiration and attempt refresh
                String refreshToken = httpRequest.getHeader("Refresh-Token");
                if (refreshToken != null && jwtUtil.validateRefreshToken(refreshToken, e.getClaims().getSubject())) {
                    String newToken = jwtUtil.generateToken(e.getClaims().getSubject());
                    httpResponse.setHeader("Authorization", "Bearer " + newToken);
                    jwtUtil.setAuthentication(e.getClaims().getSubject(), httpRequest);
                } else {
                    httpResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Token expired and refresh token is
invalid or missing.");
                    return;
                }
            } catch (Exception e) {
                httpResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Invalid token.");
                return;
            }
        }

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

3.1.1.1.5.3. JwtUtil

```
package com.example.PsycheAssistantAPI.Security;

import io.jsonwebtoken.JwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import jakarta.servlet.http.HttpServletRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class JwtUtil {

    /**
     * Fetch relevant keys/configurations from application.properties (actually imported from credentials.properties)
     */
    @Value("${jwt.secret}")
    private String SECRET_KEY;

    @Value("${jwt.expiration}")
    private long JWT_EXPIRATION;

    @Value("${jwt.refresh-token-expiration}")
    private long REFRESH_TOKEN_EXPIRATION;

    @Autowired
    private UserDetailsService userDetailsService;

    /**
     * Generates a new token.
     * @param email
     * @return
     */
    public String generateToken(String email) {
        try {
            return Jwts.builder()
                    .setSubject(email)
                    .setIssuedAt(new Date())
                    .setExpiration(new Date(System.currentTimeMillis() + JWT_EXPIRATION))
                    .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                    .compact();
        }
        catch (Exception e) {
            return e.toString();
        }
    }

    /**
     * Extracts the email (subject) of the token.
     * @param token
     * @return
     */
    public String extractEmail(String token) {
        return Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
    }

    /**
     * Validates the token supplied matches the passed email.
     * @param token
     * @param email
     * @return
     */
    public boolean validateToken(String token, String email) {
        String extractedEmail = extractEmail(token);
        return (extractedEmail.equals(email));
    }

    /**
     * Checks if token is expired.
     * @param token
     * @return
     */
    private boolean isTokenExpired(String token) {
        Date expiration = Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody()
                .getExpiration();
        return expiration.before(new Date());
    }

    /**
     * Generates refresh-token.
     * @param email
     * @return
     */
    public String generateRefreshToken(String email) {
        try {
            return Jwts.builder()
                    .setSubject(email)
                    .setIssuedAt(new Date())
                    .setExpiration(new Date(System.currentTimeMillis() + REFRESH_TOKEN_EXPIRATION))
                    .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                    .compact();
        } catch (Exception e) {
            return e.toString();
        }
    }

    /**
     * Validates refresh-token based on passed email.
     * @param refreshToken
     * @param email
     * @return
     */
    public boolean validateRefreshToken(String refreshToken, String email) {
        try {
            String tokenEmail = extractEmail(refreshToken);
            return tokenEmail.equals(email) && !isTokenExpired(refreshToken);
        } catch (JwtException e) {
            return false;
        }
    }

    /**
     * Sets authentication for the username (email).
     * @param username
     * @param request
     */
    public void setAuthentication(String username, HttpServletRequest request) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        UsernamePasswordAuthenticationToken authentication =
                new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication(authentication);
    }
}
```

3.1.1.1.5.4. SecurityConfig

```
package com.example.PsycheAssistantAPI.Security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;

import java.util.Arrays;

/**
 * Configures security, i.e. which endpoints can be accessed without authentication.
 * For development purposes, all are currently permitted.
 */
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private JwtUtil jwtUtil;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeRequests(authorizeRequests ->
                        authorizeRequests
                                .requestMatchers("/**").permitAll()
                                .requestMatchers("/console/h2").permitAll()
                                .anyRequest().authenticated()
                )
                .headers(headers -> headers
                        .frameOptions().sameOrigin())
                .addFilterBefore(jwtFilter(), UsernamePasswordAuthenticationFilter.class);  // Add JWT filter

        return http.build();
    }

    @Bean
    public JwtFilter jwtFilter() {
        return new JwtFilter(jwtUtil);
    }
}
```

3.1.1.1.6. Service

3.1.1.1.6.1. ActivityService

```
package com.example.PsycheAssistantAPI.Service;

import com.example.PsycheAssistantAPI.Model.Activity;
import com.example.PsycheAssistantAPI.Model.Group;
import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Repository.ActivityRepository;
import com.example.PsycheAssistantAPI.Repository.GroupRepository;
import com.example.PsycheAssistantAPI.Repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import java.util.List;

/**
 * Links Controller (end-points) to Repository (database)
 */
@Service
public class ActivityService {

    @Autowired
    private ActivityRepository activityRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private GroupRepository groupRepository;

    public List<Activity> findAll() {
        return activityRepository.findAll();
    }

    public Activity findById(int id) {
        return activityRepository.findById(id).orElse(null);
    }

    /**
     * Creates a new activity in the database, and updates the groups list of activities.
     * @param user
     * @param deadline
     * @param description
     * @param energyCost
     * @return
     */
    public Activity createActivity(User user, String deadline, String description, int energyCost) {
        Group targetGroup = groupRepository.findById(user.getGroup().getId()).orElseThrow(() -> new
IllegalArgumentException("Group not found"));

        LocalDate deadlineDate = LocalDate.parse(deadline, DateTimeFormatter.ISO_LOCAL_DATE);

        Activity activity = new Activity(user, description, energyCost, deadlineDate);
        Activity newActivity = activityRepository.save(activity);

        targetGroup.addActivity(newActivity);
        groupRepository.save(targetGroup);
        return newActivity;
    }

    /**
     * Validates a given activity isn't already completed, before marking it as complete din the database, using the
relevant user to mark who handled it.
     * @param user
     * @param activityId
     * @return
     */
    public Activity completeActivity(User user, int activityId) {
        Activity activity = findById(activityId);
        if (activity.isCompleted()) {
            throw new IllegalStateException("Activity is already completed");
        }

        activity.setCompleted(true);
        activity.setHandledBy(user);
        activity.setHandledOn(LocalDate.now());
        activityRepository.save(activity);
        return activity;

    }

    /**
     * Deletes an activity if it exists.
     * @param activityId
     * @return
     */
    public boolean deleteActivity(int activityId) {
        Activity activity = findById(activityId);
        if (activity != null) {
            activityRepository.delete(activity);
            return true;
        }
        return false;
    }

    /**
     * Fetches activities with deadlines of a specific date.
     * @param groupId
     * @param date
     * @return
     */
    public List<Activity> getActivitiesForGroupWithDeadline(int groupId, String date) {
        LocalDate dateBoundary = LocalDate.parse(date);
        return activityRepository.findActivitiesForGroupWithDeadline(groupId, dateBoundary);
    }

    /**
     * Fetches activities based on their deadline, if it is between two specific dates.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    public List<Activity> getByPeriod(int groupId, String startDate, String endDate) {
        LocalDate dateStart = LocalDate.parse(startDate);
        LocalDate dateEnd = LocalDate.parse(endDate);
        return activityRepository.findActivitiesForPeriod(groupId, dateStart, dateEnd);
    }

    /**
     * Fetches activities based on their handledOn date, if it is between two specific dates.
     * @param groupId
     * @param startDate
     * @param endDate
     * @return
     */
    public List<Activity> getHandledByPeriod(int groupId, String startDate, String endDate) {
        LocalDate dateStart = LocalDate.parse(startDate);
        LocalDate dateEnd = LocalDate.parse(endDate);
        return activityRepository.findHandledActivitiesForPeriod(groupId, dateStart, dateEnd);
    }
}
```

3.1.1.1.6.2. GroupService

```
package com.example.PsycheAssistantAPI.Service;

import com.example.PsycheAssistantAPI.Model.Activity;
import com.example.PsycheAssistantAPI.Model.Group;
import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Repository.ActivityRepository;
import com.example.PsycheAssistantAPI.Repository.GroupRepository;
import com.example.PsycheAssistantAPI.Repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.UUID;

/**
 * Links Controller (end-points) to Repository (database)
 */
@Service
public class GroupService {

    @Autowired
    private GroupRepository groupRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private ActivityRepository activityRepository;

    public List<Group> findAll() { return groupRepository.findAll(); }
    public Group findById(int id) { return groupRepository.findById(id).orElse(null); }
    public Group findByCode(String code) {
        return groupRepository.findByCode(code);
    }
    /**
     * Verifies the user exists, before creating a new group, generating its join-code (code), setting the owner, and saving
it in the database.
     * @param userId
     * @return
     */
    public Group createGroup(int userId) {
        User creator = userRepository.findById(userId).orElse(null);
        if (creator == null) {
            throw new IllegalArgumentException("User not found");
        }

        Group group = new Group();
        group.setCode(generateUniqueCode());
        group.addUser(creator);
        group.setOwner(creator);


        return groupRepository.save(group);
    }

    /**
     * Adds a given user to a specific group, if found, matching the code passed.
     * @param code
     * @param user
     * @return
     */
    public Group joinGroup(String code, User user) {
        Group group = findByCode(code);
        if (group == null) {
            throw new IllegalArgumentException("Group not found");
        }

        if (group.getUsers().contains(user)) {
            throw new IllegalArgumentException("User is already in this group");
        }

        group.addUser(user);
        groupRepository.save(group);

        return group;
    }

    /**
     * Verifies the group and user exists, and that the user is a member of the group, before removing said user from
group (and setting owner to null
     * if the user was the group owner).
     * @param groupId
     * @param userId
     * @return
     */
    public Group kickMember(int groupId, int userId) {
        Group group = groupRepository.findById(groupId).orElseThrow(() -> new RuntimeException("Group not
found"));
        User user = userRepository.findById(userId).orElseThrow(() -> new RuntimeException("User not found"));

        if (!group.getUsers().contains(user)) {
```

```java
            throw new RuntimeException("User is not a member of the group");
        }
        group.removeUser(user);

        if (user == group.getOwner())
        {
            group.setOwner(null);
        }

        groupRepository.save(group);

        return group;
    }

    /**
     * Delets the group after nulling group fields for its activities and users.
     * @param groupId
     */
    @Transactional
    public void deleteGroup(int groupId) {
        Group group = findById(groupId);
        if (group != null) {
            List<User> users = group.getUsers();
            for (User user : users) {
                user.setGroup(null);
                userRepository.save(user);
            }
            List<Activity> activities = group.getActivities();
            for (Activity activity : activities) {
                activity.setGroup(null);
                activityRepository.save(activity);
            }
            groupRepository.delete(group);
        }
    }

    /**
     * Helper function. Generates a random code, only using the first 6 characters of the generated UUID.
     * @return
     */
    private String generateUniqueCode() {
        String uuid = UUID.randomUUID().toString().replace("-", "");
        return uuid.substring(0, 6);
    }
}
```

3.1.1.1.6.3. UserService

```java
package com.example.PsycheAssistantAPI.Service;

import com.example.PsycheAssistantAPI.Model.Activity;
import com.example.PsycheAssistantAPI.Model.Group;
import com.example.PsycheAssistantAPI.Model.User;
import com.example.PsycheAssistantAPI.Repository.ActivityRepository;
import com.example.PsycheAssistantAPI.Repository.GroupRepository;
import com.example.PsycheAssistantAPI.Repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

/**
 * Links Controller (end-points) to Repository (database)
 */
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;
    @Autowired
    private ActivityRepository activityRepository;
    @Autowired
    private GroupRepository groupRepository;

    /**
     * Initialize BCryptPasswordEncoder, used to store user passwords safely hashed. Note it internally applies salt as 
well.
     */
    private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    public User findById(int id) { return userRepository.findById(id).orElse(null); }

    public List<User> findAll() { return userRepository.findAll(); }

    /**
     * Register the user with the supplied email and (hashed) password.
     * @param email
     * @param password
     */
    public void registerUser(String email, String password) {
        if (userRepository.findByEmail(email) != null) {
            throw new RuntimeException("User already exists");
        }
        if (!isValidEmail(email)) {
            throw new RuntimeException("Invalid email format");
        }
        if (!isValidPassword(password)) {
            throw new RuntimeException("Password does not meet complexity requirements");
        }

        User user = new User();
        user.setEmail(email);
        user.setPassword(passwordEncoder.encode(password));
        userRepository.save(user);
    }

    /**
     * Finds the user based on the email.
     * @param email
     * @return
     */
    public User findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    /**
     * Attempts to delete the user after nulling activities they have handled, and removing them from their group.
     * @param user
     * @return
     */
    @Transactional
    public boolean deleteUser(User user) {
        activityRepository.updateActivitiesHandledByUser(user);
        activityRepository.updateActivitiesOwnedByUser(user);
        Group group = user.getGroup();
        group.removeUser(user);
        if (user == group.getOwner()) {
            group.setOwner(null);
        }
        userRepository.delete(user);
        return true;
    }

    /**
     * Verifies the supplied email is valid, i.e. that there is:
     * a local part containing any number of characters a-zA-Z0-9_ and -,
     * separating @
     * domain (and subdomain) part of any number of characters a-zA-Z
     * a separating .
     * a top-level domain between 2 - 4 characters in length
     * @param email
     * @return
     */
    private boolean isValidEmail(String email) {
        return email.matches("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");
    }

    /**
     * Checks the validity of the password, i.e. lenght, contains digit, contains letters.
     * Stricter format validity is handled in front-end.
     * @param password
     * @return
     */
    private boolean isValidPassword(String password) {
        return password.length() >= 8 && password.chars().anyMatch(Character::isDigit) &&
password.chars().anyMatch(Character::isLetter);
    }
}
```

3.1.1.1.7. PsycheAssisantApiApplication

```java
package com.example.PsycheAssistantAPI;

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

```
@SpringBootApplication
public class PsycheAssistantApiApplication {

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

}
```

3.1.2. resources

3.1.2.1. application.properties

```
spring.application.name=PsycheAssistantAPI
# server.port=8080
# Import credentials from external file (exclude from source control!)
spring.config.import=optional:classpath:credentials.properties

# HTTPS Configuration:
server.port=8443
server.ssl.key-store=classpath:psyche.p12
server.ssl.key-store-password=${ssl_pass}
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=psyche

# PostgreSQL Configuration:
spring.datasource.url=jdbc:postgresql://localhost:5432/PsycheAssistantDB
spring.datasource.username=${sql_username}
spring.datasource.password=${sql_password}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
spring.jpa.hibernate.ddl-auto=update

# H2 Configuration:
# spring.datasource.url=jdbc:h2:file:~/data/psycheassistantdb
# spring.datasource.driver-class-name=org.h2.Driver
# spring.datasource.username=TestName
# spring.datasource.password=TestPass
# hibernate.dialect=org.hibernate.dialect.H2Dialect
# spring.h2.console.enabled=true
# spring.h2.console.path=/console/h2
# spring.h2.console.settings.web-allow-others=true
```

3.1.2.2. credentials.properties

```
sql_username=PsycheAssistantUser
sql_password=1553
ssl_pass=Nikk1553
jwt.secret=TestKey
jwt.expiration=36000000
jwt.refresh-token-expiration=604800000
```

3.1.2.3. psyche.p12

Bemærk at denne filen ikke umiddelbart er menneskelæseligt. Den indeholder det self-signed certifikat og nøgle, nødvendig for at kommunikere via HTTPS.

3.2. build.gradle.kts

```
plugins {
	java
	id("org.springframework.boot") version "3.3.2"
	id("io.spring.dependency-management") version "1.1.6"
}

group = "com.example"
version = "0.0.1-SNAPSHOT"

java {
	toolchain {
		languageVersion = JavaLanguageVersion.of(17)
	}
}

configurations {
	compileOnly {
		extendsFrom(configurations.annotationProcessor.get())
	}
}

repositories {
	mavenCentral()
}

dependencies {
	implementation("org.springframework.boot:spring-boot-starter-data-jpa")
	implementation("org.springframework.boot:spring-boot-starter-web")
	compileOnly("org.projectlombok:lombok")
	developmentOnly("org.springframework.boot:spring-boot-devtools")
	// runtimeOnly("com.h2database:h2") // H2 Integration
	runtimeOnly("org.postgresql:postgresql") // PostgreSQL Integration
	annotationProcessor("org.projectlombok:lombok")
	testImplementation("org.springframework.boot:spring-boot-starter-test")
	testRuntimeOnly("org.junit.platform:junit-platform-launcher")
	implementation("org.springframework.boot:spring-boot-starter-security")
	implementation("io.jsonwebtoken:jjwt:0.9.1")
	implementation("javax.xml.bind:jaxb-api:2.3.1")

}
tasks.withType<Test> {
	useJUnitPlatform()
}
```

3.3. CHANGELOG.md

Bemærk at relevant versions nummer for dette projekt er fra og med 0.0.1.

```
# Psyche Assistant API Changelog
- - -

> ### Author: Mads Søndergaard
> ### Current version: 0.1.2
> ### License: AGPL
- - -

## Version history:
### 0.1.2:
> General
> - Moved sensitive information to properties files.
> - Added certification so communication between front- and back-end must use HTTPS, as passwords were sent in
clear-text.
>   - NOTE: Be aware of this during presentation, as the server is hosted locally, and thus sensitive to network changes.

### 0.1.1:
> General
> - Migrated external database to PostgreSQL.

### 0.1.0:
> General
> - Updated version to major version number to reflect CRUD completeness.
> - Added logic for deleting a user, ensuring when doing so that related activities are updated accordingly.
> - Refined logic for deleting (disbanding) groups, as well as transactions to ensure related models are updated aswell.

### 0.0.7:
> General
> - Added further needed back-end functionality to facilitate better retrieval of activites and hence energy expenditure.
>   Both in terms of User object and timeframe.

### 0.0.6:
> General
> - Fleshed out Activity based logic (Controller, Service, Repository), and aligned front-end with API.
> - Added functionality for fetching Activities based on Group
> - Added functionality for creating an Activity.
> - Added functionality for deleting/completing activities, albeit untested at this point.

### 0.0.5:
> General
> - Fine-tuned certain use cases such as removal of users
> - Patched logical hole which caused recursion when fetching users/groups
> - Implemented more logic related to Activities.

### 0.0.4:
> General
> - Polished user manipulation of group content
> - Added helper method for authentication
> - Various smaller adjustments to logic as needed.
```

### 0.0.3:
> General
> - Worked on aligning de/serialization between backend and frontend in a way that makes technical sense.
> - Refactored code to be in line with this approach; backend uses full-model relations, while frontend uses ID-relations.

### 0.0.2:
> General
> - Added foundation for Group model (Entity, Repository, Controller)
> - Various minor tweaks.

### 0.0.1:
> General
> - Initial commit
> - Included fundamental structure for model (Entity, Repository, Controller)
> - Using Spring Boot API and H2 storage
> - Using JWT, begun implementation of security logic for safe access from application based on user-authentication.
> Note:
> - Opted to use JWT rather than, for example, Firebase, to avoid lock-in.