Sprawozdanie Aplikacje Mobilne

Dawid Stasiak 148112
Miłosz Matuszewski 148185

Aplikacja została napisana w języku Kotlin przy wykorzystaniu narzędzi Jetpack Compose.

Źródło danych

Aplikacja posiada wbudowaną bazę danych w której znajdują się wszystkie informacje na temat przepisów dostępnych w aplikacji.

W trakcie jej uruchomienia aplikacja za pomocą komend języka SQL tworzy odpowiednie tabele i uzupełnia je odpowiednimi danymi.

Dane dla jednego z przepisów:

```
RecipeWithIngredientsAndSteps(
    recipe = Recipe(
        id = 1,
        name = "Kotlet schabowy",
        type = RecipeType.MAIN_COURSE,
        calories = 1000,
        image = R.drawable.pork_chop,
    ),
    ingredients = listOf(
        RecipeIngredient(
            id = 1,
            recipeId = 1,
            name = "Schab",
            amount = 1f,
            unit = "kg"
        ),
        RecipeIngredient(
            id = 2,
            recipeId = 1,
            name = "Mąka",
            amount = 0.5f,
            unit = "kg"
        ),
        RecipeIngredient(
            id = 3,
            recipeId = 1,
            name = "Jajko",
```

```
            amount = 1f,
            unit = "szt"
        ),
        RecipeIngredient(
            id = 4,
            recipeId = 1,
            name = "Sól",
            amount = 0.5f,
            unit = "łyżka"
        ),
        RecipeIngredient(
            id = 5,
            recipeId = 1,
            name = "Pieprz",
            amount = 0.5f,
            unit = "łyżka"
        ),
        RecipeIngredient(
            id = 6,
            recipeId = 1,
            name = "Olej",
            amount = 0.5f,
            unit = "łyżka"
        )
    ),
    steps = listOf(
        RecipeStep(
            recipeId = 1,
            order = 1,
            description = "Schab pokroić na kawałki, wymieszać z
mąką, jajkiem, solą i pieprzem.",
            time = 0
        ),
        RecipeStep(
            recipeId = 1,
            order = 2,
            description = "Kawałki schabu panierować w mące, jajku
i mące.",
            time = 0
        ),
        RecipeStep(
            recipeId = 1,
            order = 3,
            description = "Kotlety smażyć na patelni z olejem.",
            time = 10
        )
    )
),
```

Dodanie danych do bazy danych:

```
    fun getDatabase(context: Context): KochbuchDatabase {
        return Instance ?: synchronized(this) {
            Room.databaseBuilder(context,
KochbuchDatabase::class.java, "recipe_database")
                .fallbackToDestructiveMigration()
                .addCallback(object : Callback() {
                    override fun onCreate(db:
SupportSQLiteDatabase) {
                        super.onCreate(db)
                        for (recipeWithSteps in
RecipesSource.recipes) {
                            db.execSQL(
                                "INSERT INTO recipes (id, name,
type, calories, image) VALUES (${recipeWithSteps.recipe.id},
'${recipeWithSteps.recipe.name}',
'${recipeWithSteps.recipe.type}','${recipeWithSteps.recipe.calorie
s}', ${recipeWithSteps.recipe.image})"
                            )
                            for (step in recipeWithSteps.steps) {
                                db.execSQL(
                                    "INSERT INTO recipe_steps
(recipeId, `order`, description, time) VALUES (${step.recipeId},
${step.order}, '${step.description}', ${step.time})"
                                )
                            }
                            for (ingredient in
recipeWithSteps.ingredients) {
                                db.execSQL(
                                    "INSERT INTO recipe_ingredients
(recipeId, name, amount, unit) VALUES (${ingredient.recipeId},
'${ingredient.name}', ${ingredient.amount}, '${ingredient.unit}')"
                                )
                            }
                        }
                    }
                })
                .build()
                .also { Instance = it }
        }
    }
}
```

Aplikacja

Aplikacja składa się z trzech głównych ekranów:
- ekran startowy z opisem aplikacji i animacją
- ekran z listą dostępnych zup
- ekran z listą dostępnych dań głównych

Oraz z ekranu szczegółowego dla każdego przepisu

Ekran główny

Ekran główny składa się z opisu aplikacji, witającego użytkownika oraz animacji nawiązującej do tematu aplikacji.

```
@Composable
fun InfoScreen(layoutType: LayoutType, modifier: Modifier =
Modifier) {
    if (layoutType == LayoutType.PHONE_LANDSCAPE) {
        Row(
            horizontalArrangement = Arrangement.SpaceAround,
            verticalAlignment = Alignment.CenterVertically,
            modifier = modifier.fillMaxSize()
        ) {
            InfoText()
            Box(modifier = Modifier.weight(1f), contentAlignment =
Alignment.Center) {
                InfoAnimation(200.dp)
            }
        }
    } else {
        Column(
            verticalArrangement = Arrangement.SpaceAround,
            horizontalAlignment = Alignment.CenterHorizontally,
            modifier = modifier.fillMaxSize()
        ) {
            InfoText(modifier = Modifier.fillMaxWidth())
            InfoAnimation(350.dp)
        }
    }
}
```

Opis aplikacji:

```
fun InfoText(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier
            .padding(start = 20.dp)
            .padding(bottom = 5.dp)
    ) {
        Text(
            text = stringResource(R.string.welcome),
            fontSize = 30.sp,
            modifier = Modifier.padding(bottom = 5.dp)
        )
        Text(
            text = stringResource(R.string.app_info),
            fontSize = 20.sp,
            modifier = Modifier.padding(bottom = 5.dp)
        )
        Text(
            text = stringResource(R.string.functions),
            fontSize = 20.sp,
            modifier = Modifier.padding(bottom = 5.dp)
        )
        Text(text = stringResource(R.string.soup_function))
        Text(text = stringResource(R.string.main_course_function))
        Text(text = stringResource(R.string.shopping_list))
        Text(text = stringResource(R.string.step_list))
        Text(text = stringResource(R.string.timer))
    }
}
```

Animacja:

Ze względu na obszerność kodu odnośnie animacji, nie został on tu umieszczony. Znajduje się on w folderze z kodem źródłowym projektu w pliku **InfoAnimation.kt**

Na głównym ekranie znajdują się również dwa paski:

- górny pasek z napisem informacje i przyciskiem pozwalającym wybrać losowy przepis
- dolny pasek nawigacyjny z przyciskami do przejścia na ekran główny oraz do listy zup i dań głównych

Górny pasek

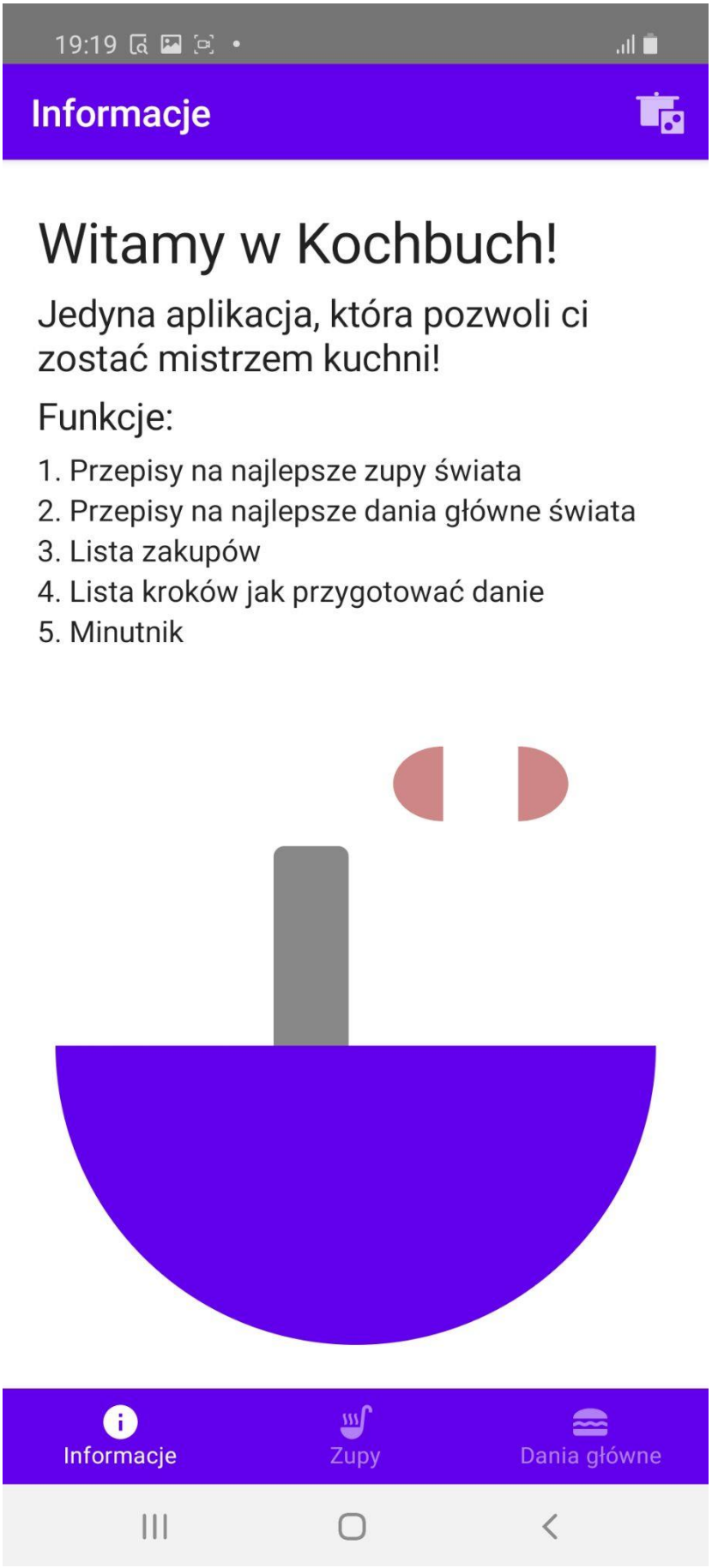
W kodzie znajduje się również przycisk do wyszukiwania, ale jest on aktywny tylko na ekranach z listą przepisów.

```
KochbuchTopAppBar(
    title = currentDestination?.let { stringResource(id =
it.titleRes) } ?: "",
    canNavigateBack = false,
    actions = {
        if (currentDestination != BottomInfoDestination) {
            IconButton(onClick = { isSearching = true }) {
                Icon(
                    Icons.Filled.Search,
                    contentDescription = "Search for recipe with
given ingredient"
                )
            }
        }
        IconButton(onClick = {
navigateToRecipe(allRecipeIds.random()) }) {
            Icon(
                painterResource(id = R.drawable.random_recipe),
                contentDescription =
stringResource(R.string.random_recipe),
                modifier = Modifier.width(32.dp),
            )
        }
    }
)
```

Dolny pasek

```
fun MainBottomBar(navController: NavController, beforeNavigation:
() -> Unit = {}) {
    val items = listOf(
        BottomInfoDestination, BottomSoupDestination,
BottomMainCourseDestination
    )
    BottomNavigation(
        backgroundColor = MaterialTheme.colors.primary,
    ) {
        val navBackStackEntry by
navController.currentBackStackEntryAsState()
        val currentRoute = navBackStackEntry?.destination?.route
        items.forEach { item ->
            BottomNavigationItem(
                icon = { Icon(item.icon, contentDescription =
stringResource(id = item.titleRes)) },
                label = {
```

```
                    Text(
                        text = stringResource(id = item.titleRes),
                    )
                },
                selectedContentColor =
MaterialTheme.colors.onPrimary,
                unselectedContentColor =
MaterialTheme.colors.onPrimary.copy(alpha = 0.5f),
                alwaysShowLabel = true,
                selected = currentRoute == item.route,
                onClick = {
                    beforeNavigation()
                    navController.navigate(item.route) {

navController.graph.startDestinationRoute?.let { screenRoute ->
                            popUpTo(screenRoute) {
                                saveState = true
                            }
                        }
                        launchSingleTop = true
                        restoreState = true
                    }
                }
            )
        }
    }
}
```


19:19
Informacje
Witamy w Kochbuch!
Jedyna aplikacja, która pozwoli ci zostać mistrzem kuchni!
Funkcje:
1. Przepisy na najlepsze zupy świata
2. Przepisy na najlepsze dania główne świata
3. Lista zakupów
4. Lista kroków jak przygotować danie
5. Minutnik
Informacje
Zupy
Dania główne

}

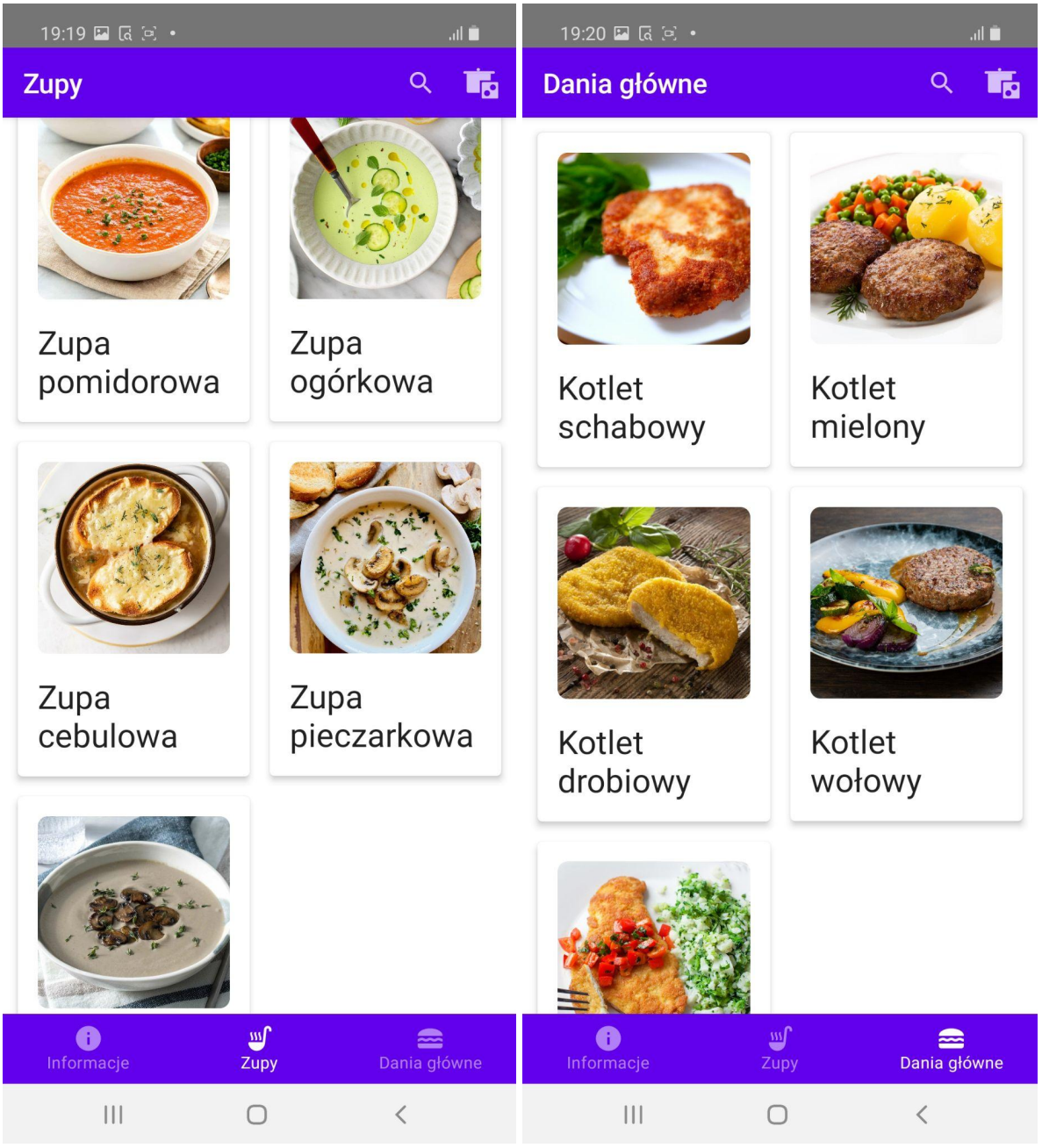
Ekran z listą przepisów

Aplikacja zawiera dwa ekrany z listą przepisów, ze względu na to, że różnią się jedynie rodzajem przepisów w sprawozdaniu zostanie opisany tylko ekran prezentujący listę przepisów zup.

Ekran składa się z listy przepisów ze zdjęciem oraz nazwą przepisu, oraz pasków opisanych we wcześniejszym fragmencie sprawozdania.

```
@Composable
fun RecipeScreen(
    recipes: List<Recipe>,
    navigateToRecipe: (Int) -> Unit,
    modifier: Modifier = Modifier
) {
    LazyVerticalGrid(
        columns = GridCells.Adaptive(150.dp),
        modifier = modifier.fillMaxWidth(),
        contentPadding = PaddingValues(4.dp)
    ) {
        items(recipes, key = { it.id }) { recipe ->
            RecipeCard(recipe = recipe, onRecipeClicked = {
navigateToRecipe(recipe.id) })
        }
    }
}

@Composable
fun RecipeCard(recipe: Recipe, onRecipeClicked: () -> Unit,
modifier: Modifier = Modifier) {
    Card(
        modifier = modifier
            .fillMaxWidth()
            .padding(8.dp)
            .clickable(onClick = onRecipeClicked),
        elevation = 4.dp,
    ) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            Image(
                painter = painterResource(recipe.image),
                contentDescription = null,
                contentScale = ContentScale.Crop,
                modifier = Modifier
                    .fillMaxWidth()
                    .aspectRatio(1f)
                    .clip(shape = RoundedCornerShape(8.dp)),
```

```
                )
                Spacer(modifier = Modifier.height(16.dp))
                Text(
                    text = recipe.name,
                    style = MaterialTheme.typography.h5
                )
            }
        }
    }
}
```



Ekran szczegółowy dla każdego przepisu

Ekran na którym znajdziemy wszystkie dostępne informacje odnośnie przepisu

Ekran składa się z:

- zdjęcia produktu
- ilością porcji wraz z możliwością ich zmiany
- kalorycznością porcji
- listą potrzebnych składników która ulega zmiania w trakcie zmieniania ilości porcji
- listą kroków wraz z minutnikiem do każdego kroku w którym istotny jest czas np.(czas gotowania)

Kod ekranu szczegółowego wyświetlające wszystkie elementy:

```
@Composable
fun RecipeDetailsBody(
    recipe: Recipe,
    ingredients: List<RecipeIngredient>,
    steps: List<RecipeStep>,
    timerStates: Map<RecipeStep, TimerUiState>,
    onTimerEvent: (TimerEvent) -> Unit,
    modifier: Modifier = Modifier
) {
    if (recipe.id == 0) {
        Column(
            modifier = modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = stringResource(R.string.no_recipe_selected),
                modifier = Modifier
                    .padding(horizontal = 16.dp, vertical = 16.dp)
                    .fillMaxWidth(),
                style = MaterialTheme.typography.h5,
                textAlign = TextAlign.Center
            )
        }
        return
    }
    Text(
        text = recipe.name,
        modifier = Modifier
            .padding(horizontal = 16.dp, vertical = 16.dp)
            .fillMaxWidth(),
        style = MaterialTheme.typography.h5
    )
    var numberOfPortions by remember { mutableStateOf(1) }
    PortionCounter(
```

```
            numberOfPortions = numberOfPortions,
            onPlusClicked = {
                if (numberOfPortions > 1)
                    numberOfPortions--
            },
            onMinusClicked = {
                if (numberOfPortions < 10)
                    numberOfPortions++
            }
        )
        Text(
            text = stringResource(R.string.calories, recipe.calories),
            modifier = Modifier
                .padding(horizontal = 16.dp, vertical = 8.dp)
                .fillMaxWidth(),
            style = MaterialTheme.typography.h6
        )
        Text(
            text = stringResource(R.string.ingredients),
            modifier = Modifier
                .padding(horizontal = 16.dp, vertical = 8.dp)
                .fillMaxWidth(),
            style = MaterialTheme.typography.h6
        )
        ingredients.forEach { ingredient ->
            RecipeIngredientItem(ingredient, numberOfPortions =
numberOfPortions)
        }
        Text(
            text = stringResource(R.string.steps),
            modifier = Modifier
                .padding(horizontal = 16.dp, vertical = 8.dp)
                .fillMaxWidth(),
            style = MaterialTheme.typography.h6
        )
        steps.forEach { step ->
            RecipeStepItem(
                step = step,
                timerState = timerStates[step]!!,
                onTimerEvent = onTimerEvent,
            )
        }
}
```

W funkcji RecipeDetailsBody wyświetlana są wszystkie elementy bezpośrednio, lub wywoływane są inne metody:

- PortionCounter do którego przekazywana jest ilość aktualna ilość porcji i funkcje do jej zmiany

```
@Composable
fun PortionCounter(
    numberOfPortions: Int,
    onPlusClicked: () -> Unit,
    onMinusClicked: () -> Unit,
    modifier: Modifier = Modifier
) {
    Column(modifier = modifier) {
        Text(
            text = stringResource(R.string.portions,
numberOfPortions),
            modifier = Modifier
                .padding(horizontal = 16.dp, vertical = 16.dp)
                .fillMaxWidth(),
            style = MaterialTheme.typography.h6
        )
        Row {
            IconButton(
                onClick = onPlusClicked
            ) {
                Icon(Icons.Default.Remove, null)
            }
            IconButton(
                onClick = onMinusClicked
            ) {
                Icon(Icons.Default.Add, null)
            }
        }
    }
}
```

- w pętli dla każdego kroku RecipeStepItem, do którego przekazywany jest aktualny krok, oraz dwa parametry odpowiedzialne za poprawne działanie minutnika

```
    @Composable
fun RecipeStepItem(
    step: RecipeStep,
    timerState: TimerUiState,
    onTimerEvent: (TimerEvent) -> Unit,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 8.dp)
    ) {
        Text(
```

```
                text = stringResource(R.string.step_number,
step.order),
                style = MaterialTheme.typography.subtitle2
            )
            Text(
                text = step.description,
                style = MaterialTheme.typography.body1,
                modifier = Modifier.padding(top = 4.dp)
            )
            RecipeStepTime(
                time = step.time * 60,
                modifier = Modifier.padding(top = 4.dp)
            )
            RecipeTimer(
                step = step,
                timerState = timerState,
                onTimerEvent = onTimerEvent,
            )
        }
    }
```

W tym fragmenie wyświetlane są informację o kroku czyli nazwa i numer kroku oraz wywoływane są dwie metody:

- RecipeStepTimer która wyświetla domyślny czas dla tego kroku jeżeli nie wynosi 0

```
@Composable
fun RecipeStepTime(time: Int, modifier: Modifier =
Modifier) {
    if(time != 0) {
        Text(
            text = stringResource(R.string.step_time,
formatTime(time = time)),
            style = MaterialTheme.typography.caption,
            modifier = modifier
        )
    }
}

@Composable
fun formatTime(time: Int): String {
    val minutes = time / 60
    val seconds = time % 60
    return stringResource(R.string.time_format, minutes,
seconds)
}
```

- RecipeTimer, która odpowiedzialna jest za wyświetlanie timera oraz przycisków do obsługi timera

Składa się ono z:

- pola wyświetlającego minutnik gdy jest uruchomiony

```
Column(verticalArrangement = Arrangement.Center,
modifier = modifier) {
    Text(
        text = formatTime(timerState.minutes * 60 +
timerState.seconds),
        style = MaterialTheme.typography.body1,
        fontSize = 25.sp,
        modifier = Modifier
            .padding(top = 20.dp, bottom = 10.dp)
            .height(30.dp)
            .width(180.dp),
        textAlign = TextAlign.Center
    )
}
```

- pól do edycji wartości gdy minutnik jest zatrzymany

```
Row(
    horizontalArrangement = Arrangement.Center,
    verticalAlignment = Alignment.CenterVertically,
    modifier = modifier
) {
    TextField(
        value = minutesInput,
        onValueChange = { newValue ->
            onTimerEvent(
                TimerEvent.TimeChanged(
                    step,
                    newValue.toIntOrNull() ?: 0,
                    timerState.seconds
                )
            )
            minutesInput = newValue
        },
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Number,
            imeAction = ImeAction.Done
        ),
        modifier = Modifier
            .width(80.dp)
            .height(60.dp),
        textStyle = LocalTextStyle.current.copy(
            textAlign = TextAlign.Center,
```

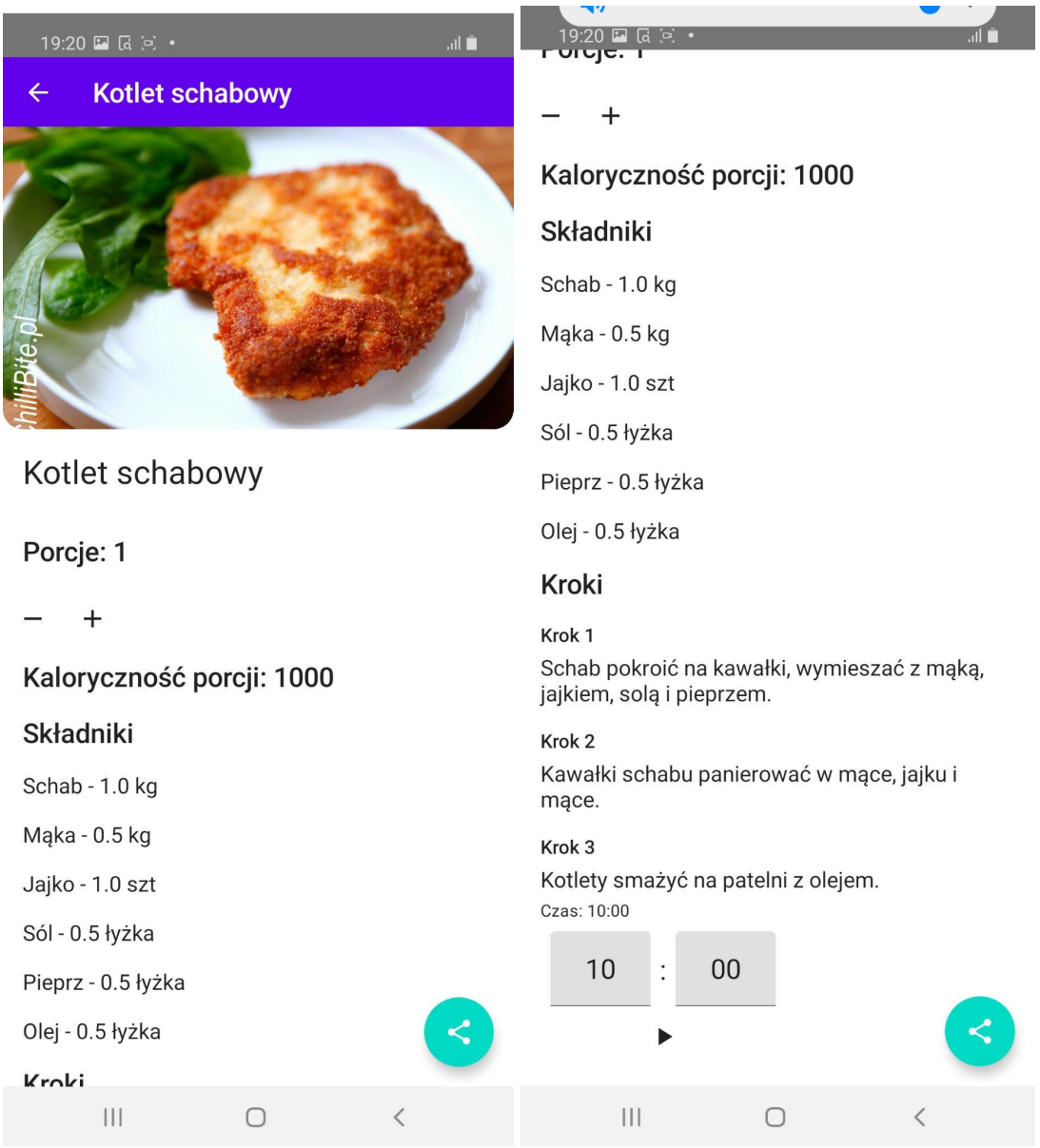
```
                fontSize = 20.sp
            ),
            visualTransformation =
        TimeVisualTransformation()
        )
        Text(
            text = ":",
            fontSize = 20.sp,
            modifier = Modifier.width(20.dp),
            textAlign = TextAlign.Center
        )
        TextField(
            value = secondsInput,
            onValueChange = { newValue ->
                onTimerEvent(
                    TimerEvent.TimeChanged(
                        step,
                        timerState.minutes,
                        newValue.toIntOrNull() ?: 0
                    )
                )
                secondsInput = newValue
            },
            keyboardOptions = KeyboardOptions(
                keyboardType = KeyboardType.Number,
                imeAction = ImeAction.Done
            ),
            modifier = Modifier
                .width(80.dp)
                .height(60.dp),
            textStyle = LocalTextStyle.current.copy(
                textAlign = TextAlign.Center,
                fontSize = 20.sp
            ),
            visualTransformation =
        TimeVisualTransformation()
        )
    }
```

- oraz przycisków do obsługi minutnika

  Gdy minutnik jest zatrzymany wyświetlany jest przycisk do startu odliczania, gdy jest w trakcie przycisk do zatrzymania, a gdy minutnik skończy odliczać czas przycisk do wyłączenia sygnału dźwiękowego i resetu minutnika.
  Działanie przycisków zostały opisane w dalszej części

```
Row(modifier = modifier) {
        if (!timerState.isRunning) {
```

```
                IconButton(onClick = {
            onTimerEvent(TimerEvent.StartClicked(step))
            }) {
                    Icon(Icons.Filled.PlayArrow,
            contentDescription = "Start Timer")
                }
            } else if (timerState.isRunning &&
            (timerState.minutes != 0 ||
            timerState.seconds != 0)) {
                IconButton(onClick = {

            onTimerEvent(TimerEvent.PauseClicked(step))
                    minutesInput =
            timerState.minutes.toString()
                    secondsInput =
            timerState.seconds.toString()
                }) {
                    Icon(Icons.Filled.Pause,
            contentDescription = "Pause Timer")
                }
            } else {
                IconButton(onClick = {

            onTimerEvent(TimerEvent.StopClicked(step))
                    minutesInput =
            step.time.toString()
                    secondsInput = ""
                }) {
                    Icon(Icons.Filled.Stop,
            contentDescription = "Stop Timer")
                }
        }
    }
```



Minutnik

Minutnik oprócz elementów do jego wyświetlania, które zostały opisane wcześniej korzysta z:

- TimerUiState - obiekt tworzony dla każdego kroku korzystającego z minutnika, przechowywane są w nim obecne wartości minutnika

```
data class TimerUiState(val minutes: Int, val seconds: Int,
val isRunning: Boolean = false, val job: Job? = null)
```

- w zależności od urządzenia metod z RecipeDetailsViewModel lub TabletMainViewModel, które są używane za pomocą klas z interfejsu TimerEvent. Klasy te są wywoływane poprzez naciśnięcie przycisku, lub w przypadku TimeChanged przy zmianie wartości w polu

Interfejs:

```
sealed interface TimerEvent {
    data class StartClicked(val step: RecipeStep) : TimerEvent
    data class PauseClicked(val step: RecipeStep) : TimerEvent
    data class StopClicked(val step: RecipeStep) : TimerEvent
    data class TimeChanged(val step: RecipeStep, val minutes:
Int, val seconds: Int) : TimerEvent
}
```

Tworzenie instancji interfejsu:

```
fun onTimerEvent(event: TimerEvent) {
    when (event) {
        is TimerEvent.StartClicked -> startTimer(event.step)
        is TimerEvent.PauseClicked -> pauseTimer(event.step)
        is TimerEvent.StopClicked -> stopTimer(event.step)
        is TimerEvent.TimeChanged -> setTimerValue(event.step,
event.minutes, event.seconds)
    }
}
```

Implementacja metody startTimer:

```
private fun startTimer(step: RecipeStep) {
    val timerState = timerStates[step]!!
    if (timerState.isRunning) {
        return
    }
    val job = viewModelScope.launch {
        while (timerStates[step]!!.seconds > 0 ||
timerStates[step]!!.minutes > 0) {
            delay(1000)
            timerStates[step] = if
(timerStates[step]!!.seconds > 0) {
                timerStates[step]!!.copy(seconds =
timerStates[step]!!.seconds - 1)
            } else {
                timerStates[step]!!.copy(
                    minutes = timerStates[step]!!.minutes - 1,
                    seconds = 59
                )
            }
        }
        isSoundPlaying = true
    }
    timerStates[step] = timerState.copy(job = job, isRunning =
true)
}
```

Metoda startTimer tworzy nowego joba, który jest nowym procesem.

odpowiedzialnym za działanie minutnika.

Implementacja metody pauseTimer:

```
private fun pauseTimer(step: RecipeStep) {
    val timerState = timerStates[step]!!
    if (!timerState.isRunning) {
        return
    }
    timerState.job?.cancel()
    timerStates[step] = timerState.copy(isRunning = false)
}

private fun stopTimer(step: RecipeStep) {
    val timerState = timerStates[step]!!
    if (!timerState.isRunning) {
        return
    }
    timerState.job?.cancel()
    timerStates[step] = timerState.copy(minutes = step.time,
seconds = 0, isRunning = false)
    if (timerStates.values.none { it.isRunning }) {
        isSoundPlaying = false
    }
}
```

Implementacja metody stopTimer:

```
private fun stopTimer(step: RecipeStep) {
    val timerState = timerStates[step]!!
    if (!timerState.isRunning) {
        return
    }
    timerState.job?.cancel()
    timerStates[step] = timerState.copy(minutes = step.time,
seconds = 0, isRunning = false)
    if (timerStates.values.none { it.isRunning }) {
        isSoundPlaying = false
    }
}
```

Implementacja metody setTimerValue:

```
private fun setTimerValue(step: RecipeStep, minutes: Int,
seconds: Int) {
    timerStates[step] = timerStates[step]!!.copy(minutes =
minutes, seconds = seconds)
}
```

Metody startTimer i stopTimer ustawiają rownież flage isSoundPlaying, która jest odpowiedzialna za włączanie i wyłączanie sygnału zakończenia odliczania.

Dźwięk po zakończeniu odliczania

```
LaunchedEffect(isSoundPlaying) {
    if (isSoundPlaying) {
        launch {
            volume.animateTo(
                1f, animationSpec = tween(15000, easing =
LinearEasing)
            )
        }
    }
}
mediaPlayer.setVolume(volume.value, volume.value)
if (isSoundPlaying) {
    if (!mediaPlayer.isPlaying) {
        mediaPlayer.isLooping = true
        mediaPlayer.seekTo(0)
        mediaPlayer.start()
    }
} else {
    if (mediaPlayer.isPlaying) {
        mediaPlayer.pause()
    }
}
```

Przycisk do udostępniania

Na każdym ekranie pokazującym konkretny przepis pojawia się przycisk służacy do udostępnienia przepisu.

Najpierw przygotowywane są dane do udostępnienia dla wybranego przepisu oraza Intent do udostępnienia danych.

```
val shareDataCreator = ShareDataCreator()
    val ingredientsToShare =
shareDataCreator.dataCreator(recipeDetailsUiState)
//        recipeDetailsUiState.ingredients.joinToString(separator
= "\n") { it.name }
    val sendIntent: Intent = Intent().apply {
        action = Intent.ACTION_SEND
        putExtra(Intent.EXTRA_TEXT, ingredientsToShare)
        type = "text/plain"
```

```
    }
    val shareIntent = Intent.createChooser(sendIntent, null)
```

Dane są tworzone za pomocą metody dataCreator w klasie shareDataCreator. Przekazywany do niej jest obiekt zawierający wszyskie informacje o przepisie.

```
fun dataCreator(recipeDetailsUiState: RecipeDetailsUiState):
String {
    val ingredients =  recipeDetailsUiState.ingredients.map { "\n"
+ it.name + " " + it.amount + " " + it.unit }
    val steps = recipeDetailsUiState.steps
    var stepsString = ""
    for(step in steps) {
        stepsString += if(step.time != 0) {
            "\n" + step.description + " Czas:" + step.time + " min"
        } else {
            "\n" + step.description
        }
    }
    val result = ingredients.joinToString(separator = "") + "\n" +
stepsString
    return result.trim()
}
```

Przykładowe dane udostępniane przez aplikację:

Cebula 1.0 kg
Mąka 0.5 kg
Bulion 1.0 l
Olej 0.5 łyżka

Cebulę pokroić w kostkę, podsmażyć na patelni z olejem. Czas:5 min
Mąkę wymieszać z bulionem, dodać do zupy, wymieszać.
Zupę zagotować. Czas:15 min

Sposób wyświetlania i działanie przycisku:

```
floatingActionButton = {
    FloatingActionButton(
        onClick = { context.startActivity(shareIntent) },
        modifier = Modifier.navigationBarsPadding(),
        contentColor = MaterialTheme.colors.primary
    ) {
        Icon(
            imageVector = Icons.Default.Share,
            contentDescription =
stringResource(R.string.recipe_details_share),
            tint = MaterialTheme.colors.onPrimary
```

```
        )
    }
}
```

Przycisk do wyświetlenia losowego przepisu.

Przycisk do losowego przepisu, jest umieszczony na górnym pasku zaprezentowanym wcześniej.

```
IconButton(onClick = { navigateToRecipe(allRecipeIds.random()) }) {
    Icon(
        painterResource(id = R.drawable.random_recipe),
        contentDescription = stringResource(R.string.random_recipe),
        modifier = Modifier.width(32.dp),
    )
}
```

Po naciśnięciu przycisku zostanie wylosowane id przepisu, a następnie aplikacja wyświetla wylosowany przepis na tej samej zasadzie jak przy naciśnięciu przepisu na liście.

Wyszukiwarka przepisów

Wyszukiwarka działa na zasadzie wyszukiwania po składnikach danego przepisu, np. dla frazy “Cebula” wyszuka wszystkie przepisy które zawierają składnik o nazwie cebula.

Przycisk do uruchomienia wyszukiwania:

```
IconButton(onClick = { isSearching = true }) {
    Icon(
        Icons.Filled.Search,
        contentDescription = "Search for recipe with given
ingredient"
    )
}
```

Wyszukiwanie przepisu:

```
var isSearching by remember { mutableStateOf(false) }
var searchValue by remember { mutableStateOf("") }

val filteredMainCourses = mainCourses.filter {
recipeWithIngredients ->
    recipeWithIngredients.ingredients.any {
        it.name.lowercase().contains(
            searchValue.lowercase()
        )
    }
}
val filteredSoups = soups.filter { recipeWithIngredients ->
```

```
    recipeWithIngredients.ingredients.any {
        it.name.lowercase().contains(
            searchValue.lowercase()
        )
    }
}
```

Wersja aplikacji na tablet

Na końcu sprawozdania znajdują się dwa zrzuty ekranu prezentujące aplikacje na tablecie. Decyzja o wersji wyświetlanej aplikacji jest podejmowana na podstawie rozdzielczości wyświetlacza.

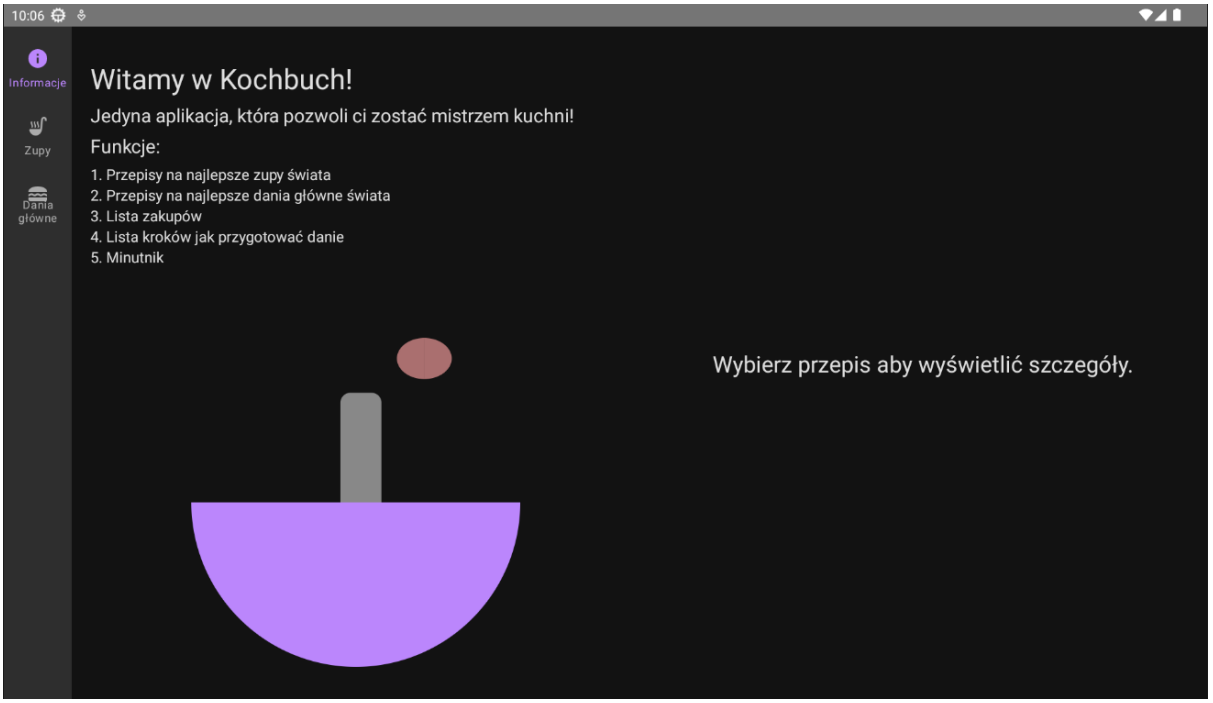
```
fun KochbuchApp(
    widthSizeClass: WindowWidthSizeClass,
    heightSizeClass: WindowHeightSizeClass,
    navController: NavHostController = rememberNavController()
) {
    val isCompactMedium =
        widthSizeClass == WindowWidthSizeClass.Compact &&
heightSizeClass >= WindowHeightSizeClass.Medium
    val isMediumCompact =
        widthSizeClass >= WindowWidthSizeClass.Medium &&
heightSizeClass == WindowHeightSizeClass.Compact

    val layoutType =
        if (isCompactMedium)
            LayoutType.PHONE
        else if (isMediumCompact)
            LayoutType.PHONE_LANDSCAPE
        else
            LayoutType.TABLET
    KochbuchNavHost(
        layoutType = layoutType,
        navController = navController
    )
}
```

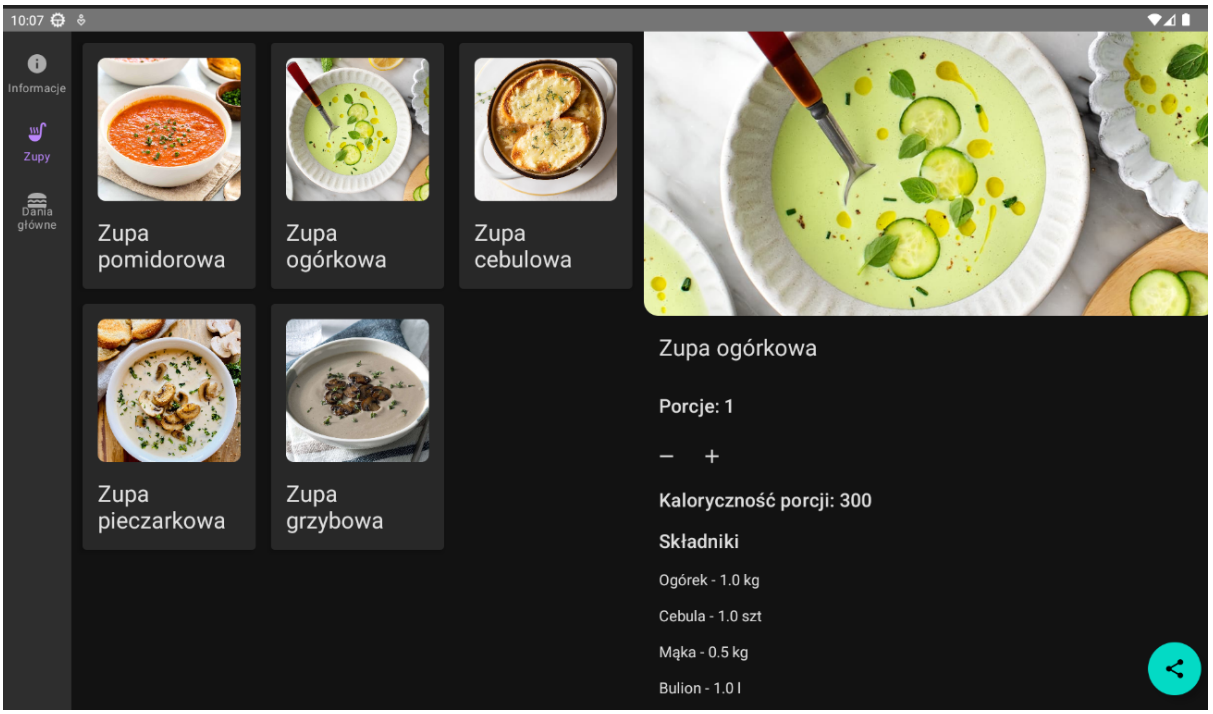
Następnie na podstawie dokonanego wyboru wyświetlana jest odpowiednia wersja aplikacji.

```
fun KochbuchNavHost(
    layoutType: LayoutType,
    navController: NavHostController,
    modifier: Modifier = Modifier,
) {
    NavHost(
        navController = navController,
        startDestination = MainDestination.route,
        modifier = modifier
```

```
    ) {
        composable(route = MainDestination.route) {
            if (layoutType == LayoutType.TABLET) {
                TabletMainScreen(navController =
rememberNavController(), layoutType = layoutType)
            } else {
                MainScreen(navController = rememberNavController(),
navigateToRecipe = {
                    navController.navigate(
                        "${RecipeDetailsDestination.route}/${it}"
                    )
                }, layoutType = layoutType)
            }
        }
        composable(
            route = RecipeDetailsDestination.routeWithArgs,
            arguments = listOf(
                navArgument(RecipeDetailsDestination.recipeId) {
                    type = NavType.IntType
                }
            )
        ) {
            if (layoutType == LayoutType.TABLET) {
                navController.popBackStack(MainDestination.route,
false)
            } else {
                RecipeDetailsScreen(navigateUp = {
navController.popBackStack() })
            }
        }
    }
}
```

W skład kodu źródłowego wchodzą również inne pliki niezbędne do działania aplikacji, które nie zostały opisane w sprawozdaniu. Są to między innymi modele przepisów i kroków a także inne pliki które łączą w całość komponenty aplikacji i zapewniają jej poprawne działanie.


10:06
Informacje
Zupy
Dania główne
Witamy w Kochbuch!
Jedyna aplikacja, która pozwoli ci zostać mistrzem kuchni!
Funkcje:
1. Przepisy na najlepsze zupy świata
2. Przepisy na najlepsze dania główne świata
3. Lista zakupów
4. Lista kroków jak przygotować danie
5. Minutnik
Wybierz przepis aby wyświetlić szczegóły.


10:07
Informacje
Zupy
Dania główne
Zupa pomidorowa
Zupa ogórkowa
Zupa cebulowa
Zupa pieczarkowa
Zupa grzybowa
Zupa ogórkowa
Porcje: 1
– +
Kaloryczność porcji: 300
Składniki
Ogórek - 1.0 kg
Cebula - 1.0 szt
Mąka - 0.5 kg
Bulion - 1.0 l