LAMPIRAN

```
1. class LoginScreen : Screen {
   @Composable
   override fun Content() {
      val viewModel: LoginViewModel = koinViewModel()
      val navigator = LocalNavigator.currentOrThrow
      val context = LocalContext.current
      val lifecycleOwner = LocalLifecycleOwner.current
      val keyboardController = LocalSoftwareKeyboardController.current
      val pref = koinInject<SharedPrefUtils>()
      var email by remember { mutableStateOf("") }
      var password by remember { mutableStateOf("") }
      var accountType by remember { mutableStateOf("") }


      fun emailValidator(email: String): Boolean {
         return android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()
      }
      Column(
         modifier = Modifier
            .fillMaxSize()

.windowInsetsPadding(WindowInsets.safeDrawing)

.verticalScroll(rememberScrollState())
            .pointerInput(Unit) {

detectTapGestures(onTap = {

keyboardController?.hide()
               })
            },
         horizontalAlignment = Alignment.CenterHorizontally,
         verticalArrangement = Arrangement.Center,
      ) {
         Column(
            modifier = Modifier.padding(top = 12.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
         ) {
            Image(
               painterResource(id = R.drawable.illustration_login),
               modifier = Modifier.size(280.dp),
```

```
            contentDescription = "Illustration Login",
        )
        Text(
            text = "Login",
            style = TextStyle(
                fontSize = 24.sp,
                fontWeight = FontWeight.Bold,
                fontFamily = fontFamily,
            ),
        )
        Spacer(modifier = Modifier.size(8.dp))
        Text(
            modifier = Modifier.padding(horizontal = 16.dp),
            text = "Masukan Email dan Password anda yang sudah di daftarkan sebelumnya",
            textAlign = TextAlign.Center,
            softWrap = true,
            style = TextStyle(
                fontSize = 16.sp,
                fontFamily = fontFamily,
                color = Color.Gray,
            ),
        )
    }
    Column(
        modifier = Modifier.padding(top = 32.dp, start = 16.dp, end = 16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        DefaultTextField(label = "Email", value = email) { email = it }
        Spacer(modifier = Modifier.size(4.dp))

        PasswordTextField(label = "Password", value = password) { password = it }
        Spacer(modifier = Modifier.size(4.dp))

        DropDownTextField(label = "Tipe Akun", value = accountType) { accountType = it }
        Spacer(modifier = Modifier.size(8.dp))

        FilledTonalButton(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 8.dp),
            onClick = {
```

```
                // check if email and password is not empty
                if (email.isNotEmpty() && password.isNotEmpty() && accountType.isNotEmpty()) {
                    if (!emailValidator(email)) {
                        Toast.makeText(
                            context,
                            context.getString(R.string.pastikan_penulisan_email_benar),
                            Toast.LENGTH_SHORT
                        ).show()
                    } else {
                        viewModel.loginUser(email, password, accountType)
                            .observe(lifecycleOwner) { result ->
                                when (result) {
                                    is Response.Loading -> {
                                        // show loading
                                        Toast.makeText(
                                            context,
                                            "Loading...",
                                            Toast.LENGTH_SHORT
                                        ).show()
                                    }
                                    is Response.Success -> {
                                        // show success message
                                        Toast.makeText(
                                            context,
                                            "Login berhasil!",
                                            Toast.LENGTH_SHORT
                                        ).show()
                                        // move to home screen
                                        Log.d("Login", "Login berhasil!")
                                        pref.saveString(Const.EMAIL, email)
                                        pref.saveString(Const.PASSWORD, password)
                                        pref.saveString(Const.ACCOUNT_TYPE, accountType)
```

```
navigator.push(MainScreen())
                                }


                                is
Response.Failure -> {
                                    // show
error message

Toast.makeText(

context,

result.msg,

Toast.LENGTH_SHORT

).show()

Log.e("Login", "msg :
${result.msg}")
                                }


                                is
Response.Empty -> {}
                            }
                        }
                    }
                } else {
                    // show error
message
                    Toast.makeText(
                        context,
                        "Pastikan semua
kolom terisi!",

Toast.LENGTH_SHORT
                    ).show()
                }
            },
            colors =
ButtonDefaults.buttonColors(
                containerColor =
Color(0xFFFF8700),
            )
        ) {
            Text(
                "Masuk",
                style = TextStyle(
                    fontFamily =
fontFamily,
                    color =
Color.White,
                    fontSize = 18.sp,
                    fontWeight =
FontWeight.Bold,
                )
            )
        }


        Spacer(modifier =
Modifier.size(4.dp))
```

```
            Row(
                modifier = Modifier.padding(bottom = 46.dp),
                horizontalArrangement = Arrangement.Center,
                verticalAlignment = Alignment.CenterVertically,
            ) {
                Text(
                    "Belum punya akun?",
                    style = TextStyle(
                        fontFamily = fontFamily,
                        color = Color.Gray,
                        fontSize = 16.sp,
                    )
                )
                Spacer(modifier = Modifier.size(4.dp))
                Text(
                    "Daftar",
                    modifier = Modifier.clickable(
                        onClick = {

navigator.push(RegisterScreen())
                        },
                    ),
                    style = TextStyle(
                        fontFamily = fontFamily,
                        color = Color(0xFFFF8700),
                        fontSize = 16.sp,
                    )
                )
            }
        }
    }
}
```

2.
```
class LoginViewModel(private val mUseCase: IAnnaClinicUseCase) : ViewModel() {

    fun loginUser(
        email: String,
        password: String,
        accountType: String
    ) = mUseCase.loginUser(email, password, accountType).asLiveData()
}
```

3.
```
class RegisterScreen : Screen {

@OptIn(ExperimentalMaterial3Api::class)
    @Composable
    override fun Content() {
```

```
    val viewModel: RegisterViewModel = koinViewModel()
    val navigator = LocalNavigator.currentOrThrow
    val context = LocalContext.current
    val lifecycleOwner = LocalLifecycleOwner.current
    val keyboardController = LocalSoftwareKeyboardController.current
    var name by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var confirmPassword by remember { mutableStateOf("") }
    var accountType by remember { mutableStateOf("") }

    fun emailValidator(email: String): Boolean {
        return android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .windowInsetsPadding(WindowInsets.safeDrawing)
            .verticalScroll(rememberScrollState())
            .pointerInput(Unit) {
                detectTapGestures(onTap = {
                    keyboardController?.hide()
                })
            },
        verticalArrangement = Arrangement.SpaceAround,
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Column(
            modifier = Modifier.padding(horizontal = 16.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Image(
                painterResource(id = R.drawable.illustration_signup),
                // change size of image
                modifier = Modifier.size(280.dp),
                contentDescription = "Illustration Signup"
```

```
        )
        Text(
            text = "Daftar Akun",
            style = TextStyle(
                fontSize = 24.sp,
                fontWeight = FontWeight.Bold,
                fontFamily = fontFamily,
            ),
        )
        Spacer(modifier = Modifier.size(16.dp))

        DefaultTextField(label = "Nama", value = name) { name = it }
        Spacer(modifier = Modifier.size(8.dp))

        DefaultTextField(label = "Email", value = email) { email = it }
        Spacer(modifier = Modifier.size(8.dp))

        PasswordTextField(label = "Password", value = password) { password = it }
        Spacer(modifier = Modifier.size(8.dp))

        PasswordTextField(
            label = "Konfirmasi Password",
            value = confirmPassword
        ) {
            confirmPassword = it
        }
        Spacer(modifier = Modifier.size(8.dp))

        FilledTonalButton(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 8.dp),
            onClick = {
                // configure form check
                if (name.isEmpty() || email.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()) {
                    // show error message

                    Toast.makeText(context, "Data tidak boleh kosong", Toast.LENGTH_SHORT)
                        .show()
                } else {
                    if (!emailValidator(email)) {
                        Toast.makeText(
```

```
                    context,
context.getString(R.string.pastikan_penulisan_email_benar),
Toast.LENGTH_SHORT
                )
                    .show()
return@FilledTonalButton
            }

viewModel.registerUser(
                name,
                email,
                password,
confirmPassword,
).observe(lifecycleOwner) {
                when (it) {
                    is Response.Loading -> {
Toast.makeText(
context,
"Loading...",
Toast.LENGTH_SHORT
                        ).show()
                    }
                    is Response.Success -> {
Toast.makeText(
context,
"Berhasil mendaftar",
Toast.LENGTH_SHORT
                        ).show()
navigator.push(LoginScreen())
                    }
                    is Response.Failure -> {
Toast.makeText(
context,
"${it.msg}",
Toast.LENGTH_SHORT
                        ).show()
                    }
                    is Response.Empty -> {
```

```
Toast.makeText(
context,
"${it.msg}",
Toast.LENGTH_SHORT
                    ).show()
                }
            }
        }
    },
    colors = ButtonDefaults.buttonColors(
        containerColor = Color(0xFFFF8700),
    )
) {
    Text(
        "Daftar",
        style = TextStyle(
            fontFamily = fontFamily,
            color = Color.White,
            fontSize = 18.sp,
            fontWeight = FontWeight.Bold,
        )
    )
}

Spacer(modifier = Modifier.size(4.dp))

Row(
    modifier = Modifier.padding(bottom = 24.dp),
    horizontalArrangement = Arrangement.Center,
    verticalAlignment = Alignment.CenterVertically,
) {
    Text(
        "Sudah punya akun?",
        style = TextStyle(
            fontFamily = fontFamily,
            color = Color.Gray,
            fontSize = 16.sp,
        )
    )
    Spacer(modifier = Modifier.size(4.dp))
    Text(
        "Masuk",
        modifier = Modifier.clickable(
            onClick = {

navigator.push(LoginScreen())
```

```
                    },
                ),
                style = TextStyle(
                    fontFamily = fontFamily,
                    color = Color(0xFFFF8700),
                    fontSize = 16.sp,
                )
            )
        }
    }
}
```

4. class RegisterViewModel (private val mUseCase: IAnnaClinicUseCase) : ViewModel() {

```
    fun registerUser(
        name: String,
        email: String,
        password: String,
        confirmPassword: String,
    ) = mUseCase.registerUser(name, email, password, confirmPassword).asLiveData()
}
```

5. class UploadBannerScreen : Screen {

```
    @Composable
    override fun Content() {
        val navigator = LocalNavigator.currentOrThrow
        Scaffold(
            topBar = { CenterTopBar(navigator = navigator, title = "Tambah Banner") }
        ) {
            Column(
                modifier = Modifier.padding(it)
            ) {
                ImagePicker()
            }
        }
    }
}
```

6. data class ReservationDetailScreen(

```
    val reservation: Reservation
) : Screen {
    @Composable
    override fun Content() {
        val navigator = LocalNavigator.currentOrThrow

        Column {

ReservationDetailTopBar(navigator = navigator)
```

```
        Column(
            modifier = Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {

ReservationTicket(reservation = reservation)
        }
    }
  }
}
```

7.
```
class ReservationDetailViewModel(private val mUseCase: IAnnaClinicUseCase) : ViewModel() {
  val getRealTimeQueue = mUseCase.getRealTimeQueue()
```

8.
```
data class ReservationFormScreen(
  val name: String,
  val price: Long,
  val products: List<Products>? = null
) : Screen {

@RequiresApi(Build.VERSION_CODES.O)
  @Composable
  override fun Content() {
    val keyboardController = LocalSoftwareKeyboardController.current
    val navigator = LocalNavigator.currentOrThrow

    Scaffold(
      topBar = {
        CenterAppBar(titleRes = R.string.reservasi, navigator = navigator)
      }
    ) {
      Column(
        modifier = Modifier
          .padding(it)
          .background(androidx.compose.ui.graphics.Color.White)
          .fillMaxSize()
          .verticalScroll(rememberScrollState())
          .pointerInput(Unit) {
            detectTapGestures(onTap = {
              keyboardController?.hide()
            })
          },
      ) {
```

```
            ReservationForm(
                serviceName = name,
                servicePrice = price,
                addOnProducts = products,
                navigator = navigator,
            )

        }
    }

}
```

9.
```
class ReservationFormViewModel(private val mUseCase: AnnaClinicUseCase) : ViewModel() {
    val queue = mUseCase.getRealTimeQueue().asLiveData()

    fun postTheReservation(reservation: Reservation) =

mUseCase.postReservation(reservation).asLiveData()
}
```

10.
```
class ReservationListScreen : Screen {

@OptIn(ExperimentalMaterial3Api::class)
    @Composable
    override fun Content() {
        val navigator = LocalNavigator.currentOrThrow
        val context = LocalContext.current
        val scrollBehavior = TopAppBarDefaults.pinnedScrollBehavior(rememberTopAppBarState())
        val viewModel: ReservationListViewModel = koinViewModel()
        val reservationList = viewModel.reservationList.collectAsState(initial = Response.Loading)
        val sheetStateProducts = rememberModalBottomSheetState(skipPartiallyExpanded = true)
        var showBottomSheetProducts by remember { mutableStateOf(false) }
        var name by remember { mutableStateOf("") }
        var price by remember { mutableStateOf(0L) }

        Scaffold(
            modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
            topBar = { CenterTopBar(navigator = navigator, title = "Reservasi") },
        ) { it ->
            Column(
```

```
            modifier = Modifier
                .padding(it)

.background(androidx.compose.ui.graphics.Color.White)
        ) {
            when (reservationList.value) {
                is Response.Loading -> {

CircularLoading(isLoading = true)
                }

                is Response.Failure -> {
                    // Display the error

CircularLoading(isLoading = false)
                }

                is Response.Success -> {

CircularLoading(isLoading = false)
                    val data = (reservationList.value as Response.Success).data
                    val numberFormat = NumberFormat.getInstance()
                    val rupiah = data.map { numberFormat.format(it.price) }

                    LazyColumn(
                        contentPadding = PaddingValues(vertical = 16.dp),
                    ) {
                        items(data) { reservation ->
                            // convert price to rupiah
                            val index = data.indexOf(reservation)
                            val rupiahPrice = rupiah[index]
                            Column(
                                modifier = Modifier.padding(horizontal = 16.dp, vertical = 8.dp)
                            ) {
                                OutlinedCard(
                                    onClick = {
                                        name = reservation.name
                                        price = reservation.price

showBottomSheetProducts = true
                                    },
                                ) {
                                    Row(
                                        modifier = Modifier

.fillMaxWidth()

.padding(16.dp),
```

```
horizontalArrangement = Arrangement.SpaceBetween,
verticalAlignment = Alignment.CenterVertically
                ) {
                    Box(
modifier = Modifier
.weight(1f)
.padding(end = 16.dp)
                    ) {
                        Text(
                            text = reservation.name,
                            style = TextStyle(
fontFamily = fontFamily,
fontSize = 16.sp,
                            )
                        )
                    }
                    Text(
                        text = "Rp. $rupiahPrice",
                        style = TextStyle(
fontFamily = fontFamily,
fontSize = 16.sp,
                        )
                    )
                }
            }
        }
    }

    is Response.Empty -> {}
  }
 }
}

when {
    showBottomSheetProducts -> {
        // Display the bottom sheet
        ModalBottomSheet(
            modifier = Modifier.safeDrawingPadding(),
            onDismissRequest = {
showBottomSheetProducts = false
            },
```

```
                sheetState = sheetStateProducts,

            ) {

ProductBottomSheetContent(
                onSkip = {
                    navigator.push(

ReservationFormScreen(
                            name = name,
                            price = price,
                        )
                    )
                    Log.d(

"ReservationListScreen",
                        "name : $name \n price : $price \nProducts onContinue: null"
                    )

showBottomSheetProducts = false
                },
                onContinue = {
                    if (it.isNotEmpty()) {
                        navigator.push(

ReservationFormScreen(
                                name = name,
                                price = price,
                                products = it
                            )
                        )
                        Log.d(

"ReservationListScreen",
                            "name : $name \n price : $price \nProducts onContinue: $it"
                        )

showBottomSheetProducts = false
                    } else {
                        Toast.makeText(
                            context,
                            "Pilih produk terlebih dahulu!!",

Toast.LENGTH_SHORT
                        ).show()
                    }
                }
            )
        }
    }
}
```

```
11. class ReservationListViewModel(mUseCase: AnnaClinicUseCase): ViewModel() {
    val reservationList = mUseCase.getTreatmentList()
}
12. fun ListProducts(
    modifier: Modifier = Modifier,
    viewModel: ProductViewModel = koinInject(),
    sendSelectedProducts: (List<Products>) -> Unit
) {
    var selectedProducts by remember { mutableStateOf(listOf<Products>()) }

    val reservationList = viewModel.productList.collectAsState(initial = Response.Loading)

    // Display the product list
    when (reservationList.value) {
        is Response.Loading -> {
            // Handle loading
            LazyColumn(
                verticalArrangement = Arrangement.spacedBy(8.dp),
            ) {
                items(10) {
                    Box(
                        modifier = modifier
                            .fillMaxWidth()
                            .height(85.dp)
                            .clip(RoundedCornerShape(16.dp))
                            .shimmer()
                            .background(color = Color.LightGray)
                    )
                }
            }
        }

        is Response.Failure -> {
            // Handle failure
        }

        is Response.Success -> {
            // Handle success
            val data = (reservationList.value as Response.Success).data

            LazyColumn(
                verticalArrangement = Arrangement.spacedBy(8.dp),
                contentPadding = PaddingValues(bottom = 16.dp)
            ) {
```

```
            items(data) {
                ProductItem(product = it) { isChecked, product ->
                    selectedProducts = if (isChecked) {
                        selectedProducts + product
                    } else {
                        selectedProducts - product
                    }
                    Log.d("ProductList", "Selected Products: $selectedProducts")

sendSelectedProducts(selectedProducts)
                }
            }
        }
    }

    is Response.Empty -> {
        // Handle empty
    }
  }
}
```

13.
```
fun ProductItem(
  modifier: Modifier = Modifier,
  product: Products,
  onProductChecked: (Boolean, Products) -> Unit
) {
  var checked by remember { mutableStateOf(false) }
  var isExpanded by remember { mutableStateOf(false) }
  val numberFormat = NumberFormat.getInstance()

  OutlinedCard(
    modifier = modifier
      .fillMaxWidth()

.animateContentSize(tween(durationMillis = 500)),
  ) {
    Row(
      modifier = modifier.fillMaxWidth(),
      verticalAlignment = Alignment.CenterVertically,
    ) {
      Box(
        modifier = modifier
          .size(100.dp)
          .padding(12.dp)

      ) {
        Column(
```

```
            modifier = modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            Card(
                shape = RoundedCornerShape(16.dp)
            ) {
                AsyncImage(
                    model = product.imageUrl,
                    contentDescription = null,
                    contentScale = ContentScale.Crop
                )
            }
        }
    }

    Box(
        modifier = modifier
            .weight(1f)
            .padding(
                end = 16.dp,
                start = 8.dp,
                top = 16.dp,
                bottom = 16.dp,
            )
    ) {
        Column(
            verticalArrangement = Arrangement.spacedBy(8.dp),
        ) {
            Text(
                text = product.name,
                style = TextStyle(fontSize = 16.sp, fontFamily = fontFamily)
            )
            Text(
                text = "Rp. ${numberFormat.format(product.price)}",
                style = TextStyle(fontSize = 16.sp, fontFamily = fontFamily)
            )
            Text(
                modifier = modifier
                    .clickable {
                        isExpanded = !isExpanded // Toggle expansion

Log.d("ProductItem", "ProductItem: ${product.detail}")
                    },
                text = if (isExpanded) "Hide Details" else "Detail",
```

```
                style = TextStyle(
                    fontSize = 16.sp,
                    fontFamily = fontFamily,
                    color = MaterialTheme.colorScheme.primary,
                    textDecoration = TextDecoration.Underline
                )
            )
        }
    }

    Row(
        modifier = modifier.padding(12.dp),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.End
    ) {
        Checkbox(
            checked = checked,
            onCheckedChange = {
                checked = it
                onProductChecked(it, product)
                Log.d("ProductItem", "ProductItem: $it \n Product: $product")
            },
        )
    }
}

// add detail below the product item
// Show details if expanded
if (isExpanded) {
    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(12.dp)
    ) {
        Text(
            text = product.detail,
            style = TextStyle(fontSize = 16.sp, fontFamily = fontFamily)
        )
    }
}
}
}
```

14.
```
data class ProfileScreen(val user: User) : Screen {
    @Composable
    override fun Content() {
        Log.d("ProfileScreen", "Content -> $user")
```

```
        val navigator = LocalNavigator.currentOrThrow
        val scrollBehavior = TopAppBarDefaults.pinnedScrollBehavior(rememberTopAppBarState())
        val prefs = koinInject<SharedPrefUtils>()

        Scaffold(
            modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
            topBar = { CenterTopBar(navigator = navigator, title = "Profile") }
        ) { paddingValues ->
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(paddingValues)
                    .verticalScroll(rememberScrollState())
            ) {
                ProfileContent(user = user) {
                    prefs.remove(Const.USER_ID)
                    prefs.remove(Const.EMAIL)

                    // navigator to login screen and clear back stack
                    navigator.replaceAll(LoginScreen())
                }
            }
        }
    }
}
```

15.

```
fun AdminScreen() {
    val navController = rememberNavController()

    Scaffold(
        modifier = Modifier.semantics {
            testTagsAsResourceId = true
        },
        bottomBar = {
            BottomNavigationBar(navController = navController)
        }
    ) {

        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(it)
                .consumeWindowInsets(it)
                .windowInsetsPadding(
                    WindowInsets.safeDrawing.only(
```

```
WindowInsetsSides.Horizontal,
                ),
            ),
        ) {
            NavigationGraph(
                navController = navController
            )
        }
    }
}
```

16.
```
fun UserScreen() {
    val navController = rememberNavController()

    Scaffold(
        modifier = Modifier.semantics {
            testTagsAsResourceId = true
        },
        bottomBar = {
            BottomNavigationBar(navController = navController)
        }
    ) {

        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(it)
                .consumeWindowInsets(it)
                .windowInsetsPadding(
                    WindowInsets.safeDrawing.only(
                        WindowInsetsSides.Horizontal,
                    ),
                ),
        ) {
            NavigationGraph(
                navController = navController
            )
        }
    }
}
```

17.
```
fun UsersQueue(
    modifier: Modifier = Modifier,
    viewModel: MainViewModel = koinInject(),
    sharedPref: SharedPrefUtils = koinInject()
) {
    val userQueue = viewModel.getQueueByEmail(sharedPref.getString(Const.EMAIL)!!).collectAsState(
        initial = Response.Loading
    )
    Card(
```

```
                modifier = modifier
                    .size(180.dp, 120.dp)
                    .clip(RoundedCornerShape(16.dp))
            ) {
                Column(
                    modifier = Modifier
                        .padding(horizontal = 24.dp, vertical = 24.dp)
                        .fillMaxSize(),
                    horizontalAlignment = Alignment.CenterHorizontally
                ) {
                    Text(text = "Antrian Kamu")
                    when (userQueue.value) {
                        is Response.Loading -> {
                            // Display the loading
                            Log.d("UsersQueue", "Loading...")
                            Column(
                                modifier = modifier.fillMaxHeight(),
                                horizontalAlignment = Alignment.CenterHorizontally,
                                verticalArrangement = Arrangement.Center
                            ) {
                                Box(
                                    modifier = modifier
                                        .padding(top = 12.dp)
                                        .size(40.dp)
                                        .clip(RoundedCornerShape(12.dp))
                                        .shimmer()
                                        .background(Color.White)
                                )
                            }
                        }
                        is Response.Success -> {
                            // Handle success
                            val queueNumber = (userQueue.value as Response.Success<Int>).data
                            Log.d("UsersQueue", "Data: $queueNumber")
                            Column(
                                modifier = modifier.fillMaxHeight(),
                                horizontalAlignment = Alignment.CenterHorizontally,
                                verticalArrangement = Arrangement.Center
                            ) {
                                Spacer(modifier = modifier.padding(2.dp))
                                Text(
                                    text = queueNumber.toString(),
                                    style = TextStyle(
                                        fontSize = 44.sp,
```

```
                        fontWeight = FontWeight.Bold,
                    )
                )
            }
        }

        is Response.Empty -> {
            // Handle empty
            Log.d("UsersQueue", "Empty")
            Column(
                modifier = modifier.fillMaxHeight(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Spacer(modifier = modifier.padding(2.dp))
                Text(
                    text = "0",
                    style = TextStyle(
                        fontSize = 44.sp,
                        fontWeight = FontWeight.Bold,
                    )
                )
            }
        }

        is Response.Failure -> {
            // Handle failure
            Log.d("UsersQueue", "Error")
            Column(
                modifier = modifier.fillMaxHeight(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Spacer(modifier = modifier.padding(2.dp))
                Text(
                    text = "0",
                    style = TextStyle(
                        fontSize = 44.sp,
                        fontWeight = FontWeight.Bold,
                    )
                )
            }
        }
    }
}
18. fun UserReservationItem(
    modifier: Modifier = Modifier,
```

```
    reservation: Reservation,
    viewModel: MainViewModel,
    onClick: () -> Unit
) {
    val realtimeQueue = viewModel.getRealTimeQueue.collectAsState(initial = Response.Loading)

    Column {
        OutlinedCard(onClick = { onClick() }) {
            Row(
                modifier = modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                horizontalArrangement = Arrangement.SpaceBetween,
                verticalAlignment = Alignment.CenterVertically
            ) {
                Box(
                    modifier = modifier
                        .weight(1f)
                        .padding(end = 16.dp)
                ) {
                    Column(
                        verticalArrangement = Arrangement.spacedBy(6.dp),
                    ) {
                        Text(
                            text = reservation.service,
                            style = TextStyle(
                                fontSize = 18.sp,
                                fontFamily = fontFamily,
                                fontWeight = FontWeight.Medium
                            )
                        )
                        if (reservation.product != "" && reservation.totalProductPrice != "") {
                            Text(
                                text = reservation.product!!,
                                style = TextStyle(fontSize = 14.sp, fontFamily = fontFamily)
                            )
                        }
                        Text(
                            text = reservation.date,
                            style = TextStyle(fontFamily = fontFamily)
                        )
                        Text(
                            text = reservation.totalPrice,
```

```
                    style =
TextStyle(fontFamily = fontFamily)
                )
            }
        }

        when
(realtimeQueue.value) {
            is Response.Loading ->
{
                Box(
                    modifier = modifier
                        .size(40.dp)

.clip(CircleShape)
                        .shimmer()

.background(color =
Color.LightGray)
                )
            }

            is Response.Success -> {
                Box(
                    contentAlignment =
Alignment.Center,
                    modifier = modifier
                        .size(40.dp)

.clip(CircleShape)
.background(Color(0xFFFFCCBC))
                ) {
                    Text(
                        text =
reservation.queue.toString(),
                        textAlign =
TextAlign.Center,
                        style =
TextStyle(
                            fontFamily =
fontFamily,
                            fontSize =
24.sp,
                            fontWeight =
FontWeight.Bold
                        ),
                    )
                }
            }

            is Response.Empty -> {}

            is Response.Failure ->
{}
        }
    }
}
```

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

PROPOSAL

Mahasiswa : Sitti Aisyah Abdullah	**Pembimbing I** : Marlina, S.Kom.,M.Kom.
NIM : 220280010	**Pembimbing II** : Mughaffir Yunus, ST.,MT.
Judul Skripsi : Aplikasi Layanan Klinik Kecantikan Berbasis Android	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1		Konsultasi 1 - Integrasi sistem layanan member - Perbandingan dan persamaan penelitian - sistem usul Flowchart	
Konsultasi 2		Konsultasi 2 - kajian teori klinik - BAB III lengkapi	
Konsultasi 3 Acc Ujian Proposal		Konsultasi 3 Acc	
Konsultasi 4		Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

Hasil

Mahasiswa : Sitti Aisyah Abdullah	**Pembimbing I : Marlina, S.Kom.,M.Kom.**
NIM : 220280010	**Pembimbing II : Mughaffir Yunus, ST.,MT.**
Judul Skripsi : Aplikasi Layanan Klinik Kecantikan Berbasis Android	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Uji Coba Aplikasi pada Tempat Penelitian		Konsultasi 1 1. Tipe akun daftar dikapus 2. Alur	
Konsultasi 2 Buat Laporan Harian, bulanan dan tahunan		Konsultasi 2 - Antrian Relevang mulai 0 - Cek Resensi 1x sehari	
Konsultasi 3		Konsultasi 3 - penulisan - abstrak - Tabel Judul + sumber - Flowchart Bab III. - analisis kebutuhan Cad 9 dan Perancangan.	
Konsultasi 4 Acc Ujian Hasil		Konsultasi 4 Acc	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

SKRIPSI

Mahasiswa : Sitti Aisyah Abdullah	**Pembimbing I : Marlina, S.Kom.,M.Kom.**
NIM : 220280010	**Pembimbing II : Mughaffir Yunus, ST.,MT.**
Judul Skripsi : Aplikasi Layanan Klinik Kecantikan Berbasis Android	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Perbaiki Format penulisan		Konsultasi 1 Perbaiki sesuai petunjuk penguji.	
Konsultasi 2		Konsultasi 2 Acc Ujian Tutup	
Konsultasi 3		Konsultasi 3	
Konsultasi 4 Acc Ujian Tutup		Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik