|  |  |
| --- | --- |
| **Coursework #2** | |
| **Module’s Information:** | |
| Module | IBM4203 Mobile Apps Development using Android  PRG4203 Mobile Application Development |
| Session | JAN2025 |
| Programme | BCSI / BITI |
| Lecturer | Lai Mei Yoon |
| Coursework Type | Assignment 2 (Individual) |
| Percentage | 30% |
| Hand-out Date | Week 6 |
| Due Date | Week 10 |
| **Student’s Declaration:** | |
| ***We declare that:***   1. ***We understand what is meant by plagiarism*** 2. ***This assignment is all our own work and we have acknowledged any use of the published or unpublished works of other people.*** 3. ***We hold a copy of this assignment which we can produce if the original is lost or damaged***  |  |  |  |  | | --- | --- | --- | --- | | **Name** | **ID** | **Signature** | **Mark** | | **Anup Bamrel** | **I25032049** |  |  | | |

|  |  |
| --- | --- |
| **Learning Outcomes Assessed:** | |
| LO2 | Demonstrate an appropriate user interface design and logic using variety of Android programming techniques. (P5, PLO3) |
| **Penalty for late submission:** | |
| 1 day – minus 20% of total mark awarded.  2 days – minus 50% of total mark awarded.  3 days – 0 mark for this piece of coursework. | |

**Rubric**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Excellent** | **Good** | **Satisfactory** | **Needs Improvement** | **Your Score** |
| **Functionality (16)** | All required features are implemented and work flawlessly. **(13-16)** | Most features are implemented and work with minor issues. **(9-12)** | Some features are implemented, but there are significant issues. **(5-8)** | Few features are implemented, and there are major issues. **(1-4)** |  |
| **UI/UX Design (16)** | The app has an intuitive, user-friendly interface with excellent visual design. **(13-16)** | The app has a good interface with minor usability issues. **(9-12)** | The app has a basic interface with several usability issues. **(5-8)** | The app has a poor interface with major usability issues. **(1-4)** |  |
| **Data Handling (12)** | Efficiently handles data passing between activities. **(10-12)** | Handles data passing with minor issues. **(7-9)** | Handles data passing but has significant issues. **(4-6)** | Poor handling of data passing between activities. **(1-3)** |  |
| **External Integrations (8)** | Successfully integrates external features (maps, web links, calls, etc.). **(7-8)** | Integrates external features with minor issues. **(5-6)** | Integrates some external features but has significant issues. **(3-4)** | Poor or no integration of external features. **(1-2)** |  |
| **Creativity and Innovation (8)** | Demonstrates creativity and innovation in app design and functionality. **(7-8)** | Shows some creativity and innovation. **(5-6)** | Basic app design with limited creativity. **(3-4)** | Lacks creativity and innovation. **(1-2)** |  |
| **Code Quality (8)** | Code is clean, well-organized, and follows best practices. **(7-8)** | Code is mostly clean and organized, with minor deviations from best practices. **(5-6)** | Code is somewhat organized but has several issues and does not follow best practices. **(3-4)** | Code is poorly organized and does not follow best practices. **(1-2)** |  |
| **Documentation (20)** | Well-written and well-organized report. It contains all the required information. **(16-20)** | Clear and organized report with minor details missing. **(11-15)** | Basic report with some organization. Some required information is missing. **(6-10)** | Incomplete or poorly organized report. **(1-5)** |  |
| **Presentation (12)** | Engaging and well-structured presentation. Effectively communicates design. **(10-12)** | Clear and structured presentation. Communicates design well. **(7-9)** | Basic presentation with some structure. Communicates design adequately. **(4-6)** | Incomplete or poorly structured presentation. Fails to communicate design. **(1-3)** |  |
|  |  |  |  | **Total** |  |

**Notes:**

* **Excellent:** Exceeds expectations in all aspects.
* **Good:** Meets expectations with minor improvements needed.
* **Satisfactory:** Meets basic requirements but lacks depth.
* **Needs Improvement:** Does not meet basic requirements.

|  |
| --- |
| **Lecturer’s Overall Comments** |
|  |

**Warning:**

It is important to include citation and references in the assignment. You are required to upload your document to Turnitin for plagiarism checks which advisable not more than 25% out of 100%. Please read the following statement properly to avoid penalty:

**“Plagiarism & Copying”**

Plagiarism and copying are strictly prohibited AND CONSIDERED A SERIOUS OFFENCE. THE FACULTY HAS THE FULL RIGHTS TO TAKE DISCIPLINARY ACTIONS AGAINST ANY STUDENT WHO PLAGIARISE OR COPY THE WORK OF OTHERS. AN ACCEPTABLE TURN IT IN / SAFE ASIGN SIMILARITY REPORT MUST BE INCLUDED AS APPENDIX FOR ALL SUBMITTED ASSIGNMENTS. THE TOTAL PERCENTAGE IN THE REPORT SHOULD LESS THAN 25% and EACH INDIVIDUAL ITEM IN THE REPORT SHOULD LESS THAN 3%. Your assignment will NOT BE GRADED if it does not meet the above threshold.

The Faculty reserves the right to use any means to identify plagiarism and copying. If you reference the work of other authors, you must ensure that you properly reference them to avoid the charge of plagiarism. Copying the work of others (current students, past students or the works of others) automatically earns you a Fail grade. If two or more students copied from each other, both students (regardless of who copies from whom) will earn a Fail grade.”

|  |
| --- |
| **Description of Coursework #2** |

Develop a doctor appointment app for a clinic with the following features or functions: -

* **Doctor profiles** — Users should be able to browse doctor profiles to choose the doctor they like. Profiles should contain information on certifications, experience, competencies, and languages spoken.
* **Appointments booking** — Users should be able to schedule an appointment with a doctor using a calendar and time pickers. The booking form should contain various type of input controls such as EditText, radio buttons, check boxes and spinner. Data validation is required when the user submits the form. The information needs to be passed to the next activity and displayed as a booking confirmation.
* **Clinic information** – The user should be able the view the information about the clinic. You need to use implicit intent(s) to carry out activities, such as opening the clinic website, opening the clinic location on a map

You are required to submit a **report** that contains the following:

1. Cover Page
2. Rubric
3. Table of Content
4. Application title, objective, and storyboard
5. Features and Functions
6. Code explanations
7. Screenshots of app
8. Lesson learnt and conclusion.
9. Code listings as attachment

A presentation with app demonstration is required after the submission.

|  |
| --- |
| **End of Coursework #2** |

TABLE OF CONTENTS

1.0 application title7

2.0 objective7

3.0 storyboard 7

4.0 features and functions11

5.0 code explainations13

6.0 screenshots of app 22

7.0 lesson learnt 24

8.0 conclusion 26

9.0 code listings as attachment 26

10.0 references 26

**APPLICATION TITLE**

BookDoc

**OBJECTIVE**

The **BookDoc** app is engineered to book appointments with doctors seamlessly. Users can search doctor profiles, make appointments, and check information about clinics. The end intent is to make healthcare services more open through an easy and friendly mobile application.

**STORYBOARD**

|  |  |
| --- | --- |
| **Figure** | **Explaination** |
|  | The application starts with a Home Screen, where users are presented with a Get Started button. Click button to proceed. |
|  | Upon clicking, they are taken to the Doctor List Screen, which features a search bar that enables users to search for doctors by name or specialty. Below the search bar, a list of doctors is displayed, showing their images, names and specialties. Users can click on a doctor’s card to access their Doctor Profile Screen. |
|  | Users are allowed to view detailed information about the doctor on the Doctor Profile Screen, such as name with images, specialities, certifications, experience, competencies, languages spoken and other information provided by the doctor. Moreover, the clinic also displays the clinic’s image, name, location, the clinic’s website, opening hours and the contact number of the clinic. |
|  | Clicking on location opens the clinic’s address in Google Maps. |
|  | Clicking on website opens the clinic’s website in a browser. |
|  | Clicking on call icon initiates a phone call via the dial pad. |
|  | If users click Book Appointment button in the above form, it takes to the Appointment Booking Screen, which is a form in which the user has to enter its personal data such as name, date of birth, gender, contact number, email, street, city, state and postal/zip code. On top of that, users can choose the procedure type, the date on which they prefer their appointment, as well as the slots that are available. |
|  | The user can select their date of birth by clicking on the DOB field, which triggers a calendar pop-up. The user can then select the date from the calendar, which will be displayed in the field in the format YYYY-MM-DD. |
|  | The user can select gender from a dropdown with three options: Male, Female, and Other. |
|  | All the fields also go under validation check to make sure that they are being filled correctly before submission. After completing, users can then click **Confirm Appointment** button to take them to the Appointment Confirmation Screen. Users are greeted on the Appointment Confirmation Screen with a summary of the booking information. The two buttons we have here are Share and Done, Share will allow the user to share their appointment details and Done will bring him back to Doctor List Screen. |

**FEATURES AND FUNCTIONS**

**Search Doctor:**

* Users can search for doctors by name or specialty using a search bar.
* As users type, the list filters dynamically to show matching results.
* The doctor list updates in real time based on the search query.

**Doctor Profiles:**

* Users can browse a list of doctors.
* Each profile includes details such as certifications, experience, competencies, and languages spoken.
* A Read More / Read Less toggle allows users to expand or collapse detailed information about the doctor.

**Clinic Information:**

* Users can view details about the clinic, including name, location, website, and contact number.
* Implicit intents allow users to:

Open the clinic’s website in a browser by clicking on the website text.

View the clinic’s location in Google Maps by clicking on the location text.

Initiate a phone call by clicking on the contact number text.

**Appointment Booking:**

* Users can schedule an appointment using a Date Picker and a Time Picker.
* The form includes the following fields:

**Name:** Required text input.

**Date of Birth (DOB):** Clicking this field opens a calendar pop-up for date selection.

**Gender Selection:** Users can choose from three options (Male, Female, Other) in a dropdown menu.

**Phone Number:** Input field with validation to ensure exactly 10 digits are entered.

**Email Address:** Input field with format validation to ensure a correctly structured email.

**Street, City, State, and Zip Code:** These are required address fields, with the Zip Code field validated to ensure exactly 5 digits are entered.

**Procedure Type:** Users select their procedure from a dropdown menu.

**Preferred Appointment Date & Available Time Slots:** Users can select an appointment date and time.

* Validation checks ensure all fields are correctly filled before submission.
* Upon confirmation, appointment details are passed to the next screen for review.

**Appointment Confirmation:**

* Users are shown a summary of their appointment details after booking.
* Two buttons are provided:

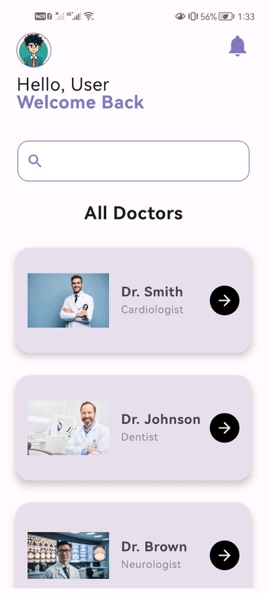
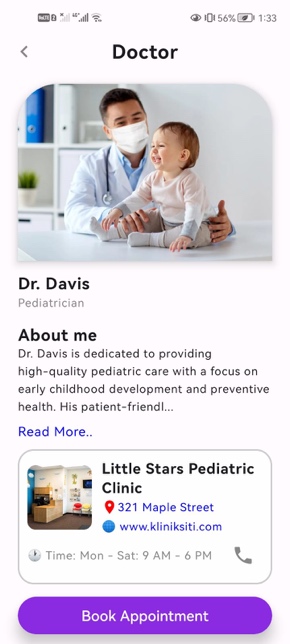
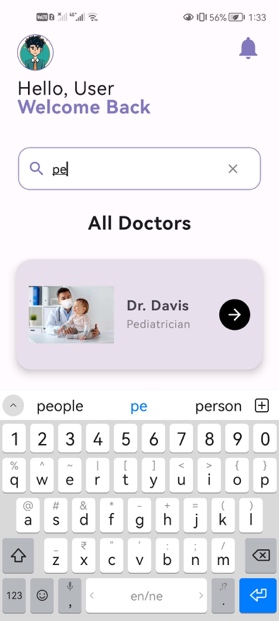
**Share:** Allows users to share their appointment details.

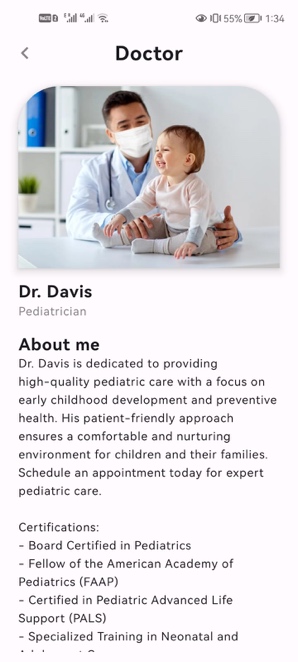
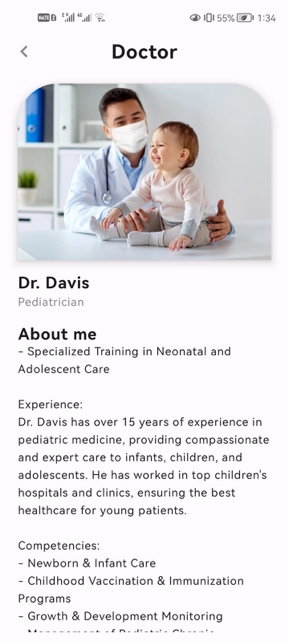
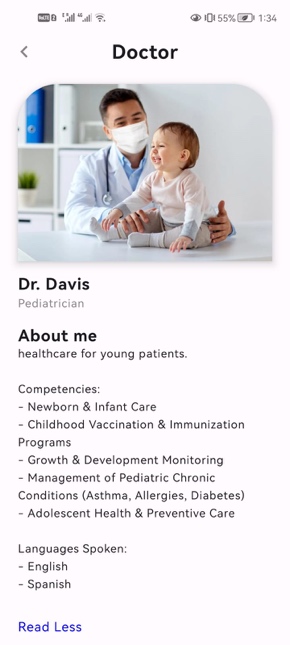
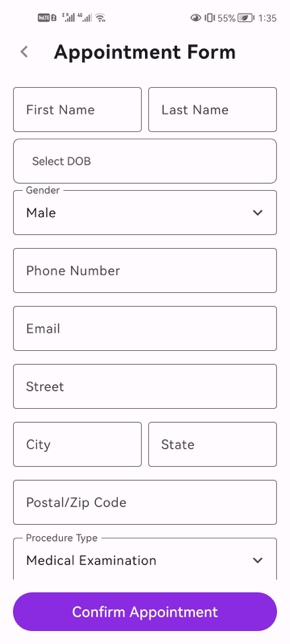
**Done:** Redirects them back to the Doctor List Screen.

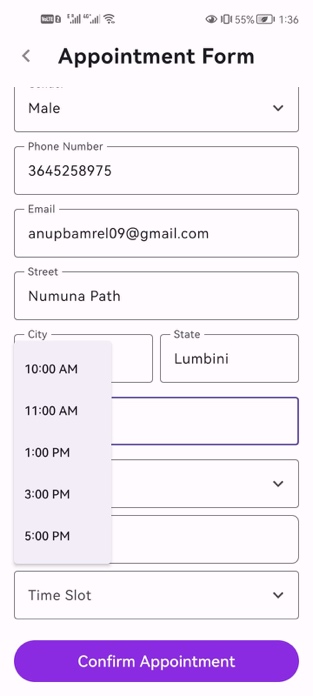
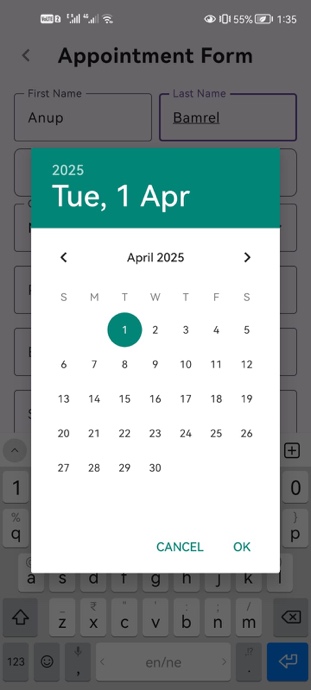
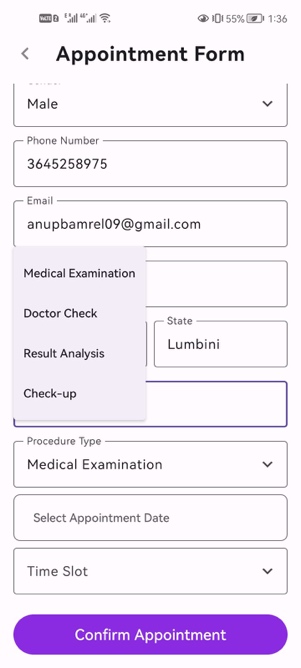
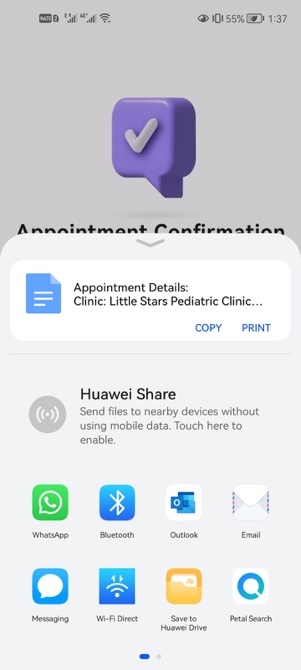
**CODE EXPLANATION**

|  |  |
| --- | --- |
| **Code** | **Explaination** |
| **Intro.kt** | |
| @Composable fun AppNavigation(navController: NavHostController) {  NavHost(navController = navController, startDestination = "home") **{** *composable*("home") **{** HomeScreen(navController) **}** *composable*("next")**{**DoctorListScreen(navController)**}** *composable*("doctorList") **{** DoctorListScreen(navController) **}** *composable*("doctorDetails/{name}/{specialty}/{description}/{clinicName}/{clinicLoc}/{clinicNo}/{doctorImage}/{clinicImage}/{url}") **{** backStackEntry **->** val name = backStackEntry.arguments?.getString("name") ?: ""  val specialty = backStackEntry.arguments?.getString("specialty") ?: ""  val description = backStackEntry.arguments?.getString("description") ?: ""  val clinicName = backStackEntry.arguments?.getString("clinicName") ?: ""  val clinicLoc = backStackEntry.arguments?.getString("clinicLoc") ?: ""  val clinicNo = backStackEntry.arguments?.getString("clinicNo") ?: ""  val doctorImage = backStackEntry.arguments?.getString("doctorImage") ?: ""  val clinicImage = backStackEntry.arguments?.getString("clinicImage") ?: ""  val url = Uri.decode(backStackEntry.arguments?.getString("url") ?: "")   DoctorDetailsScreen(  name, specialty, description,  clinicName,  clinicLoc,  clinicNo,  doctorImage,  clinicImage,  url,  navController = navController  )  **}** *composable*("appointment/{name}/{clinicName}") **{** backStackEntry **->** val name = backStackEntry.arguments?.getString("name") ?: ""  val clinicName = backStackEntry.arguments?.getString("clinicName") ?: ""  AppointmentScreen(name, clinicName, navController = navController)  **}** *composable*("confirmation/{name}/{clinicName}") **{** backStackEntry **->** val name = backStackEntry.arguments?.getString("name") ?: ""  val clinicName = backStackEntry.arguments?.getString("clinicName")?:""  ConfirmationScreen(name, clinicName, navController=navController)  **}  }** } | **Navigation Setup (AppNavigation Function):**  Using Jetpack Compose’s NavHost, the navigation structure is set up by the AppNavigation setup that is called from various places in the project.  Mainly it defines multiple routes like:   * home → **HomeScreen** * doctorList → **DoctorListScreen** * doctorDetails/{name}/{specialty}/{description}/{clinicName}/{clinicLoc}/{clinicNo}/{doctorImage}/{clinicImage}/{url} → **DoctorDetailsScreen** * appointment/{name}/{clinicName} → **AppointmentScreen** * confirmation/{name}/{clinicName} → **ConfirmationScreen**   BackStackEntry arguments is used to get such data into doctorDetails and appointment screen. |
| @Composable fun HomeScreen(navController: NavController) {  Box(  modifier = Modifier.*fillMaxSize*()  ) **{** Image(  painter = painterResource(id = R.drawable.*front\_page*),  contentDescription = "Background Image",  contentScale = ContentScale.Crop,  modifier = Modifier.*fillMaxSize*()  )   Button(  onClick = **{** navController.navigate("doctorList") **}**,  shape = *RoundedCornerShape*(30.*dp*),  modifier = Modifier  .*align*(Alignment.BottomCenter)  .*padding*(bottom = 55.*dp*)  .*height*(50.*dp*)  .*shadow*(8.*dp*, *RoundedCornerShape*(30.*dp*)), // Adds shadow effect  colors = ButtonDefaults.buttonColors(  containerColor = *Color*(123, 207, 223) // Example RGB color (Blue)  ),  elevation = ButtonDefaults.elevatedButtonElevation(8.*dp*)  ) **{** Text(text = "Get Started", fontSize = 18.*sp*, color = Color.White)  **}  }}** | **Home Screen Implementation:**  It has a background image and a Get Started button to display via the HomeScreen function.  The button is styled with:   * A rounded shape (30.dp) * A shadow effect (8.dp elevation) * A custom light blue color (RGB: 123, 207, 223)   It will be navigated to the Doctor List Screen when clicked by using the navController.navigate("doctorList"). |
| val url = Uri.decode(backStackEntry.arguments?.getString("url") ?: "") | This ensures that special characters in URLs are properly handled when passed between screens. |
| **DoctorList.kt** | |
| data class Doctor(  val name: String,  val specialty: String,  val description: String,  val clinicName: String,  val clinicLoc: String,  val clinicNo: String,  val doctorImage: String,  val clinicImage: String,  val url: String ) | Defines a data class Doctor with properties like name, specialty, clinic details, and images. |
| @Composable fun UserProfileImage() {  Row(  modifier = Modifier  .*fillMaxWidth*()  )**{** // profile image  Image(  painter = painterResource(id = R.drawable.*user\_pic*), // Replace with your image  contentDescription = "User Profile Picture",  contentScale = ContentScale.Crop,  modifier = Modifier  .*size*(47.*dp*)  .*clip*(*CircleShape*)  .*border*(1.*dp*, *Color*(130,120,189), *CircleShape*)  )   Spacer(modifier = Modifier.*weight*(1f))   // notification icon  Icon(  imageVector = Icons.Default.*Notifications*,  contentDescription = "Notifications",  tint = *Color*(130,120,189),  modifier = Modifier  .*size*(35.*dp*)   )  **}** } | The profile image and notification icon are arranged horizontally in the horizonal version of UserProfileImage composable. The notification icon is displayed as the default notification icon, and the profile image is in the shape of a circle with a border. To keep clear and balanced layout I push the notification icon to the right with a spacer. |
| @Composable fun SearchBar(searchQuery: String, onSearchQueryChange: (String) -> Unit) {  Box(  modifier = Modifier  .*fillMaxWidth*(0.9f)  .*height*(55.*dp*)  .*clip*(*RoundedCornerShape*(15.*dp*))  .*background*(Color.White)  .*border*(1.*dp*, *Color*(130, 120, 189), *RoundedCornerShape*(15.*dp*)),  contentAlignment = Alignment.CenterStart  ) **{** Row(  verticalAlignment = Alignment.CenterVertically,  modifier = Modifier.*padding*(horizontal = 12.*dp*)  ) **{** Icon(  imageVector = Icons.Default.*Search*,  contentDescription = "Search",  tint = *Color*(130, 120, 189),  modifier = Modifier.*size*(24.*dp*)  )  Spacer(modifier = Modifier.*width*(8.*dp*))  BasicTextField(  value = searchQuery,  onValueChange = onSearchQueryChange,  textStyle = TextStyle(  fontSize = 16.*sp*,  color = Color.Black  ),  singleLine = true,  modifier = Modifier.*weight*(1f)  )  if (searchQuery.*isNotEmpty*()) {  IconButton(onClick = **{** onSearchQueryChange("") **}**) **{** Icon(  imageVector = Icons.Default.*Close*,  contentDescription = "Clear",  tint = Color.Gray,  modifier = Modifier.*size*(20.*dp*)  )  **}** }  **}  }** } | The SearchBar composable creates a custom search input field with a search icon on the left and a clear icon on the right. It is enclosed in a rounded box with a border. The BasicTextField is used to capture and display the search query, and the onSearchQueryChange function updates the query state. If the search query is not empty, the clear icon allows users to reset the search. The layout is styled with padding and colors to create an attractive search bar. |
| var searchQuery by remember **{** *mutableStateOf*("") **}** val filteredDoctors = *doctorsList*.*filter* **{ it**.name.*contains*(searchQuery, ignoreCase = true) ||  **it**.specialty.*contains*(searchQuery, ignoreCase = true)  LazyColumn(  modifier = Modifier.*fillMaxSize*() ) **{** *items*(filteredDoctors) **{** doctor **->** DoctorItem(doctor = doctor, onClick = **{** Log.d("ClinicCard", "Opening URL: ${doctor.url}")  navController.navigate("doctorDetails/${doctor.name}/${doctor.specialty}/${doctor.description}/${doctor.clinicName}/${doctor.clinicLoc}/${doctor.clinicNo}/${doctor.doctorImage}/${doctor.clinicImage}/${Uri.encode(doctor.url)}")  **}**)  **} }** | This code allows for real time search as in it filters doctorsList based on the searchQuery, matching on doctor’s name or specialty case in sensitively. The searchQuery is placed as a state variable and updated dynamically in the UI.  The LazyColumn would render a list filtered down of doctors, and each item on the list would be a DoctorItem. The URL for an item clicked is logged by Log.d("ClinicCard", "Opening URL: ${doctor.url}"), because it can be helpful for debugging navigation issues. It then lands on doctorDetails, navController to send doctor details and it uses Uri.encode(doctor.url) to send safe URL. |
| @Composable fun DoctorItem(doctor: Doctor, onClick: () -> Unit) {  Card(  modifier = Modifier  .*fillMaxWidth*()  .*padding*(15.*dp*)  .*clickable*(onClick = onClick),  shape = *RoundedCornerShape*(20.*dp*),  elevation = CardDefaults.cardElevation(defaultElevation = 8.*dp*)  ) **{** Box(modifier = Modifier.*padding*(16.*dp*)) **{** Row(  modifier = Modifier.*fillMaxWidth*(),  verticalAlignment = Alignment.CenterVertically  ) **{** val context = *LocalContext*.current  val doctorImageRes = context.*resources*.getIdentifier(doctor.doctorImage, "drawable", context.*packageName*)  Image(  painter = painterResource(id = doctorImageRes),  contentDescription = "Doctor Icon",  modifier = Modifier.*size*(110.*dp*)  )  Spacer(modifier = Modifier.*width*(16.*dp*))  Column **{** Text(text = doctor.name, fontSize = 18.*sp*, fontWeight = FontWeight.Bold)  Text(text = doctor.specialty, fontSize = 14.*sp*, color = Color.Gray)  **}** Spacer(modifier = Modifier.*weight*(1f))  Box(  modifier = Modifier  .*size*(40.*dp*) // Adjust the size as needed  .*background*(Color.Black, shape = *CircleShape*), // Black circular background  contentAlignment = Alignment.Center  ) **{** Icon(  imageVector = Icons.Default.*ArrowForward*,  contentDescription = "Right Arrow",  tint = Color.White, // White arrow  modifier = Modifier.*size*(24.*dp*) // Adjust size of arrow  )  **}   }  }  }** } | DoctorItem composable is a card containing single doctor info that can be clicked. The doctor’s image, name, and specialty appear. The dynamic image is fetched according to doctor data. The card has a visual effect provided by rounded corners and elevation. It will trigger onClick action in onClick when clicked and may be used to navigate to doctor’s details. On the right side of the card an arrow icon is displayed that indicates that it is possible to navigate. |
| **Doctor\_Profile.kt** | |
| private fun openLocation(context: Context, location: String){  val intent = Intent(Intent.*ACTION\_VIEW*, Uri.parse("geo:0,0?q=$location"))  context.startActivity(intent) } | The openLocation function opens a location via a map app. It accepts context object and location string as a parameters. It generates an Intent with the ACTION\_VIEW action and a geo: URI which coerces the given location into. It next begins the activity to open the location in a map application. |
| private fun openWebsite(context: Context, url: String){  val intent = Intent(Intent.*ACTION\_VIEW*, Uri.parse(url))  context.startActivity(intent) } | openWebsite function opens a URL on a web browser. Its parameter is a Context and a string url The meaning of this is that it creates an Intent with ACTION\_VIEW action and whatever URL is passed. It then proceeds to open the activity to do the URL in the web browser. |
| @Composable fun DoctorDescription(description: String) {  var expanded by remember **{** *mutableStateOf*(false) **}** Column(modifier = Modifier.*verticalScroll*(rememberScrollState())) **{** Text(  text = if (expanded) description else description.*take*(150)+"...",  fontSize = 14.*sp*,  lineHeight = 22.*sp* )   Spacer(modifier = Modifier.*height*(8.*dp*))   Text(  text = *buildAnnotatedString* **{** append(if (expanded) "Read Less" else "Read More..")  addStyle(  style = SpanStyle(color = Color.Blue),  start = 0,  end = length  )  **}**,  fontSize = 16.*sp*,  modifier = Modifier.*clickable* **{** expanded = !expanded **}** )  **}** } | The doctor description DoctorDescription composable is used to display the doctor’s description text. First 150 characters of the description appear with an ellipsis ("...") at first. When the "Read More.." When the text is clicked, the full description shows how the text changes to "Read Less" `remember` is used by the component to remember the expanded state and make vertical scrolling possible with long descriptions. |
| Row(  verticalAlignment = Alignment.CenterVertically ) **{** Text(  text = "🌐 ${Uri.parse(url).*host*}",  fontSize = 14.*sp*,  color = Color.Blue,  modifier = Modifier  .*clickable* **{** *openWebsite*(context, url)  **}** ) **}** | It’s a Text composable inside Row, and wrapped inside a Modifier.clickable so that it can be interactive. To get just the host (domain) part of a full URL, such as "www.example.com", we will use Uri.parse(url).host which will be displayed as text. This text is a click handler, and when the user clicks on this text it will trigger the clickable modifier function which is in fact openWebsite. This was probably a function, called from elsewhere, to start an intent to open the given URL in a browser. To perform this action the openWebsite function is called with the parameters being the current context, as well as the url string. |
| IconButton(onClick = **{** val intent = Intent(Intent.*ACTION\_DIAL*, Uri.parse("tel:$phoneNumber"))  context.startActivity(intent) **}**) | When the IconButton is clicked, it triggers a phone call via just the phone icon. On top of that, it writes the action ACTION\_DIAL, opening up the dialer with the specified phone number ($phoneNumber) as an Intent. The startActivity method launches the dialer and the phone number is already ready to be dialed. |
| val doctorImageRes = context.*resources*.getIdentifier(doctorImage, "drawable", context.*packageName*) val clinicImageRes = context.*resources*.getIdentifier(clinicImage, "drawable", context.*packageName*) | * getIdentifier() is a method used to fetch a resource identifier dynamically by its name. * "drawable" specifies that the resources are located in the drawable folder. * context.packageName is used to provide the package name of the current app, which is required to resolve the resource within the app's scope. |
| **{** navController.navigate("appointment/${Uri.encode(name)}/${Uri.encode(clinicName)}") **}** | This code navigates to the "appointment" screen, passing the doctor’s name and clinicName as URL-encoded parameters, ensuring special characters are safely handled in the URL. |
| **AppointmentScreen.kt** | |
| var firstName by remember **{** *mutableStateOf*("") **}** var lastName by remember **{** *mutableStateOf*("") **}** var dob by remember **{** *mutableStateOf*("") **}** var selectedGender by remember **{** *mutableStateOf*("Male") **}** var phoneNumber by remember **{** *mutableStateOf*("") **}** var street by remember **{** *mutableStateOf*("") **}** var city by remember **{** *mutableStateOf*("") **}** var state by remember **{** *mutableStateOf*("") **}** var zipCode by remember **{** *mutableStateOf*("") **}** var email by remember **{** *mutableStateOf*("") **}** var selectedProcedure by remember **{** *mutableStateOf*("Medical Examination") **}** var selectedDate by remember **{** *mutableStateOf*("") **}** var selectedTime by remember **{** *mutableStateOf*("") **}** val genderOptions = *listOf*("Male", "Female", "Other") val procedureOptions = *listOf*("Medical Examination", "Doctor Check", "Result Analysis", "Check-up") val timeSlots = *listOf*("10:00 AM", "11:00 AM", "1:00 PM", "3:00 PM", "5:00 PM")  var phoneError: String by remember **{** *mutableStateOf*("") **}** var emailError: String by remember **{** *mutableStateOf*("") **}** var submitted by remember **{** *mutableStateOf*(false) **}** var zipError by remember **{** *mutableStateOf*("") **}** var dobError by remember **{** *mutableStateOf*("") **}** var timeError by remember **{** *mutableStateOf*("") **}** var appointmentDateError by remember **{** *mutableStateOf*("") **}** | mutableStateOf is used by the code to make form field variables observable by creating state variables that start with firstName, dob, selectedGender etc. The variables store user inputs and user selections. The remember makes sure that the state does not disappear after a UI reconsition. Lists store the predefined options for gender, procedures, and time slots reducing the user choices. Finally, error variables like phoneError, emailError hold error messages that can trigger error messages when the error variable is true. submitted tracks when the form is submitted to modulate the UI behavior. With this setup, it’s possible to have real time updates on UI from the user interactions. |
|  | This code uses the default Scaffold to frame the structure using predefined slots for topBar, bottomBar, and the rest of the content. A back button, title, and a confirmation button are placed in the topBar and within the bottomBar. A form is loaded inside the LazyColumn, which is the main content area and used for the efficient scroll. This is to provide the padding values to make sure there is proper spacing between UI elements, in particular, when working with the padding of the Scaffold and nested components. Inside the LazyColumn,padding(paddingValues) is used so that the content is laid out properly, not overlapping anything above or beneath, and ensuring a clean UI. It allows the form to scroll without tarring the layout structure.  In LazyColumn, the item must provide a query which defines the individual items the scrollable list has. Inside of it, the pieces of UI content are in a LazyColumn. |
| Column(horizontalAlignment = Alignment.Start) **{** BoldTitleText("Clinic", clinicName)  BoldTitleText("Doctor", name)  BoldTitleText("Name", "${data.firstName} ${data.lastName}")  BoldTitleText("DOB", data.dob)  BoldTitleText("Gender", data.selectedGender)  BoldTitleText("Phone", data.phoneNumber)  BoldTitleText("Email", data.email)  BoldTitleText("Address", "${data.street}, ${data.city}, ${data.state}, ${data.zipCode}")  BoldTitleText("Procedure", data.selectedProcedure)  BoldTitleText("Date", data.selectedDate)  BoldTitleText("Time", data.selectedTime) **}** | This Column aligns patient and appointment details left as noted, vertically (Alignment.Start). Bold Title text label and value for each detail is used which is formatted in a structured and readable format. It will fetch the data from data having appointment details and the present it in a key value fashion i.e. "Clinic: XYZ", "Doctor: Dr. ABC". |
| val intent = Intent(Intent.*ACTION\_SEND*).*apply* **{** *type* = "text/plain"  putExtra(Intent.*EXTRA\_TEXT*, shareText) **}** context.startActivity(Intent.createChooser(intent, "Share Appointment Details")) | * **Intent Creation:** It creates an Intent with the action Intent.ACTION\_SEND, which is used to send data to other apps. * **Setting the Type:** The type is set to "text/plain", indicating that the content being shared is plain text. * **Adding Data:** The putExtra() method adds the actual content (shareText) to be shared as Intent.EXTRA\_TEXT. * **Starting the Activity:** The context.startActivity() launches the sharing process using Intent.createChooser(). This shows a dialog allowing the user to choose which app to use for sharing the content. |
| DatePickerDialog(  context,  **{** \_, year, month, dayOfMonth **->** val formattedDate = "$dayOfMonth/${month + 1}/$year"  onDateSelected(formattedDate)  **}**,  calendar.get(Calendar.*YEAR*),  calendar.get(Calendar.*MONTH*),  calendar.get(Calendar.*DAY\_OF\_MONTH*) ).show() | In this code, we open a DatePickerDialog in an Android app for the users to pick a date. It sets the dialog with the current date by calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH), and calendar.get(Calendar.DAY\_OF\_MONTH) When the user selects a date this lambda function is triggered, based on which the date is formatted as day/month/year and is passed on to the onDateSelected function. After that, the dialog is displayed with show() method. |
| emailError = if (Pattern.matches("[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}", **it**)) "" else "Invalid email format" | It is using a regular expression for validating this line to do email validation. In this, the Pattern.matches() function checks whether the entered email (it) is in the format of an email (e.g., example@domain.com). The if statement is only dealt with if the email matches said pattern, in which case emailError is set to an empty string (no error) indicating no error. If the input is otherwise, it returns the value of "Invalid email format" to say the input is not an email. |
| phoneError = if (**it**.length == 10 && **it**.*all* **{** char **->** char.*isDigit*() **}**) "" else "Invalid phone number" | Having said that, in this line, we have validated a phone number in checking two conditions.   * it.length == 10 → Verifies the length of the phone number is 10 digits. * it.all { char -> char.isDigit() } is guaranteed to be fulfilled.   In this case, if both conditions, the attempt to launch the url and the response to the attempt, the url launch turns out to be an empty string (no error). It is set to "Invalid phone number" otherwise, which means incorrect input. |
| **AppointmentData.kt** | |
| object AppointmentData {  var firstName: String = ""  var lastName: String = ""  var dob: String = ""  var selectedGender: String = ""  var phoneNumber: String = ""  var email: String = ""  var street: String = ""  var city: String = ""  var state: String = ""  var zipCode: String = ""  var selectedProcedure: String = ""  var selectedDate: String = ""  var selectedTime: String = "" } | AppointmentData is a Singleton storing appointment details globally, making data accessible across screens without explicit passing. It simplifies navigation by temporarily holding patient and appointment info. |

**SCREENSHOTS OF APP**

**­­**

****

****

**LESSON LEARNT**

1. **Jetpack Compose for UI**

* The combination of Column, Row, LazyColumn and Scaffold creates declarative and efficient development processes for the user interface.
* Modifier provides tools to design interfaces through alignment features and styling elements and padding options. These features enable better presentation in UI structures.

1. **State Management**

* Before managing UI state dynamically we need to establish the important mutableStateOf("") feature with remember.
* The real-time functionality of all fields comes from state variables which update instantaneously.

1. **Navigation & Data Passing**

* The correct syntax for navigating between screens and passing data is navController.navigate("route/param1/param2").
* The Uri.encode() function adds encoding to stop online navigation failures from specific text characters.

1. **Form Handling & Validation**

* A phone number validation system together with email and ZIP code checks protects form entries from invalid information.
* Error messages such as emailError and phoneError demonstrate helpful functionality since they educate users about correct input.

1. **Reusability & Clean Code**

* The codebase benefits from user interface scalability because the developer created functions that can be reused multiple times (BoldTitleText, CustomTextField, DropdownField).
* The appointment details storage before confirmation uses object AppointmentData as a Singleton implementation.

1. **Debugging & Logging**

* The statement "Log.d("Tag", "Message") tracks user actions while permitting developers to detect errors.
* This functionality helps developers solve bugs by recording the users' movement through screens and system API requests.

1. **Intent Handling for External Actions**

* Intent.ACTION\_DIAL enables phone calls to doctors and Intent.ACTION\_SEND enables appointment sharing options through selected apps.
* When users need to select a sharing or dialing application they will find this option possible through Intent.createChooser().

1. **Search & Filtering**

* The search functionality is possible through the expression doctorsList.filter { it.name.contains(searchQuery, ignoreCase = true) }.

1. **Date Selection & Formatting**

* The combination of DatePickerDialog with the formatting rule "$day/$month/$year" gives users the ability to select dates while maintaining uniform date presentation.

1. **API/Resource Handling**

* The API method context.resources.getIdentifier() automatically fetches doctor and clinic images.
* The system provides flexible asset loading which does not depend on hardcoded drawable references.

1. **Extracting Website Domain Name**

* The parser Uri.parse(url) provides the ability to retrieve the host domain from complete website URLs.
* The software makes it possible to present URLs with better organization or to perform quick external link verifications.

**CONCLUSION**

The project is a simple doctor appointment booking app without a database, focusing on collecting and displaying patient details dynamically. It emphasizes Jetpack Compose for UI, state management with remember and mutableStateOf, and navigation using NavController. Data persistence is handled temporarily through a Singleton (AppointmentData). Key features include search functionality, form validation, appointment confirmation, and sharing options via Intent. The project demonstrates effective use of LazyColumn for efficient UI rendering, Inten for external actions (calls, sharing, and website navigation), and debugging with Log.d().

**CODING LIST AS ATTACHMENT**

<https://drive.google.com/file/d/1d42RpnF5evGVKerg-IduN0RyWmg5NmZK/view?usp=drive_link>

**REFERENCES**

Android Developers. (2025). *Navigation with Compose.* [online] Available at:

https://developer.android.com/develop/ui/compose/navigation#kts [Accessed 28 March. 2025].

Kotlin Help. (2025). *Filtering Collections* | Kotlin. [online] Available at:

https://kotlinlang.org/docs/collection-filtering.html.