|  |
| --- |
| Development Scenario 1: Smart Home Controller App Day 1: Android Introduction and Setup Task 1: Install the necessary SDKs and emulators for the Smart Home Controller project. Task 2: Create the initial project structure with a basic activity.  Day 2: Project Structure and UI Design Task 1: Construct the MainActivity layout with controls for smart devices (like lights, thermostats). Task 2: Set up a Gradle configuration for dependency management and build customization.  Day 3: UI Enhancements and Data Binding Task 1: Utilize Data Binding to connect UI components with the backend logic. Task 2: Create a dashboard UI showing the status of various smart devices using data binding.  Day 4: User Interaction, Navigation, and Data Handling Task 1: Implement event handling for user interactions with device controls. Task 2: Develop a settings screen for device preferences and configure navigation to this screen.  Day 5: Networking Task 1: Set up Retrofit to communicate with a backend server managing smart devices. Task 2: Handle server responses, updating the UI accordingly, and manage potential network errors.  Day 6: Coroutines for Asynchronous Tasks Task 1: Use Kotlin coroutines to perform network operations for sending device control commands. Task 2: Apply suspending functions to update device statuses in real-time.  Day 7: WorkManager for Background Tasks Task 1: Implement WorkManager to periodically sync device states with the server. Task 2: Use WorkManager to schedule tasks for device automation based on user-defined rules  Submission Guidelines: 1. Ensure that each answer is clear, concise, and reflects an understanding of the core concepts. 2. Diagrams can be hand-drawn and scanned or created using any digital drawing tool. 3. Provide references for any external sources used. Submit your work in a single PDF document by end of Module. 4. You must submit your code on gitlab by the end of next day. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| **Development Scenario 2: Vehicle Health Monitoring System** **Day 1: Android Introduction and Setup** **Task 1:** Install the necessary SDKs and emulators that simulate vehicle hardware interfaces. **Task 2:** Initialize the Android project with a basic activity that will serve as the dashboard for vehicle health metrics.  **Day 2: Project Structure and UI Design Task 1:** Design the MainActivity layout to display real-time data such as engine health, oil level, and tire pressure. **Task 2:** Configure Gradle for dependency management, focusing on libraries necessary for vehicle data communication.  **Day 3: UI Enhancements and Data Binding Task 1:** Implement Data Binding to connect the dashboard UI components to the backend vehicle data. **Task 2:** Develop a dynamic UI that reflects the vehicle’s health status, using LiveData to ensure real-time updates.  **Day 4: User Interaction, Navigation, and Data Handling Task 1:** Create interactive elements for users to report issues or request maintenance services. **Task 2:** Build a settings screen where users can set thresholds for alerts (like minimum tire pressure).  **Day 5: Networking Task 1:** Establish Retrofit communication with a backend server for storing long-term vehicle health data and scheduling services. **Task 2:** Ensure the network layer gracefully handles errors, with retry mechanisms and user notifications for connectivity issues.  **Day 6: Coroutines for Asynchronous Tasks Task 1:** Use Kotlin coroutines for handling asynchronous communications with onboard diagnostics and sensors. **Task 2:** Implement functions that use coroutines to fetch and push data to the backend without blocking the user interface.  **Day 7: WorkManager for Background Tasks Task 1:** Integrate WorkManager to conduct daily diagnostics and sync this data with the user’s account on the server. **Task 2:** Set up WorkManager to remind users of upcoming maintenance or service appointments based on vehicle usage data and manufacturer recommendations.  **Submission Guidelines:** Ensure clarity and conciseness, demonstrating understanding of Android development and vehicle monitoring concepts. Include diagrams of system architecture and data flow, which can be created digitally or by hand. Cite any external sources used for developing the system. Compile the entire project documentation into a single PDF and submit the code to GitLab by the deadline. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |