Đại học Thái Nguyên
Trường Đại học Công nghệ Thông tin và Truyền thông
Khoa Công nghệ thông tin

GIÁO TRÌNH

LẬP TRÌNH CHO THIẾT BỊ DI ĐỘNG

Chủ biên: **TS. Nguyễn Thế Vịnh**
Đồng chủ biên: **TS. Tô Hữu Nguyên**
Tham gia biên soạn: **ThS. Đoàn Ngọc Phương**
Tham gia biên soạn: **ThS. Nguyễn Thu Phương**
Tham gia biên soạn: **ThS. Nguyễn Thị Dung**

Thái Nguyên – 2026

Đại học Thái Nguyên
Trường Đại học Công nghệ Thông tin và Truyền thông

GIÁO TRÌNH

LẬP TRÌNH CHO THIẾT BỊ DI ĐỘNG

Đơn vị phụ trách: Khoa Công nghệ thông tin
Chủ biên: TS. Nguyễn Thế Vịnh
Đồng chủ biên: TS. Tô Hữu Nguyên
Tham gia biên soạn: ThS. Đoàn Ngọc Phương
Tham gia biên soạn: ThS. Nguyễn Thu Phương
Tham gia biên soạn: ThS. Nguyễn Thị Dung

Thái Nguyên, năm 2026

Thông tin biên soạn

Họ và tên	Vai trò
TS. Nguyễn Thế Vịnh	Chủ biên
TS. Tô Hữu Nguyên	Đồng chủ biên
ThS. Đoàn Ngọc Phương	Tham gia biên soạn
ThS. Nguyễn Thu Phương	Tham gia biên soạn
ThS. Nguyễn Thị Dung	Tham gia biên soạn

Lời nói đầu

Giáo trình Lập trình cho thiết bị di động được biên soạn từ nhu cầu giúp sinh viên tiếp cận phát triển ứng dụng Android theo một mạch công việc hoàn chỉnh. Một ứng dụng không chỉ được tạo nên bởi vài màn hình hoặc những đoạn mã chạy riêng lẻ; phía sau đó còn có cấu trúc dự án, trạng thái, dữ liệu, kết nối mạng và nhiều quyết định kỹ thuật cần được kiểm chứng. Vì vậy, giáo trình đi từ những thao tác đầu tiên với môi trường phát triển đến quá trình hoàn thiện một sản phẩm có thể build, chạy và quan sát trên thiết bị.

Tài liệu hướng tới sinh viên ngành Công nghệ thông tin, Kỹ thuật phần mềm và các học phần liên quan đến phát triển ứng dụng di động. Kiến thức nền về lập trình hướng đối tượng, cấu trúc dữ liệu và cơ sở dữ liệu là một thuận lợi, nhưng những nội dung Kotlin thiết yếu vẫn được giải thích trong chính ngữ cảnh Android.

Chín chương được sắp xếp theo độ phức tạp tăng dần. Sau phần giới thiệu hệ sinh thái Android, Android Studio và cấu trúc dự án, người học lần lượt làm việc với giao diện XML View, Kotlin và Jetpack Compose. Từ một màn hình đơn, nội dung mở rộng sang quản lý trạng thái, tương tác người dùng, danh sách và điều hướng; tiếp đó là lưu trữ cục bộ với Room, vòng đời, ViewModel và kiến trúc MVVM. Phần cuối kết nối ứng dụng với API bằng Retrofit, xử lý trạng thái mạng và quy tụ kiến thức trong mini project. Trong phạm vi học phần, build, chạy, quan sát, kiểm chứng và ca kiểm thử cơ bản gắn với mục tiêu chương là nội dung cốt lõi; kiểm thử tự động nâng cao, AI tác tử, Firebase, WorkManager và mục ghi "Mở rộng" dành cho bổ trợ hoặc tự học, không làm thay đổi chuẩn đầu ra hay thời lượng bắt buộc.

Giáo trình gắn khái niệm với kết quả có thể quan sát và kiểm tra. Mã nguồn, sơ đồ, bảng kiểm và ảnh chụp ứng dụng thật giúp người học lần theo dữ liệu và sự kiện. Sau mỗi thay đổi, kết quả được đối chiếu bằng build, Logcat, kiểm thử hoặc hành vi trên thiết bị; mã đúng cú pháp chưa đủ chứng minh ứng dụng hoạt động đúng.

Giảng viên có thể chọn ví dụ và mức mở rộng theo điều kiện lớp; mini project nối kiến thức giữa các chương. Đường thực hành chính gồm đọc dự án, dựng giao diện, xử lý trạng thái, danh sách, điều hướng, CRUD, ViewModel và dữ liệu mạng. Trường hợp biên có thể giao theo năng lực, nhưng sản phẩm cuối phải được đánh giá bằng mã nguồn và bằng chứng chạy. AI chỉ hỗ trợ đọc lỗi, giải thích hoặc rà soát; quyết định cuối cùng dựa trên tài liệu kỹ thuật và kết quả thực thi.

Giáo trình được hình thành từ giảng dạy, thực hành và kiểm chứng các ví dụ Android. Nền tảng, công cụ còn thay đổi; nhóm tác giả mong nhận góp ý để tài liệu ngày càng phù hợp với đào tạo và thực tiễn phát triển ứng dụng di động.

Thái Nguyên, 2026

TS. Nguyễn Thế Vịnh; TS. Tô Hữu Nguyên; ThS. Đoàn Ngọc Phương; ThS. Nguyễn Thu Phương; ThS. Nguyễn Thị Dung

MỤC LỤC

Thông tin biên soạn i

Lời nói đầu ii

Danh mục hình ảnh vii

Danh mục bảng biểu x

Danh mục từ viết tắt xii

I Nền tảng Android và Kotlin 1

1 Tổng quan về phát triển ứng dụng trên thiết bị di động 1
Nội dung chính . . . 1
Mục tiêu cần đạt . . . 1
1.1 Tổng quan về ứng dụng di động . . . 2
1.2 Đặc điểm của phát triển ứng dụng di động . . . 5
1.3 Nền tảng Android và hệ sinh thái phát triển . . . 8
1.4 Môi trường và công cụ phát triển ứng dụng Android . . . 10
1.5 Giới thiệu ngôn ngữ Kotlin trong Android . . . 21
1.6 Kiểm thử dự án Android đầu tiên . . . 23
1.7 Sử dụng AI tác tử trong thực hành Android đầu tiên . . . 26
Tổng kết . . . 27
Câu hỏi ôn tập . . . 27
Bài tập thực hành . . . 28

2 Cơ sở xây dựng giao diện Android theo mô hình View (XML) 29
Nội dung chính . . . 29
Mục tiêu cần đạt . . . 29
2.1 Mô hình View và ViewGroup trong Android . . . 30
2.2 Các bố cục cơ bản trong Android . . . 35
2.3 Các thành phần giao diện người dùng cơ bản . . . 42
2.4 Liên kết giao diện và xử lý sự kiện cơ bản . . . 46
2.5 Đánh giá mô hình xây dựng giao diện bằng XML . . . 49
2.6 Kiểm thử giao diện XML bằng Espresso . . . 51
2.7 Sử dụng AI tác tử để rà soát giao diện XML . . . 53
Tổng kết . . . 54

Câu hỏi ôn tập . . . 54
Bài tập thực hành . . . 55

3 Lập trình Kotlin ứng dụng trong Android 56
Nội dung chính . . . 56
Mục tiêu cần đạt . . . 56
3.1 Lập trình hướng đối tượng với Kotlin trong Android . . . 57
3.2 Lập trình hàm và biểu thức lambda trong Kotlin . . . 68
3.3 Xử lý dữ liệu và bộ sưu tập trong Kotlin . . . 71
3.4 Kiểm thử logic Kotlin bằng kiểm thử đơn vị cục bộ . . . 78
3.5 Sử dụng AI tác tử để rà soát mã Kotlin . . . 81
Tổng kết . . . 84
Câu hỏi ôn tập . . . 84
Bài tập thực hành . . . 84

II Phát triển giao diện và tương tác 85

4 Jetpack Compose và giao diện khai báo 85
Nội dung chính . . . 85
Mục tiêu cần đạt . . . 85
4.1 Tổng quan về Jetpack Compose và tư duy giao diện khai báo . . . 86
4.2 Hàm Composable và nguyên tắc xây dựng giao diện . . . 90
4.3 Các thành phần giao diện cơ bản trong Jetpack Compose . . . 92
4.4 Modifier và bố cục giao diện trong Compose . . . 98
4.5 Kiểm thử giao diện Jetpack Compose bằng cây ngữ nghĩa . . . 104
4.6 Sử dụng AI tác tử để rà soát màn hình Compose . . . 107
Tổng kết . . . 110
Câu hỏi ôn tập . . . 110
Bài tập thực hành . . . 111

5 Quản lý trạng thái và tương tác người dùng trong Compose 112
Nội dung chính . . . 112
Mục tiêu cần đạt . . . 112
5.1 Trạng thái giao diện và cơ chế recomposition . . . 113
5.2 Quản lý trạng thái trong Jetpack Compose . . . 119
5.3 Xử lý tương tác người dùng và phân tách logic . . . 124
5.4 Kiểm chứng luồng sự kiện–trạng thái–UI bằng kiểm thử Compose . . . 133
5.5 Sử dụng AI tác tử để rà soát trạng thái, sự kiện và recomposition . . . 135
Tổng kết . . . 137
Câu hỏi ôn tập . . . 137
Bài tập thực hành . . . 138

6 Hiển thị danh sách và điều hướng ứng dụng 139
Nội dung chính . . . 139
Mục tiêu cần đạt . . . 139
6.1 Hiển thị danh sách dữ liệu trong Jetpack Compose . . . 140

6.2 Điều hướng giữa các màn hình trong ứng dụng Android . . . 148
6.3 Truyền dữ liệu và quản lý luồng điều hướng . . . 154
6.4 Kiểm thử danh sách và luồng điều hướng . . . 158
6.5 Sử dụng AI tác tử để kiểm tra luồng màn hình và trường hợp biên . . . 163
Tổng kết . . . 164
Câu hỏi ôn tập . . . 164
Bài tập thực hành . . . 165

III Dữ liệu, kiến trúc và ứng dụng hoàn chỉnh 166

7 Lưu trữ và quản lý dữ liệu trong ứng dụng Android 166
Nội dung chính . . . 166
Mục tiêu cần đạt . . . 166
7.1 Lưu trữ dữ liệu cục bộ dạng đơn giản . . . 167
7.2 Cơ sở dữ liệu quan hệ trong Android . . . 171
7.3 Thao tác và truy xuất dữ liệu với Room . . . 173
7.4 Kết hợp quản lý dữ liệu với giao diện Compose . . . 181
7.5 Kiểm thử dữ liệu và DAO của Room . . . 189
7.6 Sử dụng AI tác tử để rà soát mô hình dữ liệu . . . 190
Tổng kết . . . 190
Câu hỏi ôn tập . . . 191
Bài tập thực hành . . . 192

8 Vòng đời và kiến trúc ứng dụng Android 193
Nội dung chính . . . 193
Mục tiêu cần đạt . . . 193
8.1 Các thành phần cơ bản của ứng dụng Android . . . 193
8.2 Vòng đời Activity và quản lý trạng thái ứng dụng . . . 194
8.3 Kiến trúc ứng dụng Android theo mô hình MVVM . . . 199
8.4 Quản lý trạng thái giao diện với ViewModel và Jetpack Compose . . . 207
8.5 Kiểm thử trạng thái giao diện và ViewModel . . . 215
8.6 Sử dụng AI tác tử để phân tích lỗi vòng đời và kiến trúc . . . 216
Tổng kết . . . 218
Câu hỏi ôn tập . . . 219
Bài tập thực hành . . . 219

9 Kết nối mạng và phát triển ứng dụng hoàn chỉnh 221
Nội dung chính . . . 221
Mục tiêu cần đạt . . . 221
9.1 Tổng quan kết nối mạng trong ứng dụng di động . . . 221
9.2 Kết nối và trao đổi dữ liệu với Retrofit . . . 225
9.3 Hiển thị và xử lý dữ liệu mạng trong Compose . . . 230
9.4 Phát triển và hoàn thiện ứng dụng Android . . . 235
9.5 Kiểm thử ranh giới mạng và quá trình đồng bộ . . . 244
9.6 Sử dụng AI tác tử để rà soát dự án nhỏ và kế hoạch kiểm chứng . . . 245
Tổng kết . . . 246

Câu hỏi ôn tập 246
Bài tập thực hành 247

Kết luận và định hướng mở rộng **248**

Tài liệu tham khảo **249**

DANH MỤC HÌNH ẢNH

Hình 1.1. Ứng dụng di động trong các ngữ cảnh hằng ngày 2
Hình 1.2. Ràng buộc tài nguyên của thiết bị di động 6
Hình 1.3. Vòng đời Activity theo tài liệu Android Developers 7
Hình 1.4. Các lớp nền tảng Android ở mức nhập môn 9
Hình 1.5. Quy trình cài đặt Android Studio 10
Hình 1.6. Hai bước đầu trong Android Studio Setup Wizard 11
Hình 1.7. Bảng kiểm môi trường sau khi cài Android Studio 11
Hình 1.8. Chuỗi công cụ từ Android Studio đến thiết bị chạy thử 13
Hình 1.9. Ba điểm chẩn đoán chuỗi build–cài đặt–khởi chạy 13
Hình 1.10. Ba bước cấu hình thiết bị ảo trong Device Manager (nguồn ảnh giao diện: Android Developers) 14
Hình 1.11. Từ tùy chọn nhà phát triển đến kết nối gỡ lỗi bằng USB hoặc Wi-Fi (phỏng theo Android Developers) 15
Hình 1.12. Từ màn hình chào đến lựa chọn mẫu dự án 16
Hình 1.13. Các trường cấu hình dự án mới 17
Hình 1.14. Tệp MainActivity của dự án Compose tối giản vừa được tạo . 17
Hình 1.15. Các vùng chính của dự án trong chế độ Android 19
Hình 1.16. Chuỗi thao tác khi bấm Run 20
Hình 1.17. Màn hình Hello World trên trình giả lập 21
Hình 1.18. Panel Tác tử của Gemini trước khi đăng nhập 27

Hình 2.1. Kiến trúc màn hình XML/View 30
Hình 2.2. Màn hình hồ sơ và vùng biểu mẫu ở trạng thái ban đầu 31
Hình 2.3. LayoutParams của View được xác định theo ViewGroup cha (nguồn hình: Android Developers) 32
Hình 2.4. Luồng từ XML đến giao diện trên trình giả lập 35
Hình 2.5. Bốn ràng buộc View với phần tử cha 39
Hình 2.6. Guideline, Barrier và các kiểu Chain trong ConstraintLayout (nguồn hình: Android Developers) 40
Hình 2.7. Kiểm tra bố cục trên nhiều cấu hình trình giả lập 41
Hình 2.8. Ba trạng thái kiểm chứng của biểu mẫu 44
Hình 2.9. Luồng từ cấu trúc XML đến phản hồi quan sát được trên ứng dụng 47

Hình 3.1. Luồng dữ liệu trong ví dụ minh họa Kotlin 57
Hình 3.2. Bốn nhánh kiểm tra Kotlin 63

Hình 4.1. Màn hình hồ sơ Compose trước và sau khi lưu hợp lệ 87
Hình 4.2. Luồng dữ liệu một chiều giữa trạng thái và giao diện 87

Hình 4.3. Ba trạng thái kiểm tra: thiếu họ tên, thiếu mã và mã sai định dạng . . . 95
Hình 4.4. Đối chiếu Modifier với giao diện Compose chạy thật . . . 101

Hình 5.1. Trạng thái quyết định UI; sự kiện yêu cầu cập nhật trạng thái . 113
Hình 5.2. Pha bắt đầu lại phụ thuộc nơi trạng thái được đọc . . . 116
Hình 5.3. Một lượt tương tác cập nhật nhiều trạng thái . . . 116
Hình 5.4. Hai trạng thái đầu vào được cập nhật độc lập . . . 121
Hình 5.5. Hai trường hợp `canSave=false` . . . 128
Hình 5.6. Đối chiếu kiểm tra nghiệp vụ và trạng thái phái sinh . . . 132

Hình 6.1. Danh sách hồ sơ bằng LazyColumn . . . 140
Hình 6.2. Hướng trình bày của LazyColumn, LazyRow và thành phần lưới lazy . . . 142
Hình 6.3. Bộ lọc Trực tuyến: hai hồ sơ . . . 143
Hình 6.4. Đối chiếu kết quả của hai bộ lọc . . . 146
Hình 6.5. Điều hướng và ngăn xếp quay lại . . . 148
Hình 6.6. Cùng đích điều hướng với hai `profileId` hợp lệ . . . 150
Hình 6.7. Các đích điều hướng thực tế trong `Chapter6App` . . . 152
Hình 6.8. Luồng từ danh sách đến màn hình chi tiết . . . 152
Hình 6.9. Lỗi khi `profileId` không tồn tại . . . 154
Hình 6.10. Ba trạng thái biên của cùng màn hình danh sách . . . 156

Hình 7.1. Hai kết quả hợp lệ của truy vấn danh sách Room . . . 167
Hình 7.2. Màn hình kiểm tra dữ liệu Room trên SQLite . . . 172
Hình 7.3. Insert một ghi chú . . . 179
Hình 7.4. Hai thao tác thay đổi bản ghi đã có . . . 180
Hình 7.5. Hai nhánh lọc theo trạng thái hoàn thành . . . 184
Hình 7.6. Truy vấn chi tiết theo khóa chính . . . 186
Hình 7.7. Ba bản ghi được đọc lại sau khi mở ứng dụng . . . 186
Hình 7.8. Ba kịch bản CRUD độc lập và phép thử mở lại sau khi xóa . . . 187

Hình 8.1. Vòng đời Activity theo tài liệu Android Developers . . . 196
Hình 8.2. Bốn trạng thái giao diện của cùng một màn hình . . . 199
Hình 8.3. Các tầng kiến trúc ứng dụng Android . . . 200
Hình 8.4. Sự kiện đi lên, trạng thái giao diện đi xuống . . . 201
Hình 8.5. Xử lý sự kiện AddTask trong ViewModel . . . 204
Hình 8.6. Xử lý sự kiện ToggleTask trong ViewModel . . . 205
Hình 8.7. Lọc danh sách DONE qua trạng thái giao diện . . . 210
Hình 8.8. Hai trạng thái phục vụ kiểm chứng thử lại và vòng đời . . . 211
Hình 8.9. Quy trình thu thập bằng chứng vòng đời và thử lại . . . 214

Hình 9.1. Luồng Android máy khách đến REST API . . . 222
Hình 9.2. Danh sách dữ liệu nhận từ API lớp học . . . 223
Hình 9.3. Chi tiết một bài viết được ánh xạ thành PostDto . . . 225
Hình 9.4. Quy trình chẩn đoán lỗi HTTP . . . 227
Hình 9.5. Trạng thái đang tải dùng để quan sát thời gian chờ API . . . 229
Hình 9.6. Luồng trạng thái và dữ liệu dự phòng khi ngoại tuyến . . . 231

Hình 9.7. Hai trạng thái dùng để đối chiếu giao diện lỗi và thử lại 233
Hình 9.8. Trạng thái dự phòng ngoại tuyến bằng dữ liệu mẫu trong bộ nhớ 234
Hình 9.9. Trạng thái danh mục sau bước tải API và ghi Room cần kiểm chứng 236
Hình 9.10. Màn hình chi tiết nhận đúng sản phẩm từ tuyến điều hướng . . 237
Hình 9.11. Giỏ hàng và đơn hàng trong luồng dữ liệu cục bộ 239
Hình 9.12. Ba kết quả cần phân biệt trong luồng mạng và đồng bộ 241

DANH MỤC BẢNG BIỂU

Bảng 1.1. Khác biệt tư duy khi chuyển sang lập trình di động 3
Bảng 1.2. So sánh nhập môn Android và iOS 4
Bảng 1.3. So sánh các hướng phát triển ứng dụng di động 4
Bảng 1.4. Từ ràng buộc thiết bị đến quyết định thiết kế 5
Bảng 1.5. Từ tương tác di động đến thiết kế giao diện 7
Bảng 1.6. Vai trò các thành phần trong hệ sinh thái Android 9
Bảng 1.7. Bộ phiên bản đã kiểm chứng với giáo trình 12
Bảng 1.8. Các thành phần của dự án Android nhập môn 19

Bảng 2.1. Từ XML đến giao diện chạy thực tế 33
Bảng 2.2. Các tệp của màn hình XML 34
Bảng 2.3. Các kích thước thường dùng trong XML 38
Bảng 2.4. Các điều kiện kiểm tra bố cục 41
Bảng 2.5. So sánh LinearLayout và ConstraintLayout 42
Bảng 2.6. Ma trận trạng thái biểu mẫu hồ sơ 45
Bảng 2.7. Luồng xử lý khi bấm Lưu hồ sơ 48
Bảng 2.8. Các lỗi thường gặp với giao diện XML 49
Bảng 2.9. Ánh xạ thao tác người dùng sang Espresso 51
Bảng 2.10. Nhật ký quyết định cho đề xuất AI 54

Bảng 3.1. Thuộc tính và phương thức của `StudentProfile` 59
Bảng 3.2. Trách nhiệm các phần mã trong ví dụ minh họa Kotlin 60
Bảng 3.3. Khả năng được sinh bởi `data class` 61
Bảng 3.4. Khi dùng `data class`, `data object` và `sealed interface` 62
Bảng 3.5. Luồng `validateProfile` theo dữ liệu đầu vào 64
Bảng 3.6. Kỹ thuật xử lý null an toàn trong ví dụ minh họa Kotlin 67
Bảng 3.7. Hướng đối tượng và lập trình hàm trong Chương 3 69
Bảng 3.8. Chuỗi xử lý tập hợp của hồ sơ sinh viên 71
Bảng 3.9. Các nhóm tập hợp trong Kotlin 71
Bảng 3.10. `List` và `MutableList` trong luồng dữ liệu 72
Bảng 3.11. Thao tác thêm, xóa và cập nhật danh sách 73
Bảng 3.12. Luồng xử lý bài toán hồ sơ sinh viên 75
Bảng 3.13. Bằng chứng kiểm tra bài toán tích hợp 77
Bảng 3.14. Bốn hành vi Kotlin cần kiểm thử 78
Bảng 3.15. Đánh giá đề xuất AI với mã Kotlin 82

Bảng 4.1. So sánh XML/View và Jetpack Compose 86
Bảng 4.2. Trách nhiệm các Composable của màn hình hồ sơ 91

Bảng 4.3. Các vai trò của `Modifier` . . . 99
Bảng 4.4. Nhầm lẫn nhập môn khi dùng `Modifier` . . . 100
Bảng 4.5. Ánh xạ hành vi sang Kiểm thử Compose . . . 105
Bảng 4.6. Kiểm chứng đề xuất AI với Compose . . . 109

Bảng 5.1. Câu hỏi nhận diện và đặt phạm vi trạng thái . . . 114
Bảng 5.2. Biến thường và biến trạng thái trong Compose . . . 120
Bảng 5.3. Ánh xạ dữ liệu đầu vào của màn hình hồ sơ . . . 122
Bảng 5.4. Hợp đồng cập nhật trạng thái cho các sự kiện chính . . . 126
Bảng 5.5. Trạng thái gốc và trạng thái phái sinh . . . 131
Bảng 5.6. Bằng chứng cần có khi đánh giá đề xuất rà soát trạng thái . . . 136

Bảng 6.1. Rủi ro khi thiếu key ổn định . . . 146
Bảng 6.2. Ba thành phần của Navigation Compose . . . 150
Bảng 6.3. Tuyến điều hướng chuỗi và tuyến điều hướng an toàn kiểu dữ liệu . . . 153
Bảng 6.4. Phạm vi của kiểm thử cục bộ và kiểm thử giao diện . . . 160

Bảng 7.1. Khi nào dùng lưu trữ khóa–giá trị hoặc Room . . . 170
Bảng 7.2. Kiến thức SQLite nền tảng cho Room . . . 172
Bảng 7.3. Kiểm chứng các thao tác CRUD . . . 179
Bảng 7.4. Trách nhiệm các lớp Room–Compose . . . 183

Bảng 8.1. Các thành phần cơ bản của ứng dụng Android . . . 194
Bảng 8.2. Các hàm gọi lại vòng đời Activity . . . 195
Bảng 8.3. Chọn nơi giữ trạng thái theo vòng đời . . . 197
Bảng 8.4. Luồng dữ liệu trong MVVM . . . 201
Bảng 8.5. Mô hình dữ liệu, trạng thái giao diện và sự kiện UI . . . 203
Bảng 8.6. Phân định trách nhiệm xử lý lỗi . . . 206
Bảng 8.7. Ranh giới ViewModel–Composable . . . 210
Bảng 8.8. Phân loại trạng thái trong màn hình Android . . . 212

Bảng 9.1. Các thành phần cơ bản của hợp đồng API . . . 223
Bảng 9.2. Phân biệt JSON, DTO và dữ liệu trong ứng dụng . . . 224
Bảng 9.3. Lỗi cấu hình Retrofit thường gặp . . . 227
Bảng 9.4. Cách xử lý một số lỗi mạng thường gặp . . . 228
Bảng 9.5. Ranh giới trách nhiệm gọi API . . . 229
Bảng 9.6. Bảng kiểm gọi mạng bất đồng bộ . . . 232
Bảng 9.7. Lỗi, rỗng và dữ liệu dự phòng khi ngoại tuyến . . . 234
Bảng 9.8. Luồng dữ liệu chính trong ICTU Market . . . 237
Bảng 9.9. Vai trò của Room, REST API và Cloud Firestore trong ICTU Market . . . 243

Danh mục từ viết tắt

Từ viết tắt	Cụm từ đầy đủ	Diễn giải ngắn
AAB	Android App Bundle	Định dạng đóng gói ứng dụng dùng khi phát hành qua Google Play.
ADB	Android Debug Bridge	Công cụ kết nối máy tính với thiết bị Android để cài đặt, chạy và gỡ lỗi.
AI	Artificial Intelligence	Trí tuệ nhân tạo; được dùng để hỗ trợ giải thích, rà soát và kiểm chứng.
API	Application Programming Interface	Giao diện cho phép ứng dụng trao đổi dữ liệu hoặc sử dụng chức năng của hệ thống khác.
APK	Android Package Kit	Gói cài đặt ứng dụng Android dùng khi chạy thử hoặc phân phối trực tiếp.
AVD	Android Virtual Device	Cấu hình thiết bị Android ảo dùng với trình giả lập.
BOM	Bill of Materials	Bảng quản lý một nhóm phiên bản thư viện tương thích, thường dùng với Compose.
CRUD	Create, Read, Update, Delete	Bốn thao tác cơ bản với dữ liệu: thêm, đọc, sửa và xóa.
DAO	Data Access Object	Đối tượng khai báo các thao tác truy cập dữ liệu trong Room.
DSL	Domain-Specific Language	Ngôn ngữ chuyên biệt cho một miền, như Kotlin DSL trong cấu hình Gradle.
DTO	Data Transfer Object	Đối tượng truyền dữ liệu, thường ánh xạ từ JSON nhận qua API.
HTTP	Hypertext Transfer Protocol	Giao thức truyền dữ liệu nền tảng cho các lời gọi web và API.
HTTPS	Hypertext Transfer Protocol Secure	Phiên bản HTTP được bảo vệ bằng cơ chế mã hóa và xác thực.

Từ viết tắt	Cụm từ đầy đủ	Diễn giải ngắn
ICTU	Information and Communication Technology University	Tên viết tắt tiếng Anh của Trường Đại học Công nghệ Thông tin và Truyền thông.
ID	Identifier	Giá trị định danh View, bản ghi, đối tượng hoặc tham số điều hướng.
IDE	Integrated Development Environment	Môi trường phát triển tích hợp; trong giáo trình chủ yếu là Android Studio.
JDK	Java Development Kit	Bộ công cụ Java được Android Studio và Gradle sử dụng khi build dự án.
JSON	JavaScript Object Notation	Định dạng văn bản phổ biến để trao đổi dữ liệu qua API.
JVM	Java Virtual Machine	Máy ảo Java, môi trường chạy của nhiều kiểm thử Kotlin cục bộ.
KSP	Kotlin Symbol Processing	Cơ chế xử lý ký hiệu Kotlin để sinh mã cho Room và các thư viện khác.
LLM	Large Language Model	Mô hình ngôn ngữ lớn dùng trong các dịch vụ AI tạo sinh.
MVVM	Model–View–ViewModel	Mô hình kiến trúc tách dữ liệu, giao diện và logic trình bày.
OOP	Object-Oriented Programming	Lập trình hướng đối tượng, nền tảng tổ chức lớp và đối tượng trong Kotlin.
RAM	Random Access Memory	Bộ nhớ truy cập ngẫu nhiên, ảnh hưởng đến Android Studio và trình giả lập.
REST	Representational State Transfer	Phong cách thiết kế API dựa trên tài nguyên và các phương thức HTTP.
SDK	Software Development Kit	Bộ công cụ, thư viện và API phục vụ phát triển ứng dụng.
SQL	Structured Query Language	Ngôn ngữ truy vấn dữ liệu quan hệ, được dùng với SQLite và Room.
UI	User Interface	Giao diện người dùng và các trạng thái hiển thị của ứng dụng.
URL	Uniform Resource Locator	Địa chỉ xác định tài nguyên hoặc điểm truy cập trên mạng.
UX	User Experience	Trải nghiệm người dùng khi tương tác và nhận phản hồi từ ứng dụng.
XML	Extensible Markup Language	Ngôn ngữ đánh dấu dùng cho giao diện View và tài nguyên Android truyền thống.

Phần I

Nền tảng Android và Kotlin

CHƯƠNG 1

Tổng quan về phát triển ứng dụng trên thiết bị di động

Nội dung chính

Trước hết, chương dẫn người học đi trọn một hành trình với ứng dụng Android nhỏ: nhận ra khác biệt của môi trường di động, chuẩn bị Android Studio và thiết bị chạy thử, mở đúng dự án, rồi build (biên dịch và đóng gói ứng dụng), cài đặt, chạy và đối chiếu kết quả. Kotlin được giới thiệu vừa đủ để lần từ một giá trị trong mã nguồn đến dòng chữ trên màn hình. Kết thúc mạch này, người học cần tự thực hiện lại các bước và giải thích ứng dụng đã thành công đến đâu bằng bằng chứng từ công cụ hoặc thiết bị.

Khi đã theo được mạch trên, phần kiểm thử và trợ lý AI giúp người học nhìn xa hơn lần chạy đầu tiên. Hai nội dung tập trung vào cách chọn một quy tắc nhỏ, đặt câu hỏi và kiểm chứng đề xuất bằng kết quả thực tế; chương chưa yêu cầu xây dựng hệ thống kiểm thử đầy đủ hoặc dựa vào AI để hoàn thành bài. Các chương sau tiếp tục cách học tạo–chạy–quan sát khi đi sâu vào giao diện, dữ liệu, nhiều màn hình, lưu trữ và kết nối mạng.

Mục tiêu cần đạt

Sau khi học xong chương, sinh viên có thể:

- Phân biệt ứng dụng di động với ứng dụng máy tính hoặc web; nêu được các ràng buộc cơ bản của thiết bị.
- Trình bày vai trò của Android, Kotlin và Jetpack Compose; nhận diện mô-đun ứng dụng, mã Kotlin, tài nguyên, manifest và cấu hình build.
- Thiết lập môi trường đã kiểm chứng và chọn thiết bị chạy thử phù hợp.
- Mở, build, cài đặt, chạy và chỉnh sửa nhỏ dự án; thu thập bằng chứng từ build, ADB, Logcat hoặc màn hình ứng dụng.
- Phân biệt kiểm thử đơn vị với kiểm tra chạy; sử dụng AI hỗ trợ đọc lỗi và lập kế hoạch kiểm chứng.

1.1. Tổng quan về ứng dụng di động

1.1.1. Khái niệm ứng dụng di động

Khi mới học lập trình, sinh viên thường làm việc với chương trình chạy liên tục trên máy tính. Ứng dụng di động lại được dùng trong những khoảng thời gian ngắn và có thể bị gián đoạn bởi cuộc gọi, thông báo, thay đổi mạng hoặc xoay màn hình. Trong học tập, mua sắm, di chuyển hay công việc, điện thoại thường là nơi người dùng hoàn tất một việc cụ thể; vì vậy, ứng dụng không chỉ cần làm đúng chức năng mà còn phải phản hồi rõ, giữ được công việc đang dở và thích ứng với hoàn cảnh sử dụng.



Hình 1.1. *Ứng dụng di động trong các ngữ cảnh hằng ngày*

Hình 1.1 đặt điện thoại vào các bối cảnh quen thuộc. Khi phân tích một tình huống, cần bắt đầu từ việc người dùng muốn hoàn tất, rồi mới xác định dữ liệu và khả năng thiết bị cần dùng. Chẳng hạn, xác nhận điểm đến cần vị trí, mạng và phản hồi khi người dùng chưa cấp quyền; chỉ hiển thị được bản đồ là chưa đủ. Ngoài câu hỏi "chức năng có chạy không", người phát triển còn phải xem khi nào cần xin quyền, điều gì xảy ra khi một điều kiện không còn sẵn sàng và dữ liệu có được giữ khi người dùng tạm rời ứng dụng hay không. Bảng 1.1 đối chiếu những khác biệt khiến việc chạy thử trên thiết bị và lưu bằng chứng cần được thực hiện ngay từ dự án đầu tiên.

Ba câu hỏi trong bảng sẽ được dùng xuyên suốt giáo trình: dữ liệu có đúng không, giao diện có phản ánh trạng thái hiện tại không và ứng dụng có tiếp tục hoạt động hợp lý khi điều kiện thay đổi không. Vì vậy, chạy thử trên thiết bị là một phần của quá trình phát triển, không chỉ là thao tác cuối. Chẳng hạn, màn hình tra lịch học phải cho biết tuần đang xem, trạng thái đang xử lý hoặc lỗi, đồng thời cho phép người dùng tiếp tục sau khi tạm rời màn hình.

1.1.2. Vai trò của ứng dụng di động

Ứng dụng di động là điểm nối giữa người dùng với một dịch vụ trong bối cảnh cụ thể. Ứng dụng nhận thao tác, đưa thông tin hoặc chức năng cần thiết lên màn hình, rồi phản hồi khi công việc đang được xử lý, đã hoàn tất hoặc gặp lỗi. Chẳng hạn, ứng dụng tra lịch học giúp sinh viên tìm đúng lớp và thời gian; ứng dụng giao hàng giúp nhân viên xác nhận điểm nhận hoặc giao. Vai trò ấy không chỉ là hiển thị thông tin: khi người dùng

Bảng 1.1. *Khác biệt tư duy khi chuyển sang lập trình di động*

Khía cạnh	**Trong cách nghĩ máy tính để bàn/dòng lệnh**	**Trong ứng dụng di động**
Phiên sử dụng	Người dùng làm việc tương đối liên tục	Người dùng thao tác ngắn, thường xuyên bị gián đoạn
Thiết bị nhập liệu	Bàn phím, chuột, màn hình lớn	Cảm ứng, bàn phím ảo, màn hình nhỏ, thao tác một tay
Kết nối mạng	Thường giả định ổn định hơn	Có thể chuyển Wi-Fi/4G/5G, mất sóng, chậm hoặc bị giới hạn dữ liệu
Tài nguyên	Nguồn điện và bộ nhớ tương đối rộng	Pin, bộ nhớ và tác vụ nền bị hệ điều hành kiểm soát chặt
Quyền truy cập	Ít gắn với cảm biến cá nhân	Camera, vị trí, micro, ảnh, thông báo và dữ liệu cá nhân cần xin quyền rõ ràng

đang di chuyển, mạng yếu hoặc bị gián đoạn, ứng dụng còn phải cho biết điều gì đang xảy ra, giữ phần việc cần thiết và cho phép tiếp tục thay vì bắt đầu lại.

Trong Chương 1, `MobileProgrammingDemo` là ví dụ nhỏ để người học quan sát chuỗi này trên một màn hình lời chào: mở đúng tệp, chạy ứng dụng và kiểm tra kết quả. Màn hình Android được tổ chức trong một thành phần gọi là Activity; tên kỹ thuật này sẽ được làm rõ khi người học mở dự án. Do kho mã đồng hành còn chứa ví dụ của các chương sau, cần chọn `Chapter1FirstAppActivity.kt` thay vì mặc định `MainActivity` luôn là màn hình đầu tiên. Từ ví dụ đó, mỗi khái niệm sẽ được gắn với một vị trí trong dự án, một thao tác và kết quả có thể quan sát.

1.1.3. Các nền tảng di động phổ biến

Khi nói về hướng phát triển ứng dụng di động, cần trả lời hai câu hỏi: ứng dụng sẽ chạy trên hệ điều hành nào và phần nào của mã nguồn được dùng lại giữa các hệ điều hành. Có hướng viết riêng cho từng nền tảng để làm việc trực tiếp với công cụ của nền tảng đó; cũng có hướng chia sẻ một phần hoặc phần lớn mã nguồn để phục vụ nhiều nền tảng. Theo cách phân biệt này, Android thuần và iOS thuần thuộc nhóm thứ nhất, còn Flutter, React Native và Kotlin Multiplatform chia sẻ mã ở những mức độ khác nhau. Không có hướng nào luôn tốt hơn; giáo trình chọn Android thuần để người học nhìn rõ một ứng dụng được tạo, build, cài đặt và chạy như thế nào trước khi đánh giá các lựa chọn khác.

Android thuần và iOS thuần làm việc trực tiếp với công cụ và quy ước của từng hệ điều hành, nên khác nhau không chỉ ở ngôn ngữ mà còn ở cách tạo ứng dụng, thiết bị dùng để chạy thử và quy trình phát hành. Học vững một nền tảng trước giúp người học đánh giá thực tế hơn phần mã nào có thể chia sẻ và phần nào vẫn phụ thuộc thiết bị. Các hướng đa nền tảng có thể giảm công việc lặp lại, nhưng vẫn phải xử lý những khác biệt về quyền truy cập, vòng đời và hành vi khi chạy trên từng hệ điều hành.

Bảng 1.2. *So sánh nhập môn Android và iOS*

Khía cạnh	Android	iOS
Hệ sinh thái thiết bị	Nhiều nhà sản xuất, nhiều kích thước màn hình và cấu hình phần cứng	Chủ yếu trên thiết bị Apple, mức đồng bộ phần cứng/phần mềm cao
Công cụ phát triển	Android Studio, Android SDK, Gradle, trình giả lập, thiết bị Android thật	Xcode, iOS SDK, Simulator, thiết bị Apple thật
Ngôn ngữ phổ biến	Kotlin, Java	Swift, Objective-C
Phân phối ứng dụng	Google Play và các kênh phân phối Android khác	`App Store` với quy trình kiểm duyệt tập trung
Ý nghĩa trong học phần	Phù hợp để học nền tảng mở, dễ thử nghiệm trên nhiều thiết bị và trình giả lập	Có giá trị so sánh, nhưng không phải trục triển khai chính của giáo trình

Bảng 1.3. *So sánh các hướng phát triển ứng dụng di động*

Hướng tiếp cận	Phù hợp khi	Điểm cần thận trọng
Android thuần	Cần hiểu sâu Android, dùng API nền tảng, học theo SDK chính thức	Chỉ giải quyết trực tiếp phía Android
iOS thuần	Cần tối ưu sâu cho thiết bị Apple	Cần hệ sinh thái công cụ và ngôn ngữ riêng
Flutter	Muốn UI nhất quán và phát hành đa nền tảng nhanh	Cần học Dart và hiểu cách tích hợp thuần khi cần
React Native	Nhóm có nền tảng JavaScript/TypeScript và muốn chia sẻ nhiều mã	Vẫn phải xử lý cầu nối và mô-đun thuần và hiệu năng ở bài toán phức tạp
Kotlin Multiplatform	Muốn chia sẻ logic nghiệp vụ, vẫn giữ UI thuần	Đòi hỏi thiết kế mô-đun rõ và hiểu nhiều nền tảng
Di động web	Muốn triển khai nhanh qua trình duyệt và không cần cài ứng dụng	Khó khai thác sâu tính năng thiết bị và trải nghiệm không hoàn toàn như ứng dụng thuần
Ứng dụng lai (hybrid)	Muốn đóng gói nội dung web trong lớp vỏ ứng dụng	Dễ bắt đầu nhưng cần thận trọng về hiệu năng, tích hợp thuần và trải nghiệm giao diện

Hai bảng trong mục này chỉ giúp nhận diện bức tranh chung, chưa yêu cầu chọn công nghệ cho một sản phẩm thật. Dù dùng hướng nào, tiêu chí cuối cùng vẫn là người dùng có hoàn tất nhiệm vụ trong điều kiện thực tế hay không: giao diện có thích ứng, dữ liệu có được giữ hợp lý, lỗi có được giải thích và ứng dụng có tiếp tục phản hồi trên thiết bị mục tiêu hay không. Trước khi so sánh công cụ, cần nêu cùng một nhiệm vụ, thiết bị đích và kết quả phải quan sát. Trong bài học này, kết quả tối thiểu là dự án build được,

cài được và mở đúng màn hình trên trình giả lập Android; từ điểm xuất phát cụ thể đó, người học mới có cơ sở để đọc tài liệu hoặc đánh giá một hướng phát triển khác.

1.2. Đặc điểm của phát triển ứng dụng di động

1.2.1. Hạn chế về tài nguyên phần cứng

Thiết bị di động có những giới hạn quen thuộc: màn hình nhỏ ảnh hưởng cách trình bày; cảm ứng cần vùng chạm rõ; pin và bộ nhớ giới hạn công việc nền; một số chức năng chỉ dùng được khi người dùng cấp quyền. Trước khi viết mã, cần biết ứng dụng đang dùng tài nguyên nào và phản hồi ra sao khi điều kiện chưa đáp ứng.

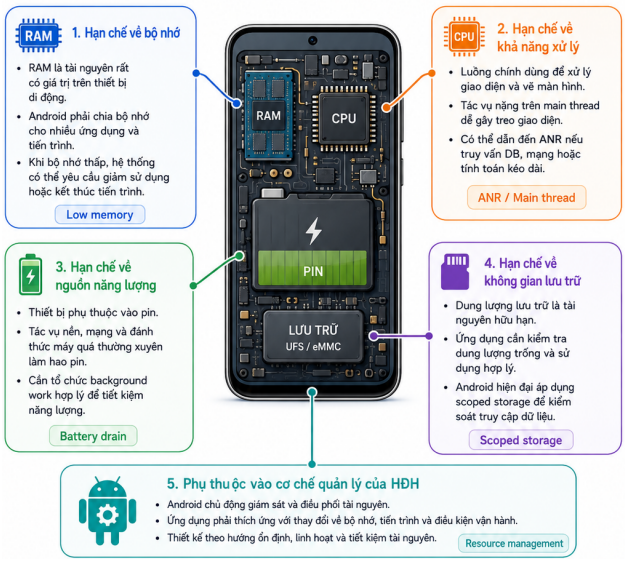
Các giới hạn này thường xuất hiện cùng lúc. Khi tải danh sách trên mạng yếu, giao diện phải cho biết đang tải hay gặp lỗi; công việc nặng không được làm màn hình mất phản hồi; dữ liệu vừa có không nên biến mất khi người dùng tạm rời ứng dụng. Để tránh ghi nhớ từng giới hạn như một ý rời, Bảng 1.4 nối mỗi điều kiện với hậu quả có thể quan sát và hướng xử lý tương ứng.

Không cần dự đoán hết mọi lỗi ngay khi bắt đầu dự án. Với mỗi chức năng, người học có thể kiểm tra một tình huống bình thường và một tình huống bị gián đoạn. Ví dụ, sau khi mở ứng dụng trên trình giả lập, có thể thử xoay màn hình hoặc tạm chuyển sang ứng dụng khác rồi quay lại, sau đó quan sát dữ liệu còn hay mất, màn hình có phản hồi hay không và có thông báo nào giúp tiếp tục hay không. Bài thử nhỏ này biến yêu cầu "không mất dữ liệu" thành một kết quả cụ thể thay vì một nhận xét chung chung.

Bảng 1.4. *Từ ràng buộc thiết bị đến quyết định thiết kế*

Điều kiện	Điều người dùng có thể gặp	Hướng xử lý
Màn hình nhỏ	Nội dung rối, nút khó chạm	Ưu tiên nội dung chính, bố cục rõ và vùng chạm đủ lớn
Mạng không ổn định	Chờ lâu, dữ liệu không xuất hiện hoặc thao tác thất bại	Hiển thị trạng thái đang xử lý, giải thích lỗi và cho phép thử lại
Pin và bộ nhớ có giới hạn	Ứng dụng chậm, bị dừng hoặc hao pin	Hạn chế công việc không cần thiết và giữ lại dữ liệu quan trọng
Quyền chưa được cấp	Chức năng không thể tiếp tục	Xin quyền đúng lúc, giải thích mục đích và có phương án thay thế
Ứng dụng bị gián đoạn	Dữ liệu đang nhập hoặc kết quả vừa tải bị mất	Lưu phần trạng thái cần thiết để người dùng có thể tiếp tục

Bảng cho thấy một ràng buộc chỉ có ý nghĩa sư phạm khi người học nhìn được chuỗi nguyên nhân–kết quả: điều kiện nào thay đổi, người dùng gặp vấn đề gì và ứng dụng nên phản hồi ra sao. Cơ chế kỹ thuật cụ thể sẽ được học dần ở các chương về trạng thái, lưu trữ cục bộ và kết nối mạng; ở đây, mục tiêu là biết đặt đúng ba câu hỏi trước khi chọn cách xử lý. Hình 1.2 tiếp tục gom các điều kiện đó quanh thiết bị để người học nhận ra chúng có thể xuất hiện đồng thời. Khi đọc hình, cần liên hệ mỗi giới hạn với biểu hiện có thể quan sát trong Bảng 1.4, thay vì ghi nhớ các nhãn như những ý rời nhau.



Hình 1.2. *Ràng buộc tài nguyên của thiết bị di động*

Hình không yêu cầu ghi nhớ cấu tạo điện thoại; người học chỉ cần chọn một giới hạn, dự đoán biểu hiện rồi đối chiếu hướng xử lý tương ứng.

1.2.2. Tương tác người dùng trên thiết bị di động

Màn hình nhỏ và cảm ứng làm thay đổi cách tổ chức giao diện. Biểu mẫu cần ngắn, có thể cuộn khi bàn phím ảo xuất hiện; vùng chạm phải đủ lớn; hành động chính, trạng thái và dữ liệu sẽ lưu phải rõ ràng. Do phiên sử dụng thường ngắn và dễ gián đoạn, giao diện cũng cần phản hồi ngay và tránh buộc người dùng nhập lại dữ liệu.

Hoàn cảnh sử dụng còn làm thay đổi cách ưu tiên thao tác. Người dùng có thể đang di chuyển, chỉ dùng một tay hoặc cần hoàn tất công việc trong vài giây; vì vậy hành động chính phải dễ nhận ra, nút quan trọng không nên nằm ngoài vùng chạm thuận tiện và thông báo phải đủ ngắn để đọc nhanh. Khi cuộc gọi, thông báo hoặc bàn phím ảo che màn hình, ứng dụng cần giữ dữ liệu đã nhập và cho phép tiếp tục thay vì buộc người dùng bắt đầu lại.

Một giao diện có màu sắc hài hòa và hình ảnh bắt mắt chưa chắc tạo ra trải nghiệm tốt. Nếu người dùng không nhận ra hành động chính, không biết dữ liệu nào sai, không nhận được phản hồi sau khi chạm hoặc phải nhập lại thông tin sau gián đoạn, ứng dụng vẫn khó sử dụng dù hình thức đẹp. Chương này chỉ gợi mở mối quan hệ đó; Chương 2 sẽ phân biệt rõ giao diện người dùng với trải nghiệm người dùng và trình bày các nguyên tắc đánh giá có thể quan sát, kiểm chứng.

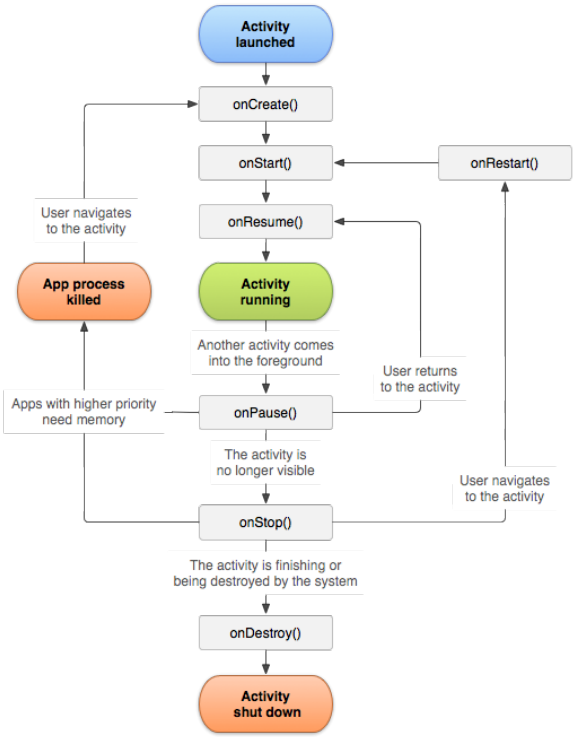
Khi phân tích một màn hình, có thể lần lượt đặt ba câu hỏi: người dùng vừa làm gì, bên trong ứng dụng thay đổi ra sao và màn hình phải phản hồi thế nào. Chẳng hạn, sau thao tác tải dữ liệu, ứng dụng chuyển sang trạng thái đang xử lý; giao diện tạm ngăn thao tác lặp và hiển thị tiến trình. Ba câu hỏi này không phải thuật ngữ cần học thuộc mà là thứ tự để đọc một màn hình. Khi nút không phản hồi hoặc thông báo lỗi khó hiểu, người học quay lại từng bước để xác định phần còn thiếu. Cách phân tích này về sau được gọi ngắn gọn là sự kiện–trạng thái–phản hồi và sẽ xuất hiện lại ở các chương về Compose và ViewModel.

Bảng 1.5. *Từ tương tác di động đến thiết kế giao diện*

Đặc điểm tương tác	Rủi ro nếu thiết kế kém	Định hướng xử lý
Chạm trực tiếp trên màn hình	Dễ bấm nhầm hoặc không biết vùng nào có thể thao tác	Nút đủ lớn, khoảng cách rõ, trạng thái bấm có phản hồi
Bàn phím ảo chiếm màn hình	Ô nhập liệu hoặc nút xác nhận bị che khuất	Biểu mẫu ngắn, cuộn hợp lý, đặt hành động chính ở vị trí dễ thấy
Phiên sử dụng ngắn và dễ gián đoạn	Người dùng mất dữ liệu khi chuyển ứng dụng, nhận cuộc gọi hoặc xoay máy	Lưu trạng thái cần thiết, hiển thị tiến trình, cho phép tiếp tục thao tác
Nhiều kích thước màn hình	Giao diện vỡ bố cục hoặc chữ quá nhỏ	Dùng bố cục linh hoạt, kiểm thử trên nhiều thiết bị/trình giả lập

1.2.3. Vòng đời ứng dụng

Màn hình `Hello World` chỉ thể hiện trạng thái thành công, trong khi ứng dụng thực tế còn có thể đang xử lý, chưa có dữ liệu, gặp lỗi hoặc thiếu quyền. Ví dụ, một màn hình tra cứu lịch học ban đầu chờ người dùng nhập điều kiện; sau khi yêu cầu được gửi, màn hình báo đang xử lý rồi hiển thị kết quả hoặc cho phép thử lại khi mất mạng. Nếu người dùng xoay điện thoại hay tạm chuyển sang ứng dụng khác, kết quả vừa tải không nên biến mất. Tình huống này dẫn đến khái niệm vòng đời: màn hình có thể được tạo, tạm dừng, quay lại hoặc kết thúc, và ứng dụng phải quyết định dữ liệu nào cần được giữ qua mỗi lần chuyển đổi.



Hình 1.3. *Vòng đời Activity theo tài liệu Android Developers*

Hình 1.3 sử dụng sơ đồ vòng đời Activity chính thức của Android Developers [27] để thể hiện cả đường đi thông thường và các nhánh quay lại. Khi Activity xuất hiện, các hàm được gọi theo thứ tự `onCreate()`–`onStart()`–`onResume()`; khi màn hình bị che hoặc chuyển xuống nền, Activity lần lượt đi qua `onPause()` và có thể đến `onStop()`. Nếu người dùng quay lại từ trạng thái tạm dừng, hệ thống gọi `onResume()`; nếu quay lại từ trạng thái đã dừng, chuỗi tiếp tục bằng `onRestart()`–`onStart()`–`onResume()`.

Người học chưa cần ghi nhớ máy móc mọi hàm, nhưng phải hiểu Activity đổi trạng thái khi màn hình được tạo, hiển thị, bị che, chuyển xuống nền hoặc bị hủy. Dòng chữ `Hello World` có thể dựng lại, còn dữ liệu biểu mẫu, bộ lọc hoặc kết quả vừa tải cần được giữ hợp lý. Khi kiểm tra một chuyển đổi, trước hết hãy nêu thao tác vừa xảy ra, sau đó quan sát trạng thái và dữ liệu còn lại. Cách diễn đạt theo chuỗi nguyên nhân–kết quả này giúp trao đổi lỗi rõ hơn. Việc ghi cả thao tác và kết quả dự kiến còn giúp phân biệt lỗi vòng đời với lỗi dữ liệu hoặc hiển thị. Một báo cáo ngắn nên nêu thao tác, trạng thái mong đợi, kết quả quan sát và dữ liệu còn lại; nhờ đó người khác có thể lặp lại đúng tình huống. Các chương sau sẽ trở lại yêu cầu này khi trình bày trạng thái, ViewModel và điều hướng.

1.3. Nền tảng Android và hệ sinh thái phát triển

1.3.1. Giới thiệu hệ điều hành Android

Khi người dùng chạm vào biểu tượng ứng dụng, mã do người phát triển viết không tự chạy độc lập. Android tạo màn hình, phân chia bộ nhớ, kiểm soát quyền và quyết định ứng dụng được tiếp tục hay tạm dừng. Vì vậy, có thể hiểu Android trước hết là môi trường điều hành ứng dụng; Kotlin là ngôn ngữ dùng để viết mã, còn các công cụ và thư viện sẽ được đưa vào khi dự án cần tạo giao diện, đóng gói hoặc chạy trên thiết bị.

Các khả năng mà ứng dụng sử dụng đến từ hai nơi. Một nhóm thuộc hệ điều hành trên thiết bị và có thể khác nhau theo phiên bản Android; nhóm còn lại là thư viện được khai báo cùng dự án. Sự phân biệt này giải thích vì sao dự án phải nêu phiên bản Android tối thiểu có thể chạy, đồng thời quản lý riêng phiên bản của các thư viện như Jetpack Compose. Tên cấu hình cụ thể sẽ được đọc ở phần cấu trúc dự án, sau khi quan hệ giữa các lớp trong Hình 1.4 đã rõ.

1.3.2. Hệ sinh thái Android

Một ứng dụng Android đi từ mã nguồn đến màn hình trên điện thoại qua nhiều bước: dự án phải chứa đúng tệp, được đóng gói, chuyển tới thiết bị rồi mới được hệ điều hành mở. Mỗi thành phần trong Bảng 1.6 phụ trách một phần của chuỗi đó. Ở lần đọc đầu, người học chỉ cần đánh dấu bước cuối cùng đã có kết quả và bước kế tiếp chưa hoàn tất. Bảng được dùng như bản đồ ban đầu; vai trò và dấu hiệu kiểm tra của từng công cụ sẽ được giải thích khi người học thực hiện bước tương ứng. Khi gặp lỗi, hãy ghi lại công cụ vừa thực hiện bước nào, kết quả mong đợi và dấu hiệu thực tế. Ba thông tin này giúp chọn đúng nơi kiểm tra tiếp theo mà chưa cần thay đổi nhiều tệp cùng lúc. Cách ghi này cũng tạo một nhật ký ngắn để so sánh các lần thử mà không phải dựa vào trí nhớ.



Hình 1.4. *Các lớp nền tảng Android ở mức nhập môn*

Hình 1.4 cho thấy ứng dụng nằm trên nhiều lớp cung cấp khả năng hệ thống. Người học chưa cần ghi nhớ tên mọi lớp; điều cần giữ lại là mã luôn chạy trong một môi trường có quy tắc riêng. Khi màn hình không mở hoặc một chức năng bị từ chối, nguyên nhân có thể nằm ở cấu hình dự án, thiết bị chạy thử hay quyền truy cập, chứ không chỉ ở câu lệnh Kotlin vừa viết. Thói quen xác định nơi lỗi xuất hiện trước khi sửa mã sẽ được dùng xuyên suốt các bài thực hành.

Bảng 1.6. *Vai trò các thành phần trong hệ sinh thái Android*

Thành phần	**Vai trò trong dự án Android đầu tiên**
Android Studio	Tạo dự án, soạn mã, quản lý thiết bị, xem lỗi và thực hiện `Run`.
Android SDK	Cung cấp API nền tảng và công cụ biên dịch, đóng gói, gỡ lỗi.
Gradle	Đọc cấu hình, tải phụ thuộc, biên dịch và tạo bản gỡ lỗi.
ADB	Kết nối thiết bị để cài ứng dụng, mở Activity và hỗ trợ gỡ lỗi.
Trình giả lập/thiết bị thật	Môi trường kiểm chứng giao diện, vòng đời và lỗi khi chạy.
Jetpack/Compose	Thư viện hỗ trợ ứng dụng hiện đại; Compose mô tả UI bằng Kotlin.

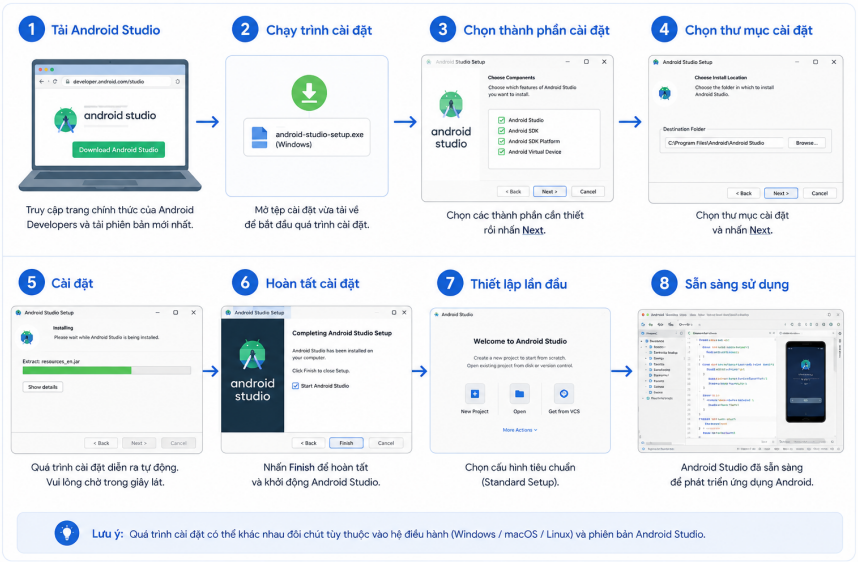
Tên của các thành phần trong bảng sẽ xuất hiện nhiều lần trong chương, nhưng mỗi thành phần có một nhiệm vụ đơn giản. Android Studio là nơi người học làm việc; Gradle chuẩn bị gói cài đặt; ADB đưa gói đó đến thiết bị; thiết bị cho thấy kết quả cuối cùng. Khi chuỗi này bị dừng, chỉ cần bắt đầu kiểm tra từ bước cuối cùng còn thành công thay vì thử sửa nhiều nơi cùng lúc.

Cách đọc theo chuỗi còn giúp người học mô tả lỗi bằng kết quả quan sát được, thay vì kết luận vội rằng toàn bộ dự án “không chạy”.

1.4. Môi trường và công cụ phát triển ứng dụng Android

Việc Android Studio mở được chỉ cho thấy chương trình soạn thảo đã sẵn sàng. Muốn đưa một ứng dụng lên điện thoại, máy tính còn cần bộ công cụ Android, cách đóng gói dự án và đường kết nối tới thiết bị chạy thử. Ba phần này lần lượt được gọi là Android SDK, Gradle và Android Debug Bridge (ADB). Quy trình cài đặt vì thế phải kiểm tra cả công cụ lẫn thiết bị; nếu một phần còn thiếu, dự án có thể dừng trước khi mã nguồn được thực hiện.

Môi trường Android thường cần nhiều tài nguyên hơn chương trình dòng lệnh hoặc web tĩnh, nhất là khi chạy trình giả lập. Máy ít RAM có thể khởi động thiết bị ảo rất chậm; ổ đĩa gần đầy dễ làm SDK Platform hoặc Build Tools tải thiếu; CPU chưa bật ảo hóa có thể khiến trình giả lập không chạy. Đây là lỗi môi trường, không phải lý do để sửa mã. Cấu hình tối thiểu thay đổi theo phiên bản Android Studio nên cần đối chiếu tài liệu cài đặt chính thức[1]; trên máy yếu, dùng điện thoại Android thật thường thực tế hơn. Hình 1.5 tóm tắt các bước chuẩn bị trước khi kiểm tra từng thành phần này.

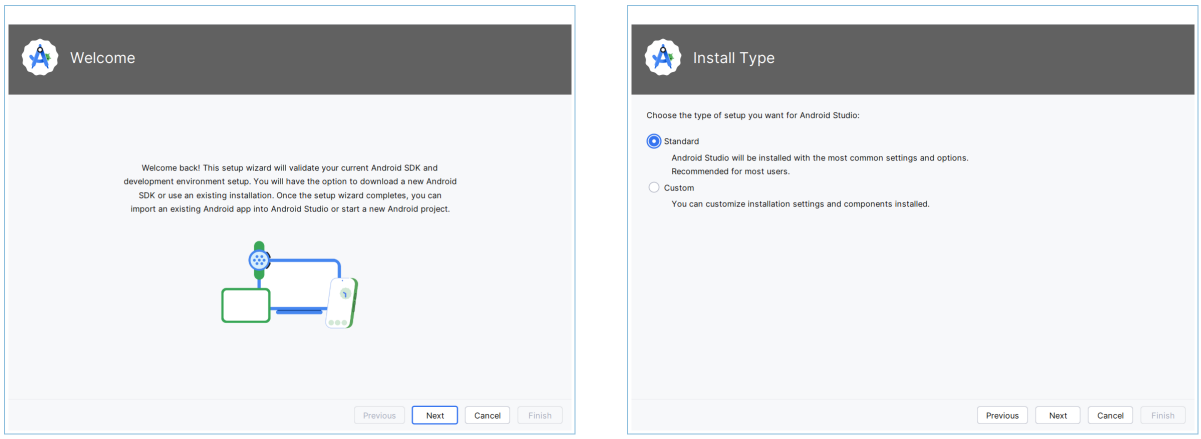


Hình 1.5. *Quy trình cài đặt Android Studio*

Quá trình chuẩn bị Android Studio gồm hai lớp thao tác: chạy bộ cài của hệ điều hành, sau đó đi qua Setup Wizard để chọn kiểu cài đặt và kiểm tra Android SDK. Bộ cài chỉ xác nhận IDE đã có trên máy; SDK Manager và Device Manager mới xác nhận platform, công cụ build và thiết bị chạy thử.

Hình 1.6 cho thấy màn hình chào và lựa chọn `Standard`, phù hợp với cấu hình phổ thông vì tự cài các thành phần thường dùng; `Custom` chỉ cần thiết khi phòng máy phải quản lý nhiều phiên bản hoặc thư mục cài đặt riêng. Sau khi trình hướng dẫn hoàn tất, cần kiểm tra lần lượt nơi lưu bộ công cụ Android, phiên bản nền tảng, công cụ đóng gói và thiết bị chạy thử. Các tên `SDK Location`, `SDK Platform`, `Build Tools` và `Platform Tools` trong cửa sổ cài đặt tương ứng với các mục đó. Android Studio mở được mới chỉ xác nhận chương trình soạn thảo đã khởi động; nếu còn báo thiếu SDK, nền tảng, công cụ build hoặc thiết bị đích thì cần hoàn tất môi trường trước khi sửa mã.

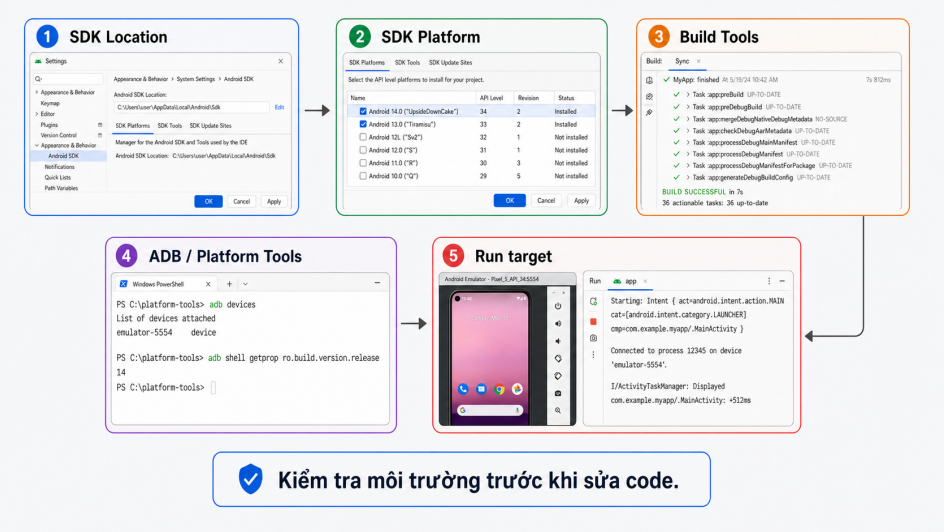
[1] `https://developer.android.com/studio/install`



(a) Khởi động Setup Wizard **(b)** Chọn kiểu cài đặt

Hình 1.6. *Hai bước đầu trong Android Studio Setup Wizard*

Nên lưu một hồ sơ môi trường gồm phiên bản Android Studio, đường dẫn SDK, API level đã cài, tên thiết bị chạy thử và kết quả `adb devices`. Nếu cùng một dự án chạy trên máy A nhưng lỗi trên máy B, các thông tin này cho phép so sánh SDK, Gradle và trạng thái ADB trước khi kết luận mã nguồn có vấn đề. Hình 1.7 hệ thống hóa các điểm kiểm tra đó.



Hình 1.7. *Bảng kiểm môi trường sau khi cài Android Studio*

Bảng kiểm này giúp tách lỗi chưa cài đủ công cụ khỏi lỗi xuất hiện khi build hoặc chạy ứng dụng. Chỉ sau khi các mục cơ bản đã được xác nhận, người học mới nên so sánh phiên bản và cấu hình chi tiết.

1.4.1. Bộ phiên bản đã kiểm chứng với giáo trình

Các ảnh thực hành và dự án `MobileProgrammingDemo` đã được kiểm chứng ngày 18/08/2026 cùng bộ phiên bản ở Bảng 1.7. Đây là một cấu hình tham chiếu đã chạy thành công, không phải danh sách phiên bản mới nhất và không buộc mọi máy phải giống tuyệt đối. Khi gặp lỗi, không nên thay các con số theo cảm tính mà cần ghi lại phiên bản trên

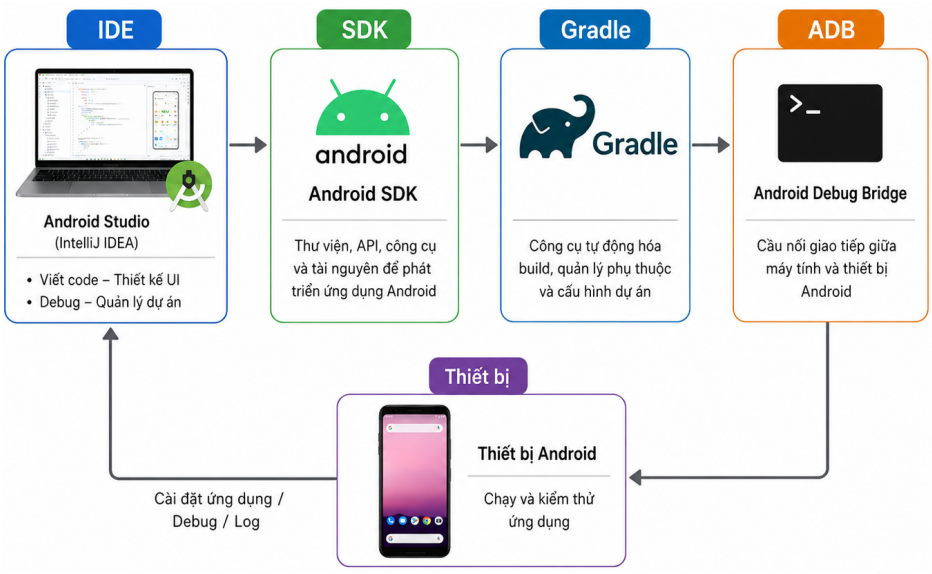
máy, xác định tác vụ nào thất bại rồi đối chiếu với bảng. Việc dùng SDK 36.1 để biên dịch nhưng chạy trên thiết bị Android 15/API 35 là hợp lệ: phiên bản dùng khi tạo ứng dụng và phiên bản của thiết bị chạy thử đảm nhiệm hai vai trò khác nhau.

Bảng 1.7. *Bộ phiên bản đã kiểm chứng với giáo trình*

Thành phần	**Phiên bản/cấu hình**	**Vai trò và nơi đối chiếu**
Android Studio	Panda 4, 2025.3.4 Patch 1	Mở, đồng bộ, build và chạy; xem tại `Help > About`.
JDK đi kèm IDE	OpenJDK 21.0.10	JDK dùng bởi Gradle; Java/Kotlin toolchain 21.
Gradle Wrapper	9.4.1	Khóa trong `gradle-wrapper.properties`; chạy bằng `gradlew`.
Android Gradle Plugin	9.2.1	Plugin build, khai báo trong `libs.versions.toml`.
Kotlin và KSP	Kotlin 2.2.10; KSP 2.3.11	Biên dịch Kotlin/Compose và sinh mã Room.
Android SDK	compile 36.1; target 36; min 24	Cấu hình trong `app/build.gradle.kts`.
SDK Build Tools	36.0.0	Tạo APK gỡ lỗi.
Platform-Tools/ADB	ADB 37.0.1	Kết nối, cài APK và mở ứng dụng; xem `adb version`.
Trình giả lập Android	37.1.11	Quản lý trong SDK Manager.
Thiết bị ảo tham chiếu	Android 15, API 35, Google APIs, x86_64	AVD dùng để chạy ứng dụng và chụp ảnh.
Jetpack Compose	Compose BOM 2026.02.01	Khóa bộ thư viện tương thích trong `libs.versions.toml`.

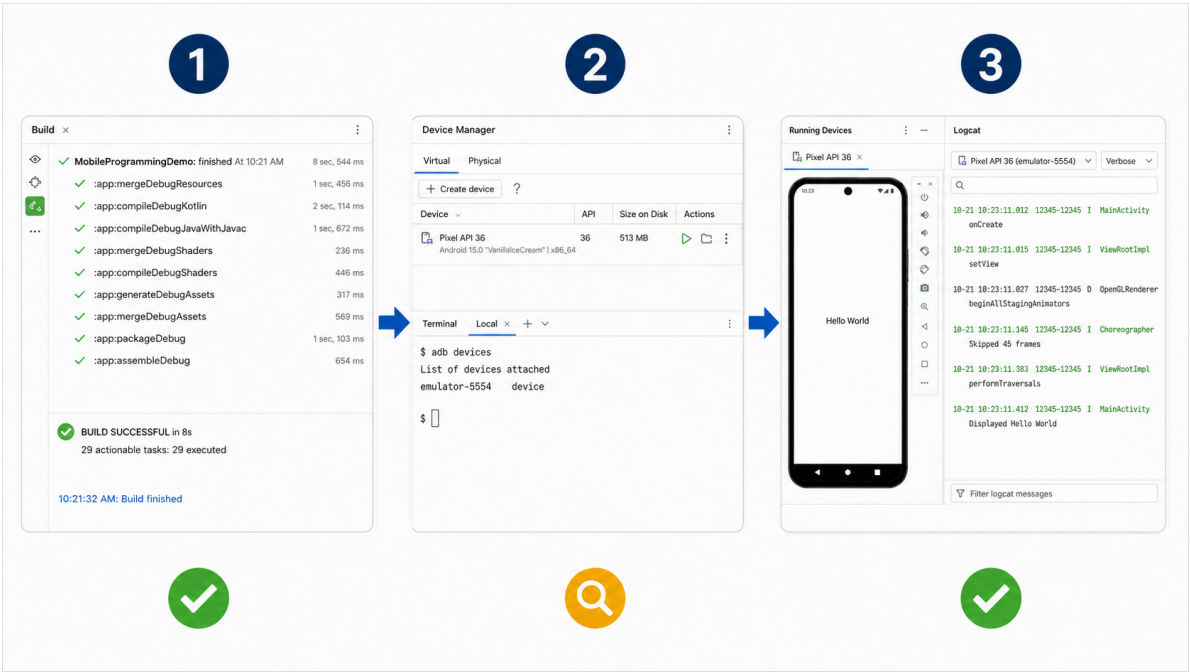
Một cấu hình tham chiếu giúp lớp học có điểm bắt đầu chung, nhất là khi cần đối chiếu ảnh minh họa hoặc hỗ trợ nhau khi gặp lỗi; nó không có nghĩa là mọi máy phải cài đúng từng số phiên bản trong bảng. Khi cấu hình khác tài liệu, cần xác định tác vụ đầu tiên bị dừng, thay đổi từng yếu tố nhỏ rồi chạy lại chính tác vụ đó. Cách làm này cho biết thay đổi nào thực sự có tác dụng và tránh tạo thêm lỗi do đổi nhiều phiên bản cùng lúc. Nếu dự án đã build và mở được đúng màn hình trên thiết bị, cấu hình đó có thể được ghi lại như một biến thể đã kiểm chứng cho phòng máy thay vì tiếp tục chỉnh chỉ để giống ảnh minh họa.

Khi bấm `Run`, Android Studio thực hiện thay người học một chuỗi thao tác: tạo bản ứng dụng, kết nối thiết bị, cài đặt rồi mở màn hình đầu tiên. Hình 1.8 cho biết công cụ nào phụ trách từng bước. Nhật ký kiểm chứng chỉ cần ghi tác vụ vừa chạy và kết quả quan sát được; nhờ đó, khi một bước thất bại, người học có thể xác định điểm dừng thay vì đổi nhiều cấu hình cùng lúc.



Hình 1.8. *Chuỗi công cụ từ Android Studio đến thiết bị chạy thử*

Hình 1.8 cho thấy Android Studio điều phối toàn bộ quy trình: Gradle tạo gói cài đặt và ADB chuyển gói đó tới thiết bị. Nếu chưa tạo được gói thì chưa cần kiểm tra kết nối; nếu thiết bị chưa sẵn sàng thì sửa giao diện không giải quyết được lỗi cài đặt. Trường hợp ứng dụng đã cài nhưng mở sai màn hình mới hướng việc kiểm tra sang cấu hình khởi chạy. Hình 1.9 chuyển chuỗi công cụ này thành ba điểm quan sát cụ thể.



Hình 1.9. *Ba điểm chẩn đoán chuỗi build–cài đặt–khởi chạy*

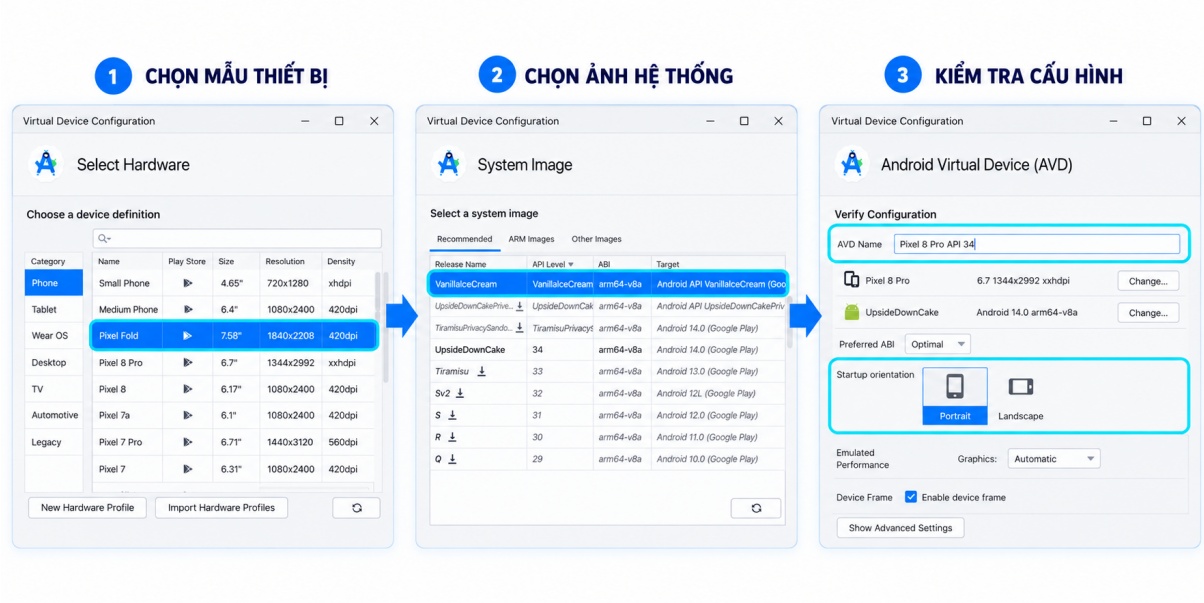
Ba điểm quan sát trong hình trả lời ba câu hỏi: gói cài đặt đã được tạo chưa, máy tính đã nhận thiết bị chưa và màn hình mong đợi đã xuất hiện chưa. Cửa sổ Build, kết quả `adb devices` và cửa sổ thiết bị cung cấp bằng chứng tương ứng. Người học không cần đọc ngay toàn bộ nhật ký hay ghi nhớ mọi trạng thái kỹ thuật; chỉ cần dừng ở mốc

đầu tiên chưa đạt. Nếu chưa build thành công thì chưa đổi thiết bị hoặc sửa giao diện; nếu thiết bị chưa sẵn sàng thì xử lý kết nối trước khi cài; chỉ khi ứng dụng đã cài mà màn hình chưa xuất hiện mới kiểm tra Activity và cấu hình khởi chạy.

Ba bằng chứng cũng không thay thế cho nhau. Dòng `BUILD SUCCESSFUL` cho biết gói cài đặt đã được tạo, kết quả `adb devices` cho biết máy tính đã nhận thiết bị, còn ảnh màn hình cho biết người dùng thực sự nhìn thấy gì. Khi chúng khớp nhau, người học có thể mô tả chính xác ứng dụng đã đi được đến đâu; khi chúng không khớp, sự khác biệt chỉ ra bước cần kiểm tra tiếp.

1.4.2. Máy ảo Android và thiết bị thật

Trình giả lập tạo một thiết bị Android ảo có phiên bản hệ điều hành, cấu hình phần cứng và kích thước màn hình xác định. Trong lớp học, nó giúp chuẩn hóa môi trường và giảm phụ thuộc vào điện thoại cá nhân. Thiết bị ảo được tạo và quản lý bằng Device Manager trong Android Studio; Hình 1.10 minh họa ba quyết định chính thay vì yêu cầu người học ghi nhớ toàn bộ cửa sổ cấu hình.



Hình 1.10. *Ba bước cấu hình thiết bị ảo trong Device Manager (nguồn ảnh giao diện: Android Developers)*

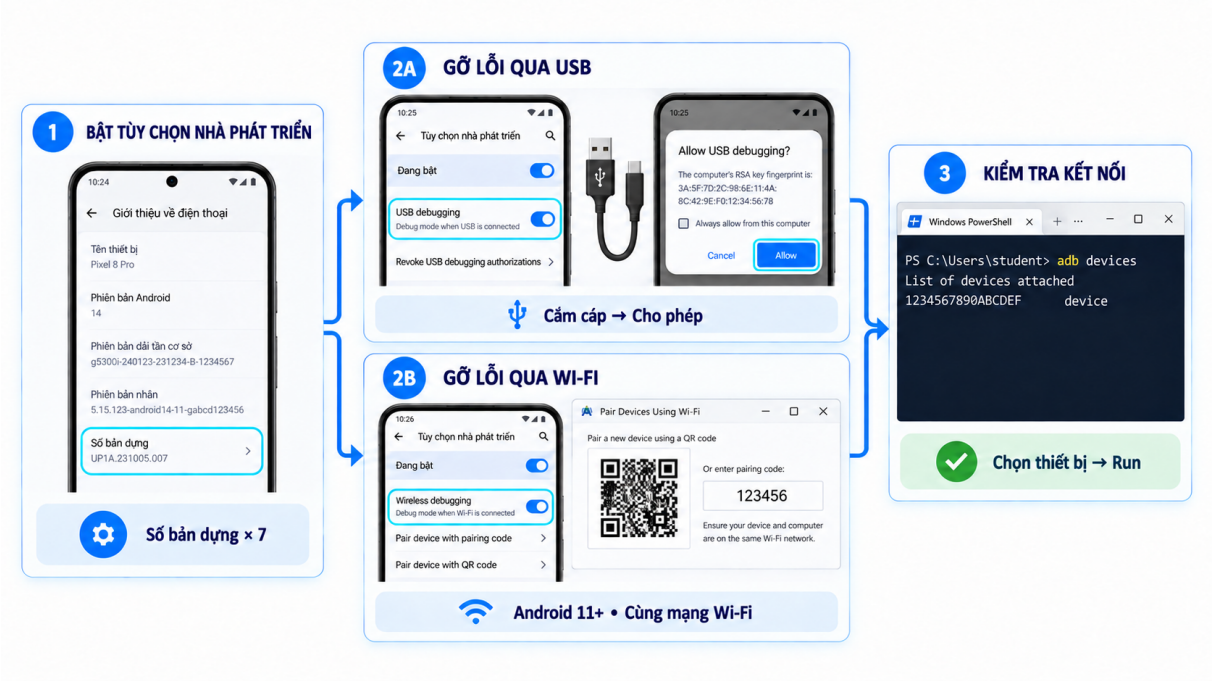
Ba khung trong hình được đọc từ trái sang phải: chọn mẫu phần cứng để xác định kiểu và kích thước màn hình; chọn ảnh hệ thống để xác định phiên bản Android cùng mức API; cuối cùng kiểm tra tên thiết bị và hướng màn hình trước khi hoàn tất. Mức API của ảnh hệ thống không được thấp hơn mức Android tối thiểu mà ứng dụng yêu cầu. Tên thiết bị, kiến trúc bộ xử lý và phiên bản trong hình chỉ là ví dụ giao diện; trên máy thực hành cần chọn ảnh hệ thống tương thích với máy và đối chiếu cấu hình tham chiếu của giáo trình. Sau khi hoàn tất, thiết bị phải xuất hiện trong Device Manager và danh sách thiết bị đích trên thanh công cụ[2]. Nếu cửa sổ trình giả lập đã mở nhưng Android Studio không chọn được thiết bị, cần kiểm tra trạng thái khởi động và kết nối ADB.

[2]Android Developers, “Create and manage virtual devices”, https://developer.android.com/studio/run/managing-avds.

Trình giả lập không thay thế hoàn toàn thiết bị thật, nhất là khi kiểm tra camera, cảm biến, vị trí, pin, hiệu năng hoặc chất lượng mạng. Dù dùng thiết bị ảo hay thật, tiêu chí vẫn giống nhau: Android Studio nhìn thấy thiết bị, Gradle tạo được ứng dụng, ADB cài được APK và màn hình chính mở thành công.

Thiết bị ảo phù hợp để tái tạo cấu hình màn hình, phiên bản Android và trạng thái khởi động, nhưng không phản ánh đầy đủ mức tiêu thụ pin, hiệu năng trên máy cấu hình thấp hoặc sự thay đổi mạng trong sử dụng thực tế. Ngược lại, thiết bị thật cho kết quả sát người dùng hơn nhưng khó chuẩn hóa giữa các máy trong lớp. Cách kiểm chứng hợp lý là dùng trình giả lập cho quy trình cơ bản và bổ sung thiết bị thật khi chức năng phụ thuộc phần cứng hoặc điều kiện mạng.

Với thiết bị thật, quy trình không bắt đầu ở Android Studio mà ở phần cài đặt của điện thoại. Sau khi mở tùy chọn nhà phát triển (`Developer options`), người học có thể kết nối bằng cáp USB hoặc ghép nối qua Wi-Fi; hai cách khác nhau ở bước cấp quyền nhưng cùng phải kết thúc bằng một thiết bị sẵn sàng trong ADB. Hình 1.11 đặt hai nhánh trong cùng một luồng để người học chọn cách phù hợp với thiết bị và điều kiện phòng máy[3].



Hình 1.11. *Từ tùy chọn nhà phát triển đến kết nối gỡ lỗi bằng USB hoặc Wi-Fi (phỏng theo Android Developers)*

Ở bước đầu, chạm bảy lần vào *Số bản dựng* để hiện tùy chọn nhà phát triển. Nhánh USB yêu cầu bật `USB debugging`, cắm cáp và chấp nhận khóa RSA; trên Windows có thể cần trình điều khiển của nhà sản xuất. Nhánh Wi-Fi yêu cầu Android 11 trở lên và cùng mạng với máy tính; sau khi bật `Wireless debugging`, chọn `Pair Devices Using Wi-Fi` rồi quét mã QR hoặc nhập mã sáu chữ số. Với cả hai cách, chỉ bấm `Run` khi `adb devices` báo `device`. Trạng thái `unauthorized` cho biết quyền chưa được chấp nhận; `offline` cho biết thiết bị chưa sẵn sàng.

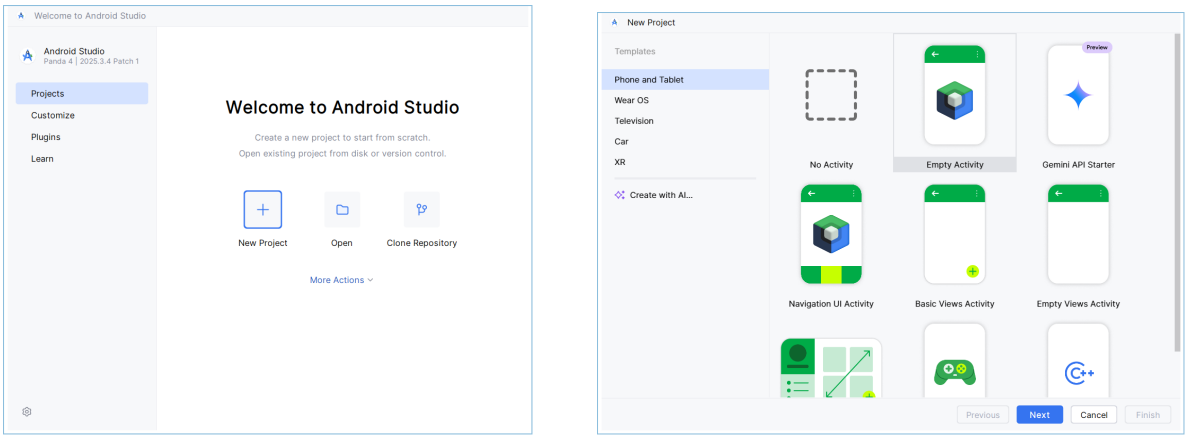
[3] Android Developers, "Run apps on a hardware device", https://developer.android.com/studio/run/device.

Trong buổi thực hành đầu tiên, nên chọn một thiết bị chạy ổn định rồi dùng lại cho toàn bộ chuỗi thao tác. Việc liên tục đổi giữa nhiều thiết bị ảo hoặc điện thoại thật làm tăng số yếu tố cần kiểm tra, trong khi mục tiêu trước mắt chỉ là chứng minh ứng dụng đầu tiên chạy được. Sau khi chuỗi cơ bản đã thành công, người học mới nên thử một cấu hình khác để quan sát sự thay đổi về kích thước màn hình, phiên bản Android hoặc trạng thái kết nối.

Khi báo cáo, cần ghi đúng mức đã kiểm chứng: cửa sổ Build chứng minh APK được tạo, ADB chứng minh kết nối thiết bị, còn ảnh trình giả lập chứng minh Activity đã mở và kết xuất giao diện. Ảnh giao diện không thay thế nhật ký build; dòng `BUILD SUCCESSFUL` cũng chưa chứng minh APK đã được cài và đúng Activity đã mở.

1.4.3. Cấu trúc dự án Android cơ bản

Ở màn hình chào, `New Project` tạo dự án từ mẫu, còn `Open` mở dự án đã có. Với dự án mới cần kiểm tra tên, tên gói, vị trí lưu và mẫu; với dự án có sẵn cần mở đúng thư mục gốc chứa cấu hình Gradle. Mở nhầm thư mục con có thể khiến Android Studio không nhận mô-đun `app` và không tạo cấu hình chạy.



(a) Chọn tạo dự án mới **(b)** Chọn mẫu `Empty Activity`

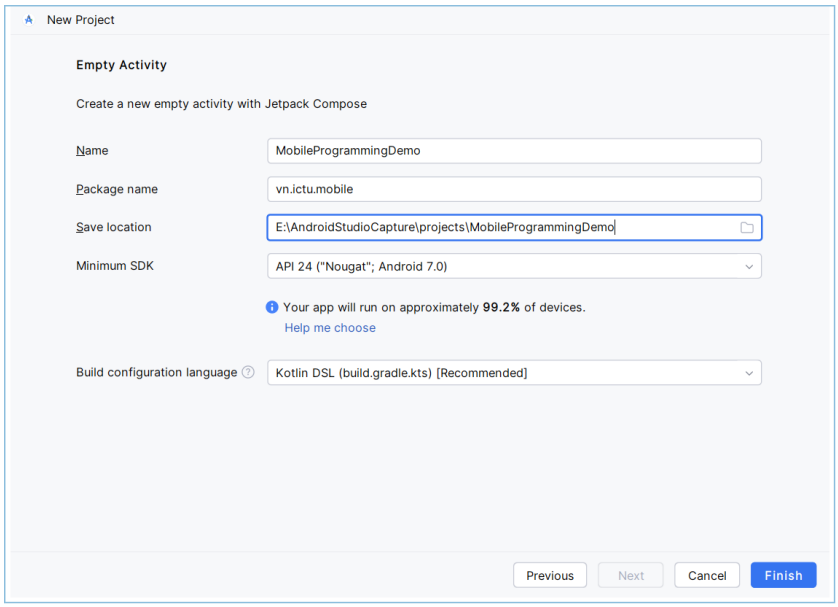
Hình 1.12. *Từ màn hình chào đến lựa chọn mẫu dự án*

Hình 1.12 ghép hai bước liên tiếp từ màn hình chào đến việc chọn mẫu. Mẫu `Empty Activity` tạo một ứng dụng tối giản, đủ nhỏ để người học đọc từ đầu đến cuối và quan sát đường đi từ mã đến giao diện. Mẫu có nhiều chức năng hơn thường tạo sẵn điều hướng hoặc màn hình phụ, khiến thành phần bắt buộc của dự án dễ bị lẫn với mã minh họa. Các tên kỹ thuật bên trong mẫu tối giản sẽ được giới thiệu lần lượt khi mở dự án, thay vì phải ghi nhớ ngay ở bước lựa chọn.

Ở bước cấu hình, trước hết cần phân biệt tên hiển thị với tên dùng để định danh trong hệ thống. Giáo trình đặt tên dự án là `MobileProgrammingDemo`; tên gói `vn.ictu.mobile` được viết bằng chữ thường, không có khoảng trắng và cần giữ ổn định để mã, cấu hình chạy và lệnh kiểm tra cùng chỉ tới một ứng dụng. Hai giá trị này xuất hiện lại ở các bước sau, nên ghi đúng từ đầu sẽ giúp ảnh chụp và kết quả thực hành khớp với dự án thật.

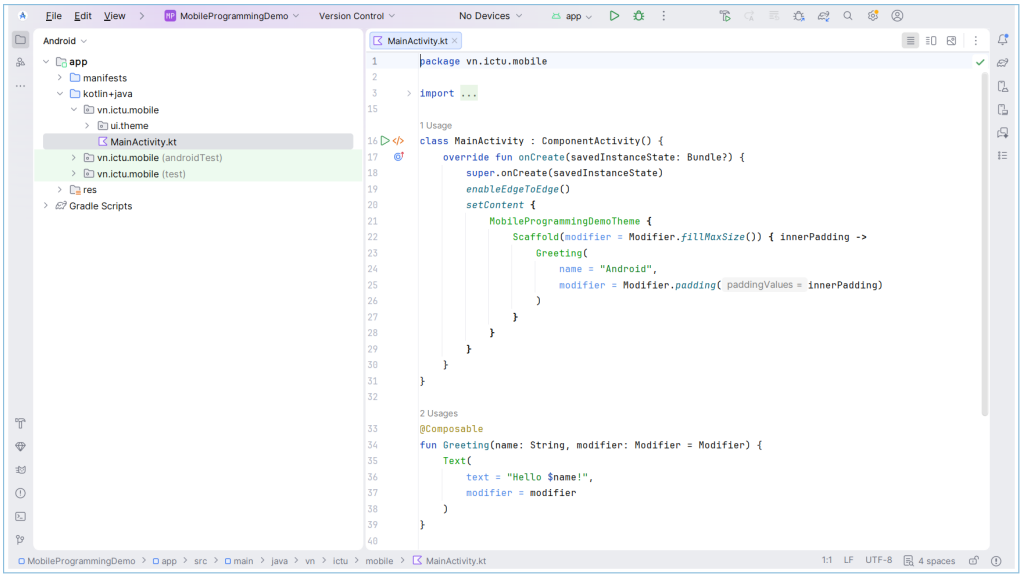
Tiếp theo là nơi lưu dự án và phiên bản Android thấp nhất có thể cài ứng dụng. Đường dẫn ngắn, dùng ký tự ASCII và không có khoảng trắng không phải yêu cầu của

Kotlin, nhưng giúp giảm lỗi trên các máy phòng thực hành khác nhau; API 24 được chọn làm mốc tham chiếu, không phải khuyến nghị cố định cho mọi sản phẩm. Trường `Build configuration language` được giữ ở lựa chọn `Kotlin DSL`, nghĩa là các tệp cấu hình có đuôi `.kts`; Chương 1 chỉ yêu cầu nhận ra nơi cấu hình được lưu, chưa yêu cầu chỉnh sâu. Hình 1.13 tập hợp các trường vừa giải thích để người học đối chiếu trước khi tạo dự án.



Hình 1.13. *Các trường cấu hình dự án mới*

Sau khi bấm `Finish`, Android Studio tạo dự án mới, mở tệp hoạt động chính và bắt đầu import/đồng bộ Gradle. Hình 1.14 ghi lại dự án tối giản ngay sau khi tạo, khi `MainActivity.kt` còn là màn hình Compose do mẫu sinh. Đây là bằng chứng của thao tác tạo dự án, không phải ảnh cấu trúc hiện tại của kho mã đồng hành đã phát triển qua nhiều chương; đồng thời, "tạo dự án" vẫn chưa chứng minh ứng dụng đã chạy.



Hình 1.14. *Tệp MainActivity của dự án Compose tối giản vừa được tạo*

Gradle đồng bộ là bước Android Studio đọc cấu hình và chuẩn bị môi trường build. Khi đồng bộ chưa hoàn tất, trình soạn thảo có thể gạch đỏ dù mã không sai; nếu đồng bộ lỗi, cần đọc cửa sổ `Build` hoặc `Sync` để kiểm tra mạng, SDK, plugin và đường dẫn dự án. Không nên sửa tệp Activity trước khi bước này thành công.

Ở chế độ `Android`, IDE nhóm tệp theo vai trò thay vì hiển thị nguyên cây thư mục vật lý. Manifest, mã Kotlin/Java, tài nguyên `res` và tệp Gradle vì thế được đặt gần nhau, nhưng đây vẫn chỉ là cách nhìn logic của Android Studio. Cách nhìn này giúp tìm đúng loại tệp trước khi lần theo đường dẫn thật trên ổ đĩa.

Sau khi nhận diện các nhóm tệp chính, người học nên đọc dự án theo hướng từ ngoài vào trong. Trước hết kiểm tra dự án có đồng bộ và build được không; tiếp theo xác định Activity cần chạy trong manifest; sau đó mới đọc `Chapter1FirstAppActivity.kt` và chỉnh chuỗi hiển thị.

Trong dự án tối giản vừa tạo, tệp đại diện cho màn hình là `MainActivity.kt`; còn kho mã đồng hành sử dụng `Chapter1FirstAppActivity.kt` cho bài này vì `MainActivity.kt` đã phục vụ ví dụ của chương khác. Khi mở đúng tệp, người học lần theo `onCreate`, `setContent` và hàm `Greeting` trong đoạn mã dưới đây để thấy màn hình được tạo như thế nào.

Mã nguồn 1.1. *Ý tưởng rút gọn của màn hình Compose đầu tiên*

```
class Chapter1FirstAppActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent { Greeting(name = "World") }
    }
}

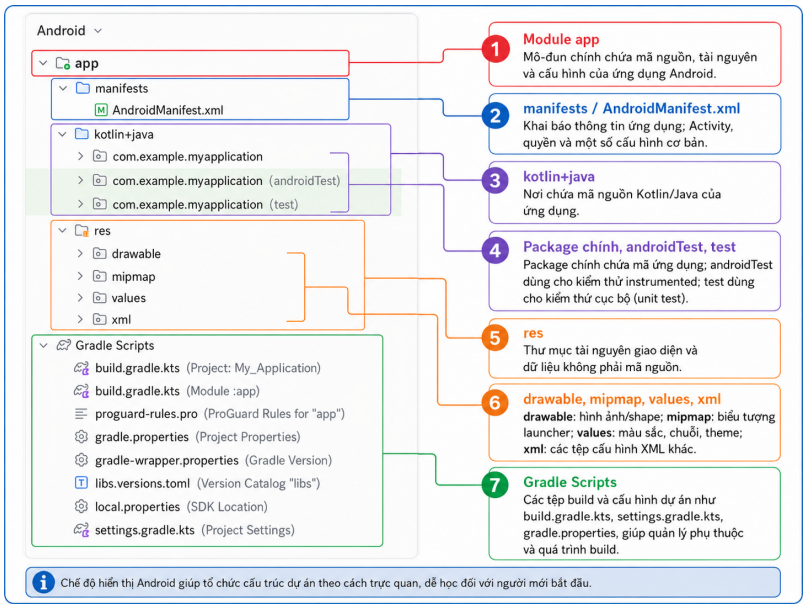
@Composable
fun Greeting(name: String) {
    Text(text = greetingText(name))
}

fun greetingText(name: String): String =
    if (name.isBlank()) "Hello Android" else "Hello
     ↪ ${name.trim()}"
```

Đoạn mã rút gọn đúng luồng của tệp đồng hành: Android gọi `onCreate`, phần `setContent` đặt nội dung lên màn hình, `Greeting` nhận tên cần hiển thị và `greetingText` tạo câu chào. Người học chưa cần giải thích mọi ký hiệu; mục tiêu là theo được dữ liệu `"World"` đến dòng chữ xuất hiện trên thiết bị. Để đặt tệp này vào toàn bộ dự án, Bảng 1.8 tóm tắt các nhóm tệp và vai trò sẽ tiếp tục được dùng ở những chương sau.

Hình 1.15 đặt tệp vừa đọc vào toàn bộ cấu trúc dự án. Các vùng được đánh số giúp chuyển từ một đoạn mã cụ thể sang quan hệ giữa mô-đun ứng dụng, manifest, mã Kotlin, tài nguyên và cấu hình build.

Từ hình, người học có thể tìm tệp đại diện cho màn hình trong nhóm mã nguồn,



Hình 1.15. *Các vùng chính của dự án trong chế độ Android*

kiểm tra màn hình khởi đầu trong manifest rồi mới xem tài nguyên hoặc cấu hình khi cần. Bảng dưới đây hệ thống lại vai trò của từng nhóm để dùng cho các chương tiếp theo.

Bảng 1.8. *Các thành phần của dự án Android nhập môn*

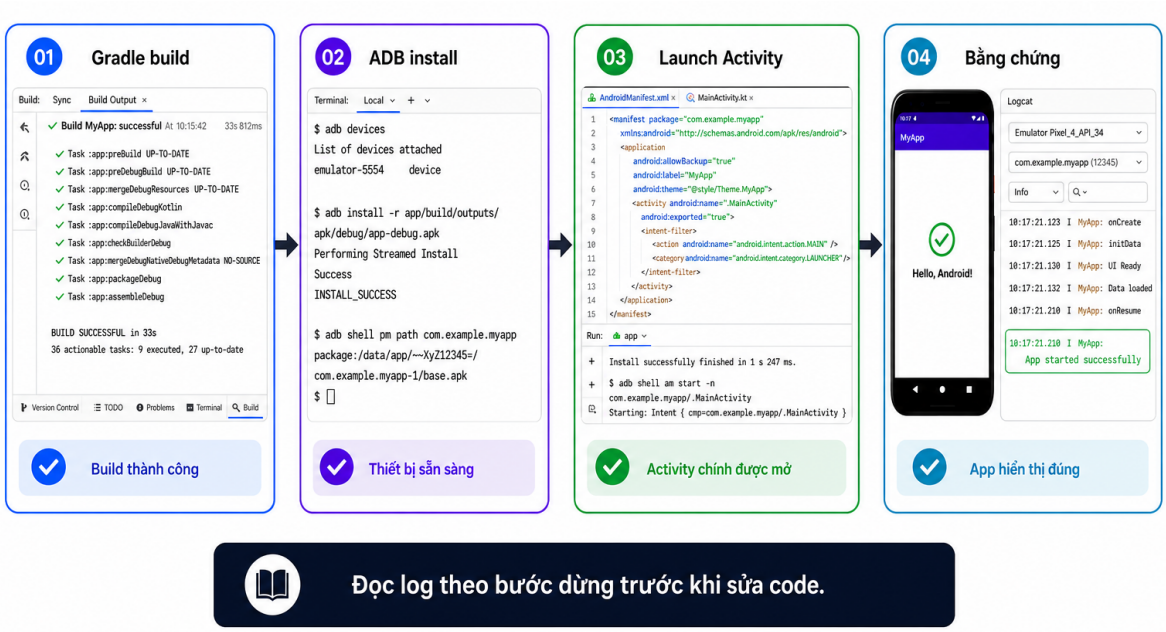
Thành phần	Ví dụ thường gặp	Vai trò
Mã nguồn Kotlin	Activity, composable, lớp dữ liệu	Logic, điểm vào màn hình và giao diện bằng mã
Tài nguyên ứng dụng	`res`, ảnh, chuỗi, màu, chủ đề	Tách tài nguyên giao diện khỏi mã xử lý
Manifest	`AndroidManifest.xml`	Khai báo thành phần, Activity và quyền
Cấu hình build	`build.gradle.kts`, plugin, SDK	Quy định cách biên dịch, đóng gói và liên kết thư viện
Tên gói/namespace	`vn.ictu.mobile`	Định danh mã và ứng dụng, thống nhất với Gradle

Cách đọc từ Bảng 1.8 giúp tránh việc mở nhầm `MainActivity.kt` của ví dụ khác hoặc cố sửa mã khi dự án chưa đồng bộ và thiết bị chưa sẵn sàng.

Mã nguồn đồng hành của Chương 1

Sau khi tự tạo, build và chạy dự án đầu tiên, người học có thể mở GitHub – mã nguồn Chương 1 để đối chiếu `Chapter1FirstAppActivity.kt` và `Chapter1GreetingTest.kt`. Activity minh họa luồng `onCreate`–`setContent`–`Greeting`; tệp kiểm thử kiểm tra riêng quy tắc tạo lời chào. Nên tự hoàn thành dự án trước khi đối chiếu để liên kết này đóng vai trò kiểm chứng, không thay thế thao tác thực hành.

Trước khi bấm `Run`, cần trả lời ba câu hỏi bằng ngôn ngữ đơn giản: phần nào của dự án sẽ chạy, chạy trên thiết bị nào và màn hình nào dự kiến xuất hiện. Android Studio sau đó mới thực hiện chuỗi build–cài đặt–khởi chạy đã giới thiệu. Với dự án tối giản, màn hình đầu là `MainActivity`; trong kho mã đồng hành, cấu hình mặc định mở `DemoHubActivity`, nên cần chọn đúng Activity của Chương 1 khi kiểm chứng bài học. Nếu thanh công cụ báo `No Devices`, quy trình phải dừng ở bước chọn thiết bị, chưa có lý do để sửa mã giao diện.



Hình 1.16. *Chuỗi thao tác khi bấm Run*

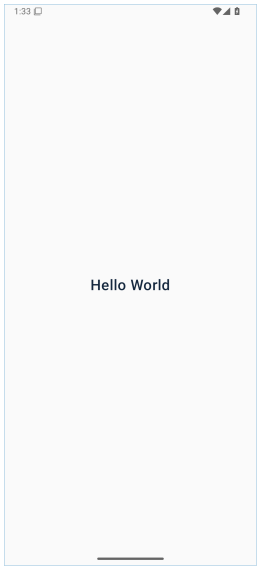
Hình 1.16 chia thao tác `Run` thành bốn điểm kiểm tra. Nếu lỗi xuất hiện khi tạo ứng dụng thì xem cấu hình hoặc mã; nếu lỗi ở bước đưa ứng dụng tới thiết bị thì xem kết nối; nếu ứng dụng đã cài nhưng mở sai màn hình thì kiểm tra cấu hình khởi chạy. Sau khi ba bước đầu đạt, Hình 1.17 cung cấp bằng chứng cuối cùng: màn hình lời chào thực sự xuất hiện trên thiết bị.

Khi ghi nhật ký chạy, mỗi điểm kiểm tra nên đi kèm đúng loại bằng chứng của nó: cửa sổ Build cho biết gói cài đặt đã được tạo, `adb devices` xác nhận thiết bị sẵn sàng, thông tin khởi chạy cho biết Activity được mở và ảnh màn hình xác nhận kết quả người dùng nhìn thấy. Nếu quy trình dừng, hãy ghi bằng chứng cuối cùng còn hợp lệ cùng thông báo đầu tiên xuất hiện sau điểm đó; cách ghi này giúp khoanh vùng lỗi mà không phải sửa đồng thời nhiều thành phần.

Trước khi chuyển sang phần đọc mã, người học cần tự xác nhận bốn năng lực tối thiểu. Thứ nhất, mở được dự án Android và hiểu mô-đun `app` là nơi chứa mã chính. Thứ hai, phân biệt được việc Android Studio chuẩn bị dự án với việc tạo gói cài đặt. Thứ ba, biết thiết bị ảo hoặc thiết bị thật nào đang được chọn để chạy. Thứ tư, biết dùng ít nhất một lệnh ADB để kiểm tra kết nối hoặc mở ứng dụng. Nếu thiếu một trong bốn điểm này, lỗi môi trường rất dễ bị hiểu nhầm là lỗi mã. Hai hoạt động sau giúp người học tự kiểm tra các năng lực đó và mô tả rõ mắt xích chưa đạt.

Thực hành 1.1: Lập hồ sơ kiểm chứng môi trường Android

1. Chụp hoặc ghi lại phiên bản Android Studio, SDK Location và API level đã cài trong SDK Manager.
2. Chạy `adb devices` và ghi lại tên thiết bị ảo hoặc thiết bị thật được nhận diện.
3. Build dự án bằng Android Studio hoặc lệnh `./gradlew :app:assembleDebug`.
4. Mở `Chapter1FirstAppActivity` trên trình giả lập, chụp màn hình `Hello World` và ghi rõ đó là bằng chứng của bước build, cài đặt hay khởi chạy.



Hình 1.17. *Màn hình Hello World trên trình giả lập*

Ảnh này là bằng chứng đầu ra khi đi kèm tên Activity đã mở: Gradle đã build, ADB đã cài APK, `Chapter1FirstAppActivity` đã chạy và Compose đã kết xuất giao diện. Ảnh không thay thế kiểm thử đơn vị, nhưng xác nhận chuỗi chạy tối thiểu đã đi tới kết quả người học nhìn thấy.

Thực hành 1.2: Phân biệt lỗi môi trường và lỗi mã nguồn

1. Mô tả dấu hiệu khi Gradle đồng bộ chưa hoàn tất.
2. Mô tả dấu hiệu khi Android Studio chưa chọn thiết bị chạy.
3. Mô tả dấu hiệu khi ứng dụng đã build được nhưng Activity chưa mở đúng.
4. Với mỗi dấu hiệu, ghi một nơi cần kiểm tra trước khi sửa tệp Kotlin.

1.5. Giới thiệu ngôn ngữ Kotlin trong Android

Sau khi dự án đã chạy, câu hỏi tiếp theo là dòng chữ trên màn hình được tạo ra từ đâu. Người học mở `Chapter1FirstAppActivity.kt` và lần theo đường đi từ tệp mã

đến kết quả quan sát được. Ngôn ngữ dùng trong tệp là Kotlin; đây cũng là ngôn ngữ chính của giáo trình. Chương này chỉ giới thiệu lượng kiến thức vừa đủ để đọc và thay đổi ví dụ đầu tiên, còn Chương 3 sẽ trình bày Kotlin có hệ thống.

Cách đọc phù hợp là đi từ ngoài vào trong: xác định tệp đang mở, tìm lớp đại diện cho màn hình, tìm phần được thực hiện khi màn hình bắt đầu và cuối cùng tìm giá trị được đưa ra giao diện. Trong mã, các từ `package`, `import`, `class` và `fun` lần lượt cho biết tệp thuộc nhóm nào, dùng thêm thành phần nào, định nghĩa đối tượng nào và thực hiện thao tác nào. Người học chưa cần ghi nhớ toàn bộ cú pháp; trước hết chỉ cần trả lời được "dữ liệu đi vào ở đâu" và "kết quả xuất hiện ở đâu".

Ví dụ sau bắt đầu bằng hai dữ liệu quen thuộc: tên học phần và số bài đã hoàn thành. `val` được dùng khi một giá trị không cần gán lại, còn `var` dành cho giá trị có thể thay đổi. Hàm `progressText` nhận tổng số bài, tính phần trăm rồi trả về chuỗi để hiển thị; biểu thức `if` tránh phép chia cho không khi chưa có bài học nào.

Mã nguồn 1.2. *Một ví dụ Kotlin nhỏ về biến, hàm và biểu thức điều kiện*

```
val courseName: String = "Lap trinh di dong"
var completedLessons: Int = 1

fun progressText(totalLessons: Int): String {
    val percent = if (totalLessons == 0) {
        0
    } else {
        completedLessons * 100 / totalLessons
    }
    return "$courseName: da hoc $percent%"
}
```

Trong Mã nguồn 1.2, phần sau dấu hai chấm cho biết loại dữ liệu: `String` là chuỗi chữ và `Int` là số nguyên. Kotlin có thể tự nhận ra loại dữ liệu từ giá trị ban đầu, nhưng việc ghi rõ loại ở đầu vào và kết quả của hàm giúp người đọc biết hàm chờ nhận gì và trả lại gì. Có thể thay `completedLessons` hoặc `totalLessons`, dự đoán chuỗi kết quả rồi chạy lại để kiểm tra; thao tác nhỏ này nối cú pháp với hành vi thay vì học từng từ khóa riêng lẻ.

Một tình huống khác thường gặp là người dùng chưa chọn thiết bị chạy thử. Khi đó, chương trình cần biểu diễn rõ rằng tên thiết bị có thể chưa tồn tại. Trong Kotlin, dấu hỏi sau loại dữ liệu cho phép giá trị nhận `null`; toán tử `?:` cung cấp nội dung thay thế, còn `?.` chỉ truy cập thuộc tính khi giá trị thực sự tồn tại.

Mã nguồn 1.3. *Xử lý tên thiết bị có thể chưa có giá trị*

```
val selectedDevice: String? = null
val displayName = selectedDevice ?: "Chua chon thiet bi"
val nameLength = selectedDevice?.length
```

Ở dòng đầu, `String?` cho biết `selectedDevice` có thể chưa có tên. Khi điều

đó xảy ra, `displayName` nhận chuỗi thay thế để giao diện vẫn có thông tin rõ ràng, còn `nameLength` cũng chưa có giá trị thay vì gây lỗi vì truy cập một đối tượng không tồn tại. Cách xử lý này được gọi là xử lý null an toàn; Chương 3 sẽ giải thích sâu hơn và áp dụng cho dữ liệu từ biểu mẫu, cấu hình và các nguồn dữ liệu khác.

Dự án đầu tiên dùng Kotlin để xử lý dữ liệu và Jetpack Compose để mô tả phần giao diện cần hiển thị. Hàm `Greeting(name: String)` là ví dụ nhỏ nhất: dữ liệu `name` quyết định dòng chữ trên màn hình, nên việc đổi đối số rồi chạy lại cho thấy trực tiếp quan hệ giữa dữ liệu và giao diện. Chương 2 sẽ trình bày cách xây dựng giao diện bằng View/XML để người học đọc được dự án Android truyền thống; Chương 4 mới giải thích chi tiết cách Compose cập nhật giao diện theo dữ liệu hiện tại. Nhờ vậy, ở Chương 1 người học chỉ cần theo được đường đi của một giá trị, không phải tiếp nhận đồng thời toàn bộ hai mô hình giao diện.

Thực hành 1.3: Đọc nhanh tệp Kotlin đầu tiên

---

1. Mở tệp `Chapter1FirstAppActivity.kt` của dự án `MobileProgrammingDemo`.
2. Tìm khai báo `package vn.ictu.mobile.chapter1` và giải thích vì sao đây là gói con của namespace `vn.ictu.mobile`.
3. Tìm ít nhất hai từ khóa hoặc tên hàm Kotlin đã gặp trong mục này, chẳng hạn `fun`, `val`, `onCreate`, `setContent`.
4. Xác định phần nào của tệp dùng để mở màn hình đầu tiên và phần nào mô tả nội dung hiển thị.
5. Đổi đối số `"World"`, build và mở lại đúng Activity; ghi bằng chứng cho thấy thay đổi đã có hiệu lực.

1.6. Kiểm thử dự án Android đầu tiên

Chạy ứng dụng và nhìn thấy kết quả đúng là bằng chứng quan trọng, nhưng chưa cho biết riêng quy tắc tạo dữ liệu có chính xác hay không. Vì vậy, chương này giới thiệu hai mức kiểm tra nhập môn [3]. Kiểm thử đơn vị cục bộ chạy một hàm Kotlin nhỏ ngay trên máy phát triển, không cần mở trình giả lập; kiểm thử nhanh lại đi qua toàn bộ chuỗi build–cài đặt–khởi chạy để xác nhận dự án có thể hoạt động trên thiết bị. Theo cách tổ chức của Android, hai loại kiểm thử nằm ở những khu vực khác nhau trong dự án: `app/src/test` dành cho kiểm thử cục bộ, còn `app/src/androidTest` dành cho kiểm thử cần môi trường Android. Chương 1 chưa yêu cầu viết kiểm thử trên thiết bị; việc phân biệt này chỉ nhằm tránh hiểu rằng mọi kiểm thử Android đều phải mở trình giả lập. Cách tổ chức kiểm thử chính thức được trình bày trong [28]; việc dùng phép thử tự động làm bằng chứng có thể lặp lại trong dự án [6] cũng phù hợp với các tài liệu nhập môn Android–Kotlin và thực hành kiểm thử phần mềm trong [2, 10, 11, 20].

1.6.1. Chọn đối tượng kiểm thử nhỏ và rõ

Hàm composable `Greeting` vừa tạo đối tượng giao diện Compose vừa quyết định chuỗi cần hiển thị. Nếu chỉ muốn kiểm tra quy tắc tạo lời chào, khởi động cả giao diện

là không cần thiết. Có thể tách quy tắc đó thành hàm `greetingText`: nhận tên, loại bỏ khoảng trắng thừa và dùng `Hello Android` khi tên chỉ gồm khoảng trắng. Sau đó, `Greeting` truyền kết quả của hàm này cho thành phần `Text`.

Mã nguồn 1.4. *Tách logic tạo lời chào khỏi hàm Composable*

```
fun greetingText(name: String): String =
    if (name.isBlank()) "Hello Android" else "Hello
    ↪ ${name.trim()}"
```

Việc tách hàm tạo một ranh giới dễ kiểm tra: đầu vào là `name`, đầu ra là một `String`. Kiểm thử chỉ kiểm tra quy tắc do dự án viết, không kiểm tra cách thư viện Compose vẽ thành phần `Text`.

1.6.2. Viết kiểm thử đơn vị cục bộ đầu tiên

Dự án đã có thư viện JUnit để chạy kiểm thử trong thư mục `test`. Tệp `Chapter1GreetingTest.kt` chứa hai tình huống độc lập: một tên thông thường và một chuỗi chỉ có khoảng trắng. Mỗi hàm được đánh dấu bằng `@Test`; tên hàm cho biết điều kiện đầu vào và kết quả mong đợi để người đọc hiểu mục đích trước khi xem từng dòng. Bên trong mỗi ca, có thể lần theo ba bước: chuẩn bị dữ liệu, gọi hàm và so sánh kết quả. Cấu trúc này thường được gọi là Arrange–Act–Assert, nhưng ở lần đầu người học chỉ cần trả lời ba câu hỏi tương ứng thay vì ghi nhớ tên tiếng Anh.

Mã nguồn 1.5. *Hai kiểm thử đơn vị cho quy tắc tạo lời chào*

```
class Chapter1GreetingTest {
    @Test
    fun nameProvided_returnsPersonalizedGreeting() {
        val actual = greetingText("World")
        assertEquals("Hello World", actual)
    }
    @Test
    fun blankName_returnsAndroidFallback() {
        val actual = greetingText("   ")
        assertEquals("Hello Android", actual)
    }
}
```

Khi đọc kết quả, cần đối chiếu riêng từng ca để biết lỗi thuộc dữ liệu có tên hay dữ liệu rỗng. Mỗi ca chỉ thay một yếu tố đầu vào, nên khi thất bại có thể nối thông báo trực tiếp với quy tắc tương ứng. Cách tách ca cũng giúp bổ sung dữ liệu biên mà không phải khởi động giao diện, nhờ đó người học nhận phản hồi nhanh hơn khi sửa hàm. Trên Windows, lệnh dưới đây chạy toàn bộ kiểm thử đơn vị cục bộ của mô-đun ứng dụng từ thư mục gốc dự án; trên macOS hoặc Linux, thay `gradlew.bat` bằng `./gradlew`. Ba bước chuẩn bị–thực hiện–đối chiếu sẽ được vận dụng dần ở các chương sau.

Mã nguồn 1.6. *Chạy kiểm thử đơn vị cục bộ của mô-đun ứng dụng*

```
gradlew.bat :app:testDebugUnitTest
```

Nếu kiểm thử thành công, Gradle báo `BUILD SUCCESSFUL`; báo cáo HTML nằm trong `app/build/reports/tests/testDebugUnitTest`. Khi `assertEquals` thất bại, cần đối chiếu yêu cầu với cả giá trị mong đợi và giá trị thực tế, không sửa kết quả mong đợi chỉ để kiểm thử chuyển sang màu xanh[4].

1.6.3. Kiểm thử nhanh build–cài đặt–khởi chạy

Kiểm thử đơn vị của `greetingText` không chứng minh manifest đúng, APK cài được hoặc trình giả lập đang hoạt động. Do đó, cần thêm một chuỗi kiểm tra nhanh để xác nhận đường đi tối thiểu của ứng dụng; đây chưa phải kiểm thử giao diện tự động.

Mã nguồn 1.7. *Các lệnh kiểm thử nhanh cho ứng dụng đầu tiên trên Windows*

```
gradlew.bat :app:assembleDebug
adb devices
adb install -r app\build\outputs\apk\debug\app-debug.apk
adb shell am start -n
 ↪  vn.ictu.mobile/.chapter1.Chapter1FirstAppActivity
```

Bốn lệnh lần lượt chứng minh APK build được, ADB nhận thiết bị, APK cài được và đúng Activity được mở. Nếu kiểm thử đơn vị đạt nhưng Activity không mở, cần kiểm tra thiết bị, APK, manifest hoặc tên Activity thay vì sửa `greetingText`. Màn hình `Hello World` là bằng chứng quan sát cuối cùng.

Thực hành 1.4: Chạy kiểm thử đơn vị và kiểm thử nhanh đầu tiên

---

1. Mở `Chapter1GreetingTest.kt`, xác định dữ liệu đầu vào, hành động và kết quả mong đợi của từng kiểm thử.
2. Chạy `:app:testDebugUnitTest` và lưu dòng kết quả cuối của Gradle.
3. Tạm đổi kết quả mong đợi của kiểm thử đầu, chạy lại để đọc thông báo thất bại, sau đó khôi phục giá trị đúng.
4. Build APK, kiểm tra thiết bị bằng ADB, cài ứng dụng và mở Activity lời chào; ghi bằng chứng cho từng bước.
5. Giải thích ngắn vì sao kiểm thử đơn vị không thể thay thế kiểm tra nhanh trên thiết bị.

Kết quả kiểm thử là ngữ cảnh kỹ thuật cho AI: cần nêu tác vụ thất bại, kết quả mong đợi, kết quả thực tế và trạng thái thiết bị. Mọi đề xuất vẫn phải được kiểm chứng.

[4] Android Developers, "Build local unit tests", https://developer.android.com/training/testing/local-tests.

1.7. Sử dụng AI tác tử trong thực hành Android đầu tiên

AI có thể giúp đọc lỗi hoặc đề xuất bước kiểm tra trong quá trình tự học và kiểm chứng môi trường, nhưng không thay thế việc tự tạo, build và chạy ứng dụng; chuẩn đánh giá vẫn là kết quả quan sát được trên dự án.

1.7.1. Gemini Tác tử trong Android Studio

Android Studio có thể dùng panel `Agent` của Gemini để đặt câu hỏi trong ngữ cảnh dự án đang mở. Dù dùng mô hình nào theo chính sách lớp học, AI chỉ hỗ trợ đọc lỗi và lập kế hoạch; mọi đề xuất vẫn phải được kiểm tra bằng build, chạy thử và quan sát trên thiết bị.

Trong bài đầu tiên, AI phù hợp nhất để giúp xác định tệp cần xem, bước đang lỗi và cách thu thập bằng chứng trong chuỗi build–cài đặt–khởi chạy. Không nên chỉ hỏi "ứng dụng không chạy"; cần nêu tác vụ thất bại, thông báo chính, trạng thái trong `adb devices` và bước cuối cùng đã thành công. Kết quả tạo sinh có thể không khớp cấu hình dự án, nên mọi đề xuất vẫn phải được giới hạn phạm vi và kiểm chứng lại.

1.7.2. Prompt mẫu cho bài thực hành

Thay vì yêu cầu chung chung rằng "ứng dụng không chạy", người học cần cho AI biết bước vừa thực hiện, dấu hiệu quan sát được và mục tiêu cần đạt. Prompt dưới đây chỉ yêu cầu lập kế hoạch kiểm tra; nó không trao quyền sửa tệp và không giả định nguyên nhân khi chưa có bằng chứng.

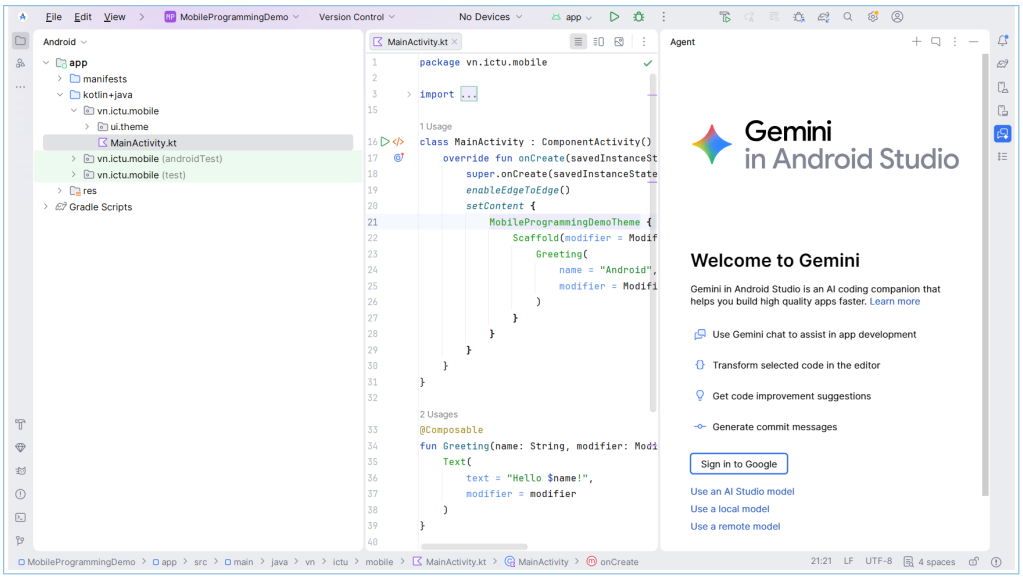
Mã nguồn 1.8. *Prompt khoanh vùng lỗi khi chạy dự án lần đầu*

```
Tôi đang mở dự án MobileProgrammingDemo trong Android Studio.
Khi chạy Chapter1FirstAppActivity, ứng dụng chưa hiện màn hình
 ↪  lời chào.
Hãy lập bảng kiểm theo thứ tự: build, kết nối thiết bị, cài
 ↪  đặt và khởi chạy.
Với mỗi bước, nêu dấu hiệu cần quan sát và chỉ đề xuất bước
 ↪  tiếp theo.
Không sửa tệp hoặc đổi phiên bản khi chưa xác định bước thất
 ↪  bại.
```

Prompt này giữ phạm vi ở một chuỗi thao tác đã học và yêu cầu AI chỉ ra bằng chứng trước khi đề xuất thay đổi. Khi sử dụng, cần thay tình trạng mô tả bằng kết quả thực tế, chẳng hạn thông báo đầu tiên trong cửa sổ Build hoặc trạng thái từ `adb devices`; tên gói, Activity và lệnh do AI nêu vẫn phải được đối chiếu với dự án đang mở.

Sau khi chuẩn bị câu hỏi, người học mở panel `Agent` để nhập prompt. Hình 1.18 minh họa vị trí panel trước khi đăng nhập; trạng thái này chỉ giúp nhận diện công cụ, chưa phải bằng chứng rằng dự án đã build hoặc chạy thành công.

Khi đã đăng nhập theo chính sách lớp học, người học có thể gửi prompt mẫu trong chính panel này, nhưng vẫn phải quay lại cửa sổ Build, trạng thái thiết bị và màn hình ứng dụng để kiểm chứng câu trả lời.



Hình 1.18. *Panel Tác tử của Gemini trước khi đăng nhập*

1.7.3. Kiểm chứng và giới hạn sử dụng AI

Sau mỗi đề xuất, người học dùng lại chuỗi build–cài đặt–khởi chạy ở mục trước và lưu kết quả cần thiết. Một đề xuất chỉ được chấp nhận khi cho biết nó giải quyết vấn đề nào, thay đổi gì và bằng chứng nào xác nhận kết quả.

Tổng kết

Bài học đã đi từ môi trường phát triển đến quy trình tạo và chạy ứng dụng Android đầu tiên. Người học nhận diện vai trò của công cụ, đọc đường đi từ mã Kotlin tới giao diện và phân biệt kiểm tra một quy tắc với kiểm tra toàn bộ chuỗi chạy. Điều cốt lõi là xác định ứng dụng đã đến bước nào: tạo được gói cài đặt, kết nối thiết bị, mở đúng màn hình và quan sát kết quả. Khi gặp lỗi, cần tìm mắt xích đầu tiên chưa có bằng chứng trước khi sửa mã hoặc cấu hình. Android thuần là lựa chọn thực hành để rèn cách suy luận đó; các chương sau mở rộng sang giao diện, dữ liệu, trạng thái, điều hướng, lưu trữ và kết nối mạng.

Trước Chương 2, người học nên tự thực hiện lại quy trình không phụ thuộc ảnh minh họa: mở đúng dự án, xác định mô-đun `app`, kiểm tra thiết bị, build, cài đặt và mở `Chapter1FirstAppActivity`. Nếu thất bại, cần chỉ ra lớp công cụ liên quan và bằng chứng còn thiếu. Mô tả chính xác "đã thành công đến đâu" quan trọng hơn việc thử nhiều cách sửa cùng lúc.

Câu hỏi ôn tập

1. Vì sao ứng dụng di động cần được thiết kế khác chương trình máy tính để bàn hoặc dòng lệnh?
2. Android Studio, SDK, Gradle, ADB và trình giả lập tham gia những bước nào trong quá trình phát triển?
3. Vì sao tạo được dự án chưa chứng minh ứng dụng chạy thành công? Nêu bằng

chứng cho các bước build, cài đặt và mở Activity.

4. Chế độ hiển thị Android có những nhóm tệp chính nào và vai trò của từng nhóm là gì?
5. Kotlin biểu diễn và xử lý dữ liệu có thể rỗng như thế nào?
6. Trình tự nào giúp phân biệt lỗi Gradle, ADB và lỗi khởi chạy Activity?
7. Kiểm thử đơn vị cục bộ và kiểm thử nhanh cung cấp những bằng chứng khác nhau nào?
8. Cần dùng bằng chứng nào để kiểm chứng đề xuất của công cụ AI trước khi áp dụng?
9. Trong vòng đời Activity, `onPause()` và `onResume()` biểu diễn hai chiều chuyển trạng thái nào?
10. Vì sao `MainActivity.kt` của dự án vừa tạo có thể khác kho mã đồng hành, và cần làm gì để chứng minh đúng Activity lời chào của Chương 1 đã chạy?

Bài tập thực hành

Các bài tập sau sử dụng kiến thức của toàn chương nhưng không cung cấp lại quy trình tuần tự. Trước khi thao tác, sinh viên cần ghi kết quả dự kiến; sau khi hoàn thành, hồ sơ nộp phải gồm phần đã thay đổi, bằng chứng trước–sau và giải thích kết quả thực tế. Nếu phải dừng vì điều kiện phòng máy, cần nêu bước cuối đã thành công và bằng chứng còn thiếu.

1. **Lập trình độc lập.** Tạo một dự án Android mới có tên, tên gói và lời chào khác dự án mẫu. Ghi bốn trường cấu hình và lựa chọn phiên bản Android tối thiểu. Thay đổi một giá trị truyền vào Composable, build dự án rồi nộp dòng `BUILD SUCCESSFUL`, tên thiết bị và ảnh màn hình tương ứng.
2. **Lập trình độc lập.** Bổ sung vào Activity lời chào thông tin lấy từ một hàm Kotlin. Hàm phải có đầu vào, kết quả trả về và cách xử lý trường hợp chưa có dữ liệu. Dự đoán hai chuỗi kết quả trước khi chạy; sau đó đối chiếu với giao diện và giải thích vai trò của `val`, tham số cùng giá trị trả về.
3. **Sửa lỗi môi trường.** Nhận một dự án có JDK hoặc Gradle Wrapper không tương thích. Xác định lỗi đầu tiên, giải thích vì sao chưa nên sửa giao diện và đề xuất thay đổi nhỏ nhất. Chỉ thay một yếu tố mỗi lần, chạy lại đúng tác vụ rồi lập bảng giả định, thao tác, kết quả và bằng chứng.
4. **Sửa lỗi thiết bị.** Xử lý một trường hợp thiết bị xuất hiện ở trạng thái `offline` hoặc `unauthorized`. Trước hết giải thích ý nghĩa của trạng thái quan sát được và chọn bước xử lý phù hợp; không sửa mã giao diện. Hồ sơ hoàn thành phải chứng minh thiết bị trở lại trạng thái `device`, APK được cài và đúng Activity được mở.
5. **Tích hợp.** Tự thực hiện chuỗi build–cài đặt–khởi chạy–quan sát cho `Chapter1FirstAppActivity` mà không dùng ảnh hướng dẫn. Ghi kết quả mong đợi và lưu đúng bằng chứng sau mỗi bước. Hồ sơ phải chỉ ra tệp đầu vào, thao tác và cách từng bằng chứng nối với bước tiếp theo.