Cau 1:

Android:

- **Hệ điều** hành: Android, được phát triển bởi Google.
- **Thiết bị**: Phổ biến trên các điện thoại và máy tính bảng từ nhiều hãng khác nhau như Samsung, Xiaomi, Oppo, Vivo, và nhiều hãng khác.
- **Đặc điểm**: Mã nguồn mở, cho phép tùy biến mạnh mẽ và có cửa hàng ứng dụng Google Play với hàng triệu ứng dụng.
- iOS:
    - **Hệ điều** hành: iOS, được phát triển bởi Apple.
    - **Thiết bị**: Chỉ chạy trên các thiết bị của Apple, bao gồm iPhone, iPad, iPod Touch.
    - **Đặc điểm**: Hệ sinh thái khép kín, tính bảo mật cao, và các ứng dụng được kiểm duyệt chặt chẽ qua App Store.
    - HarmonyOS:
        - **Hệ điều** hành: HarmonyOS, phát triển bởi Huawei.
        - **Thiết bị**: Chủ yếu được sử dụng trên các thiết bị của Huawei, bao gồm điện thoại, máy tính bảng, đồng hồ thông minh, TV và các thiết bị IoT.
        - **Đặc điểm**: Thiết kế để tương thích và kết nối giữa nhiều loại thiết bị khác nhau trong hệ sinh thái Huawei.

    Câu 2:

- So sánh các **nền tảng:**

Nền tảng	Phát **triển** trên	Ngôn **ngữ** chính	**Ưu điểm**	**Nhược điểm**
Android Studio	Android	Java, Kotlin	Hiệu suất cao, tích hợp Google tools	Chỉ dành cho Android, phát triển phức tạp
Xcode	iOS	Swift, Objective-C	Hiệu suất tốt, dễ sử dụng	Chỉ dành cho iOS, cần macOS
React Native	Android, iOS	JavaScript (React)	Code chung cho cả Android và iOS	Hiệu suất không bằng native, yêu cầu mã native
Flutter	Android, iOS	Dart	Hiệu suất cao, giao diện đẹp, code chung	Thư viện chưa phong phú, Dart ít phổ biến
Ionic	Android, iOS, Web	HTML, CSS, JavaScript	Webview-based apps, dễ học	Hiệu suất thấp, giao diện không tự nhiên
Xamarin	Android, iOS, Windows	C#	Hiệu suất gần native, tích hợp .NET	Cộng đồng nhỏ, cần học C#
PhoneGap	Android, iOS, Windows	HTML, CSS, JavaScript	Dễ học, code chung cho các nền tảng	Hiệu suất thấp, chậm và ít linh hoạt

Câu 3:

- 1. React Native

Phát **triển bởi**: Facebook (Meta)
Ngôn **ngữ** chính: JavaScript (với React)
Hệ điều hành **hỗ trợ**: Android, iOS, Windows, macOS (thông qua các thư viện bổ trợ)
Ưu điểm:

- Code chung cho Android và iOS: React Native cho phép viết ứng dụng cho cả Android và iOS từ một mã nguồn duy nhất, giúp tiết kiệm thời gian và chi phí.
- **Hỗ trợ tốt với** JavaScript và React: JavaScript là ngôn ngữ phổ biến, giúp lập trình viên web dễ dàng chuyển sang phát triển ứng dụng di động. Nếu bạn đã quen với React, việc chuyển sang React Native sẽ rất dễ dàng.
- **Gần như ứng dụng** native: React Native cho phép tích hợp mã native (Java/Kotlin cho Android, Swift/Objective-C cho iOS) khi cần tính năng đặc thù, do đó hiệu suất gần như ứng dụng native.
- **Cộng đồng lớn** và tài **liệu** phong phú: Là một trong những framework phổ biến nhất, React Native có một cộng đồng lớn, nhiều tài liệu học hỏi, và thư viện hỗ trợ.

Nhược điểm:

- **Hiệu suất** không hoàn toàn **bằng** native: Mặc dù gần như native, React Native vẫn không thể đạt được hiệu suất tối đa của các ứng dụng gốc, đặc biệt khi ứng dụng yêu cầu xử lý tài nguyên nặng (ví dụ: game hoặc ứng dụng AR/VR).
- **Phải viết** mã native trong **một số trường hợp**: Một số tính năng nâng cao hoặc thư viện có thể yêu cầu viết mã native, làm giảm tính linh hoạt của việc phát triển với một mã nguồn chung.
- **Cần kiến thức về** React và JavaScript: Dù là dễ học đối với lập trình viên web, nhưng nếu bạn chưa quen với React hoặc JavaScript, có thể gặp khó khăn trong việc phát triển.

2. Xamarin

Phát **triển bởi**: Microsoft
Ngôn **ngữ** chính: C#
Hệ điều hành **hỗ trợ**: Android, iOS, Windows, macOS
Ưu điểm:

- Code chung cho Android, iOS, và Windows: Xamarin cho phép phát triển ứng dụng cho ba nền tảng phổ biến nhất với một mã nguồn duy nhất, làm giảm chi phí và thời gian phát triển.
- **Hiệu suất gần** native: Xamarin biên dịch mã C# thành mã máy, mang lại hiệu suất gần với ứng dụng gốc. Đây là một điểm mạnh khi so với các nền tảng cross-platform khác, chẳng hạn như Ionic hay React Native.
- **Cộng đồng** và **hỗ trợ từ** Microsoft: Là sản phẩm của Microsoft, Xamarin được hỗ trợ mạnh mẽ trong hệ sinh thái .NET và Visual Studio. Cộng đồng cũng đang phát triển nhanh chóng.
- **Dễ** dàng tích **hợp với** các API **của nền tảng**: Xamarin cho phép truy cập dễ dàng vào các API và tính năng đặc thù của Android và iOS thông qua các thư viện hỗ trợ (Xamarin.Android, Xamarin.iOS).

Nhược điểm:

- **Cộng đồng nhỏ hơn** React Native: Mặc dù Xamarin có cộng đồng mạnh mẽ trong hệ sinh thái .NET, nhưng nó không lớn bằng React Native, do đó việc tìm kiếm tài liệu và hỗ trợ từ cộng đồng có thể gặp khó khăn hơn.
- Yêu **cầu kiến thức về** C# và .NET: Đối với các lập trình viên không quen với C# hoặc .NET, việc học Xamarin có thể khó khăn hơn so với học JavaScript hoặc React.
- Kích **thước ứng dụng lớn**: Các ứng dụng được phát triển bằng Xamarin thường có kích thước khá lớn vì phải bao gồm các thư viện và mã của Xamarin trong ứng dụng.

Câu 4:

1. Java

- **Giới thiệu**: Java là ngôn ngữ lập trình chính và lâu đời nhất được sử dụng để phát triển ứng dụng Android. Trước khi Kotlin trở nên phổ biến, Java là ngôn ngữ chính được Google sử dụng cho Android.
- **Ưu điểm**:
    - **Cộng đồng lớn** và tài **liệu** phong phú.
    - **Được hỗ trợ** chính **thức** bởi Android SDK.
    - **Chạy** trên **nền tảng** JVM (Java Virtual Machine), giúp ứng dụng có thể chạy trên các hệ điều hành khác ngoài Android.
- **Nhược điểm**:
    - **Dễ** gây **lỗi**: Java có thể khá phức tạp, dễ gây lỗi, đặc biệt là trong việc quản lý bộ nhớ.
    - Không **tối ưu** cho **lập** trình **hiện đại**: Cấu trúc mã nguồn có thể cồng kềnh, thiếu tính hiện đại và linh hoạt so với các ngôn ngữ mới như Kotlin.

2. Kotlin

- **Giới thiệu**: Kotlin là ngôn ngữ lập trình được Google chính thức hỗ trợ trên Android từ năm 2017, và hiện nay đã trở thành ngôn ngữ chính trong phát triển ứng dụng Android.
- **Ưu điểm**:
    - **Ngắn gọn** và **dễ đọc**: Cú pháp của Kotlin ngắn gọn và dễ hiểu hơn so với Java.
    - Tính an toàn và **mạnh mẽ**: Kotlin hỗ trợ tính năng an toàn với null (null safety), giúp tránh được lỗi phổ biến khi làm việc với giá trị null.
    - **Tương** thích hoàn toàn **với** Java: Kotlin có thể tích hợp dễ dàng với mã Java hiện có trong dự án Android.
    - **Hỗ trợ** tính **năng hiện đại**: Kotlin cung cấp nhiều tính năng hiện đại như lambda, extension functions, và coroutines, giúp cải thiện hiệu suất và khả năng đọc mã nguồn.
- **Nhược điểm**:

- **Học hỏi** thêm: Đối với lập trình viên Java, sẽ cần một thời gian để làm quen với cú pháp và các tính năng mới của Kotlin.

3. C++

- **Giới thiệu**: C++ có thể được sử dụng trong phát triển Android thông qua Android NDK (Native Development Kit), cho phép viết mã C/C++ để tối ưu hóa các phần cần hiệu suất cao trong ứng dụng Android.
- **Ưu điểm**:
  - **Hiệu suất** cao: C++ cung cấp hiệu suất cực kỳ nhanh và tối ưu, rất hữu ích cho các ứng dụng yêu cầu tính toán phức tạp hoặc game.
  - **Sử dụng lại** mã **nguồn**: C++ có thể sử dụng lại mã nguồn cho các nền tảng khác ngoài Android, như iOS và các hệ điều hành khác.
- **Nhược điểm**:
  - Khó **khăn** khi phát **triển ứng dụng** Android: C++ không phải là ngôn ngữ chính cho phát triển ứng dụng Android, và việc phát triển với NDK có thể phức tạp hơn so với Java/Kotlin.
  - **Quản** lý **bộ nhớ thủ** công: C++ yêu cầu lập trình viên quản lý bộ nhớ thủ công, điều này dễ gây ra các lỗi khó phát hiện như rò rỉ bộ nhớ.

4. Dart (cho Flutter)

- **Giới thiệu**: Dart là ngôn ngữ lập trình chính được sử dụng trong Flutter, framework phát triển ứng dụng di động cross-platform của Google.
- **Ưu điểm**:
  - Phát **triển ứng dụng** cross-platform: Dart giúp viết ứng dụng cho cả Android và iOS với một mã nguồn duy nhất.
  - **Hiệu suất tốt**: Các ứng dụng Flutter biên dịch trực tiếp thành mã máy, giúp mang lại hiệu suất gần như native.

    - **Hỗ trợ mạnh mẽ từ** Google: Được phát triển và duy trì bởi Google, Dart được tích hợp tốt với Flutter và hệ sinh thái của Google.
- **Nhược điểm**:
    - **Cộng đồng nhỏ hơn** Java/Kotlin: Dart không phổ biến như Java hay Kotlin, nên có thể khó tìm tài liệu học và giải pháp khi gặp vấn đề.
    - **Kỹ năng** Dart yêu **cầu học hỏi**: Nếu bạn chưa quen với Dart, việc học và sử dụng nó có thể mất thời gian.

5. Python (Thông qua Kivy **hoặc** BeeWare)

- **Giới thiệu**: Python không phải là ngôn ngữ chính để phát triển ứng dụng Android, nhưng có thể sử dụng các framework như Kivy hoặc BeeWare để tạo ứng dụng Android.
- **Ưu điểm**:
    - **Dễ học** và phát **triển** nhanh: Python là ngôn ngữ dễ học, có cú pháp rõ ràng, rất phù hợp với người mới bắt đầu.
    - **Hỗ trợ** cross-platform: Python có thể phát triển ứng dụng cho nhiều nền tảng ngoài Android.
- **Nhược điểm**:
    - **Hiệu suất** kém: So với Java, Kotlin hoặc C++, Python có hiệu suất thấp hơn đáng kể, không phù hợp cho ứng dụng yêu cầu tài nguyên nặng hoặc xử lý nhanh.
    - Không **phổ biến** trong phát **triển ứng dụng** Android: Python không phải là ngôn ngữ chính cho phát triển Android, vì vậy tài liệu và công cụ hỗ trợ rất hạn chế.

6. JavaScript (Thông qua React Native)

- **Giới thiệu**: React Native là một framework phát triển ứng dụng di động sử dụng JavaScript, cho phép phát triển ứng dụng Android và iOS với một mã nguồn duy nhất.
- **Ưu điểm**:
    - Phát **triển** nhanh: JavaScript là ngôn ngữ phổ biến, và React Native giúp xây dựng ứng dụng với hiệu quả cao và ít phải viết mã cho từng nền tảng.

    - **Cộng đồng mạnh mẽ**: React Native có một cộng đồng lớn và tài liệu phong phú.
- **Nhược điểm**:
    - **Hiệu suất** không **tối ưu như** native: Mặc dù gần giống với ứng dụng native, hiệu suất của React Native vẫn không thể sánh bằng ứng dụng gốc khi cần xử lý các tác vụ phức tạp hoặc yêu cầu tài nguyên cao.

Câu 5:

1. Swift

- **Giới thiệu**: Swift là ngôn ngữ lập trình hiện đại, được Apple phát triển và công bố vào năm 2014. Đây là ngôn ngữ chính được khuyến khích và sử dụng để phát triển ứng dụng iOS, macOS, watchOS và tvOS.
- **Ưu điểm**:
    - **Cộng đồng mạnh mẽ** và tài **liệu** phong phú: Là ngôn ngữ chính của Apple, Swift có cộng đồng phát triển lớn và được hỗ trợ rất tốt với nhiều tài liệu học và hướng dẫn.
    - **Hiệu suất** cao: Swift được thiết kế để nhanh chóng và hiệu quả, cho phép tạo ra các ứng dụng iOS có hiệu suất cao.
    - **Dễ học** và **sử dụng**: Swift có cú pháp đơn giản, dễ hiểu và dễ duy trì hơn so với Objective-C.
    - Tính an toàn: Swift cung cấp các tính năng bảo vệ lỗi mạnh mẽ, như xử lý nil (null safety) và kiểu dữ liệu an toàn.
    - Tích **hợp** hoàn **hảo với** Xcode: Swift và Xcode (IDE chính của Apple) được tối ưu hóa chặt chẽ với nhau, giúp lập trình viên phát triển ứng dụng nhanh chóng và hiệu quả.
- **Nhược điểm**:
    - **Tương đối mới**: Mặc dù phát triển nhanh chóng, nhưng Swift vẫn có thể gặp phải một số hạn chế về

tính tương thích ngược hoặc một số vấn đề chưa được giải quyết hoàn toàn so với Objective-C.

2. Objective-C

- **Giới thiệu**: Objective-C là ngôn ngữ lập trình cũ và là ngôn ngữ chính được sử dụng cho việc phát triển ứng dụng iOS trước khi Swift ra đời. Mặc dù hiện nay Swift là ngôn ngữ chính, Objective-C vẫn được hỗ trợ và sử dụng rộng rãi trong các ứng dụng và dự án cũ.
- **Ưu điểm**:
  - **Cộng đồng** và tài **liệu** lâu dài: Objective-C có lịch sử lâu dài và cộng đồng rộng lớn, vì vậy tài liệu học tập và nguồn lực hỗ trợ rất phong phú.
  - **Tương** thích **ngược với** C: Objective-C có thể tích hợp tốt với mã C, làm cho nó phù hợp cho các ứng dụng yêu cầu tính toán hiệu suất cao.
  - **Hỗ trợ tuyệt vời từ** Apple: Apple vẫn duy trì và hỗ trợ Objective-C, và nhiều API của iOS vẫn có sẵn bằng ngôn ngữ này.
- **Nhược điểm**:
  - Cú pháp **phức tạp**: Cú pháp của Objective-C có thể khó hiểu đối với người mới, đặc biệt là các dấu ngoặc và cú pháp "mét" (message sending).
  - Không **hiện đại bằng** Swift: Mặc dù hiệu quả, Objective-C không có nhiều tính năng hiện đại và dễ sử dụng như Swift.

3. C++

- **Giới thiệu**: C++ có thể được sử dụng để phát triển ứng dụng iOS thông qua Objective-C++, kết hợp mã C++ và Objective-C, đặc biệt khi cần tối ưu hóa hiệu suất hoặc phát triển các tính năng đặc biệt như game, đồ họa, hoặc các ứng dụng cần xử lý tài nguyên nặng.
- **Ưu điểm**:
  - **Hiệu suất** cao: C++ cho phép lập trình viên có sự kiểm soát chi tiết hơn đối với bộ nhớ và tài nguyên, thích hợp cho các ứng dụng yêu cầu hiệu suất cao.

    - Tích hợp mã với Objective-C: C++ có thể được tích hợp vào ứng dụng iOS qua Objective-C++, giúp kết hợp tốt giữa khả năng của C++ và API của iOS.
- **Nhược điểm**:
    - Khó học và phát triển: C++ yêu cầu lập trình viên quản lý bộ nhớ thủ công và có cú pháp phức tạp hơn nhiều so với Swift và Objective-C.
    - Không phổ biến cho iOS: C++ không phải là ngôn ngữ chính cho phát triển iOS, vì vậy việc phát triển ứng dụng với C++ có thể khó khăn và không linh hoạt như Swift hay Objective-C.

4. Python (Thông qua Kivy hoặc BeeWare)

- **Giới thiệu**: Python không phải là ngôn ngữ chính để phát triển ứng dụng iOS, nhưng có thể được sử dụng thông qua các framework như Kivy hoặc BeeWare.
- **Ưu điểm**:
    - Dễ học và phát triển nhanh: Python rất dễ học và có cú pháp rõ ràng, phù hợp cho những lập trình viên mới bắt đầu.
    - Cross-platform: Python có thể phát triển ứng dụng cho nhiều nền tảng ngoài iOS, bao gồm Android, Windows, Linux và web.
- **Nhược điểm**:
    - **Hiệu suất thấp**: Python không có hiệu suất tốt bằng Swift hoặc Objective-C và không phù hợp cho các ứng dụng yêu cầu tài nguyên nặng hoặc xử lý nhanh.
    - Không được Apple chính thức hỗ trợ: Python không phải là ngôn ngữ được Apple chính thức hỗ trợ cho phát triển ứng dụng iOS, và các công cụ như Kivy hoặc BeeWare vẫn còn thiếu tính hoàn thiện và hỗ trợ tốt.

5. JavaScript (Thông qua React Native hoặc Cordova)

- **Giới thiệu**: JavaScript có thể được sử dụng để phát triển ứng dụng iOS thông qua các framework cross-platform như React Native hoặc Apache Cordova.

- **Ưu điểm**:
  - Phát **triển** nhanh và **tiết kiệm thời** gian: React Native cho phép phát triển ứng dụng cho cả Android và iOS từ một mã nguồn duy nhất.
  - **Cộng đồng mạnh mẽ**: JavaScript có cộng đồng rất lớn, giúp lập trình viên dễ dàng tìm kiếm tài liệu và giải pháp khi gặp vấn đề.
- **Nhược điểm**:
  - **Hiệu suất** không **tối ưu như** native: Mặc dù React Native giúp tạo ứng dụng cross-platform, nhưng hiệu suất của ứng dụng không thể đạt được như ứng dụng native viết bằng Swift hoặc Objective-C.
  - Yêu **cầu kiến thức về** JavaScript và React: Nếu bạn chưa quen với JavaScript hoặc React, bạn sẽ cần một thời gian để làm que.

Câu 6 :

1. **Thiếu ứng dụng** và **sự hỗ trợ của** nhà phát **triển**

- Thách **thức**: Một trong những yếu tố quan trọng nhất ảnh hưởng đến sự phát triển của Windows Phone là việc thiếu ứng dụng phổ biến và không đủ sự hỗ trợ từ các nhà phát triển ứng dụng. Mặc dù Windows Phone có cửa hàng ứng dụng (Windows Store), nhưng số lượng ứng dụng sẵn có không đủ lớn so với Google Play (Android) và App Store (iOS).
- Nguyên nhân: Các nhà phát triển không đầu tư vào nền tảng Windows Phone do thị phần của hệ điều hành này quá nhỏ. Họ chủ yếu tập trung phát triển ứng dụng cho Android và iOS, nơi có lượng người dùng lớn và tiềm năng lợi nhuận cao hơn. Điều này tạo thành một vòng lặp tiêu cực, nơi thiếu ứng dụng làm giảm sức hấp dẫn của hệ điều hành, dẫn đến ít người dùng, và càng ít nhà phát triển tham gia.

2. **Chậm cải tiến** và **thiếu sự đổi mới**

- Thách **thức**: Mặc dù Windows Phone có giao diện độc đáo và nhiều tính năng sáng tạo, nhưng nó không thể duy trì sự cạnh tranh với Android và iOS trong việc cung cấp tính năng mới và cải tiến. Các bản cập nhật của Windows Phone thường đến muộn và thiếu đột phá, khiến nền tảng này không thể thu hút người dùng mới.
- Nguyên nhân: Microsoft đã gặp khó khăn trong việc phát triển và cải tiến hệ điều hành của mình một cách nhanh chóng. Windows Phone ra đời khá muộn so với Android và iOS, khiến nó phải bắt kịp với hai nền tảng đã có thâm niên lâu dài. Trong khi Google và Apple liên tục cải tiến hệ điều hành của mình với các tính năng mới, Microsoft không thể nhanh chóng bắt kịp và không thể tạo ra những thay đổi đột phá để thu hút người dùng.

3. Chính sách **phần cứng** và **hợp** tác **với** các nhà **sản xuất**

- Thách **thức**: Microsoft chủ yếu hợp tác với một số ít các nhà sản xuất phần cứng, nổi bật nhất là Nokia (mua lại vào năm 2014) để sản xuất các thiết bị chạy Windows Phone. Điều này hạn chế sự đa dạng và sự sáng tạo trong thiết kế của các sản phẩm.
- Nguyên nhân: Chính sách "Windows Phone-only" khiến Microsoft phải phụ thuộc vào một số nhà sản xuất và khiến người tiêu dùng có ít sự lựa chọn hơn so với Android, nơi có hàng trăm nhà sản xuất thiết bị. Mặc dù Nokia đã sản xuất nhiều thiết bị chất lượng cao, nhưng người dùng lại thiếu sự đa dạng trong lựa chọn điện thoại và mức giá. Điều này khiến Windows Phone không thể thu hút được một lượng lớn người dùng.

4. **Chậm trễ** trong **việc hỗ trợ** các tính **năng** quan **trọng**

- Thách **thức**: Windows Phone không hỗ trợ nhiều tính năng quan trọng mà người dùng mong đợi như multitasking, app notifications, hoặc các tính năng tùy chỉnh khác mà iOS và Android đã có từ lâu.
- Nguyên nhân: Microsoft không thể cập nhật các tính năng này đủ nhanh hoặc đầy đủ, khiến hệ điều hành này có vẻ lỗi

thời và không hấp dẫn người dùng. Điều này đặc biệt rõ rệt khi Apple và Google luôn đổi mới và cung cấp các tính năng mới trong các bản cập nhật phần mềm của họ.

5. **Định hướng** sai **lầm với hệ** sinh thái

- Thách **thức**: Microsoft có một hệ sinh thái phần mềm rất mạnh, bao gồm các sản phẩm như Windows, Office, và các dịch vụ đám mây, nhưng lại gặp khó khăn trong việc kết nối và đồng bộ hóa giữa các hệ thống này với Windows Phone một cách hiệu quả.
- Nguyên nhân: Mặc dù Microsoft đã nỗ lực tích hợp các dịch vụ của mình vào Windows Phone, nhưng các ứng dụng và dịch vụ của đối thủ như Google và Apple lại mạnh hơn nhiều. Người dùng không thấy lý do rõ ràng để chuyển sang sử dụng Windows Phone khi họ có thể sử dụng Android hoặc iPhone với tất cả các dịch vụ và ứng dụng của Google hoặc Apple đã có sẵn.

6. **Quản** lý và **chiến lược** không **nhất** quán

- Thách **thức**: Microsoft không có một chiến lược dài hạn và rõ ràng đối với Windows Phone. Đặc biệt, khi Microsoft quyết định từ bỏ Windows Phone vào năm 2017, sự không chắc chắn và sự thiếu định hướng trong chiến lược của họ đã làm mất niềm tin của người dùng và các nhà phát triển.
- Nguyên nhân: Các thay đổi liên tục trong chiến lược và việc chuyển hướng sang các nền tảng khác đã làm giảm tính ổn định của Windows Phone. Microsoft không thể duy trì một chiến lược thống nhất, đặc biệt là sau khi mua lại Nokia và tìm cách kết hợp hệ điều hành của mình với các thiết bị phần cứng mới. Những quyết định này gây nhầm lẫn cho người dùng và các nhà phát triển, làm cho Windows Phone trở nên không còn hấp dẫn.

7. **Sự thống trị của** Android và iOS

- Thách **thức**: Android và iOS đã thiết lập được sự thống trị tuyệt đối trên thị trường smartphone. Cả hai hệ điều hành này có hệ sinh thái ứng dụng phong phú, sự hỗ trợ mạnh mẽ

từ các nhà phát triển và các tính năng mà người dùng yêu cầu.

- Nguyên nhân: Khi Android và iOS trở nên phổ biến và gần như chiếm lĩnh thị trường, Windows Phone không thể cạnh tranh được với sự lớn mạnh của hai hệ sinh thái này. Microsoft không thể vượt qua sự cạnh tranh gay gắt từ Google và Apple, khiến thị phần của Windows Phone bị thu hẹp dần.

Câu 7:

1. HTML, CSS và JavaScript

HTML, CSS và JavaScript là bộ ba ngôn ngữ cơ bản trong phát triển ứng dụng web di động. Chúng là nền tảng cho tất cả các ứng dụng web, bao gồm cả các ứng dụng di động.

- HTML (HyperText Markup Language): Được sử dụng để xây dựng cấu trúc của trang web. Trong phát triển ứng dụng di động, HTML5 giúp cung cấp các tính năng đặc biệt như lưu trữ offline, video và âm thanh, và các API cho các thiết bị di động.
- CSS (Cascading Style Sheets): Quản lý giao diện và bố cục của trang web. CSS3 cho phép tạo các hiệu ứng động, transitions, animations, và đặc biệt là hỗ trợ responsive design, giúp ứng dụng web hiển thị đẹp trên mọi kích thước màn hình di động.
- JavaScript: Là ngôn ngữ lập trình chủ yếu được sử dụng để tương tác với người dùng và thực hiện các tác vụ động trên trang web. JavaScript giúp làm việc với các API của trình duyệt, thao tác với DOM, và hỗ trợ các chức năng phức tạp trên thiết bị di động. ES6 và các tính năng mới trong JavaScript giúp cải thiện hiệu suất và khả năng lập trình.

Ưu điểm:

- Được sử dụng rộng rãi, dễ tiếp cận và hỗ trợ trên tất cả các trình duyệt di động.
- Hỗ trợ responsive design, giúp ứng dụng hiển thị tốt trên các loại thiết bị di động khác nhau.

Nhược điểm:

- Các ứng dụng web truyền thống có thể không sử dụng tối đa các tính năng phần cứng của điện thoại như camera, cảm biến vân tay, v.v. (Tuy nhiên, có thể cải thiện bằng cách sử dụng các framework hoặc API khác).

2. Frameworks và **Thư viện** JavaScript cho **Ứng dụng** Web Di **Động**

Để phát triển ứng dụng web di động hiệu quả và nhanh chóng, các framework và thư viện JavaScript mạnh mẽ là một phần quan trọng. Các công cụ này giúp giảm bớt công sức lập trình và tối ưu hóa trải nghiệm người dùng.

React (ReactJS & React Native)

- React là một thư viện JavaScript được phát triển bởi Facebook, giúp tạo giao diện người dùng (UI) dễ dàng và hiệu quả. Với React, bạn có thể xây dựng các ứng dụng web di động responsive với khả năng tái sử dụng các thành phần giao diện.
- React Native là một framework phát triển ứng dụng di động cross-platform (Android và iOS) sử dụng JavaScript, giúp bạn viết ứng dụng di động mà không cần phải biết Java hoặc Swift. Tuy nhiên, với ReactJS, bạn có thể xây dựng các ứng dụng web di động tuyệt vời.

Ưu điểm:

- Cộng đồng lớn và tài liệu phong phú.
- Phát triển nhanh chóng và dễ bảo trì.
- Tối ưu hóa hiệu suất và khả năng tái sử dụng mã nguồn.

Nhược điểm:

- Cần có kiến thức về JSX và cách quản lý trạng thái trong ứng dụng.

Vue.js

- Vue.js là một framework JavaScript nhẹ nhàng và linh hoạt, giúp xây dựng giao diện người dùng. Vue cho phép bạn phát triển các ứng dụng web di động responsive và nhanh chóng với cấu trúc mã nguồn dễ dàng duy trì.

Ưu điểm:

- Dễ học và dễ tích hợp với các dự án hiện có.
- Cung cấp khả năng quản lý trạng thái mạnh mẽ với Vuex.

Nhược điểm:

- Dù rất mạnh mẽ, Vue vẫn không phổ biến như React, và có ít tài liệu hơn.

Angular

- Angular là một framework phát triển ứng dụng web mạnh mẽ, được phát triển bởi Google. Angular giúp xây dựng các ứng dụng web phức tạp, động và responsive, phù hợp với các ứng dụng di động.

Ưu điểm:

- Tích hợp tốt với TypeScript.
- Quản lý mã nguồn mạnh mẽ với các mô hình MVC (Model-View-Controller).

Nhược điểm:

- Cấu trúc phức tạp hơn so với React và Vue, có thể làm khó người mới bắt đầu.

3. Progressive Web Apps (PWAs)

- PWAs là các ứng dụng web có khả năng hoạt động giống như ứng dụng di động, bao gồm khả năng tải nhanh, hỗ trợ offline, và có thể cài đặt trên màn hình chính của thiết bị di động. PWAs sử dụng các công nghệ web hiện đại (HTML5, CSS3, và JavaScript) để cung cấp trải nghiệm gần giống với ứng dụng native.

Ưu điểm:

- Không cần phải phát triển ứng dụng riêng cho Android hoặc iOS.
- Hỗ trợ offline và khả năng thông báo đẩy.
- Dễ dàng cài đặt và cập nhật.

Nhược điểm:

- Không thể truy cập đầy đủ các tính năng phần cứng của thiết bị như ứng dụng native (như cảm biến vân tay, Bluetooth, camera, v.v. - mặc dù có thể cải thiện qua API).

4. Cordova/PhoneGap

- Apache Cordova (trước đây là PhoneGap) là một framework mã nguồn mở cho phép phát triển ứng dụng di động bằng cách sử dụng HTML, CSS và JavaScript. Cordova tạo một lớp bao bọc (wrapper) giữa ứng dụng web và các API của thiết bị, giúp truy cập các tính năng phần cứng của điện thoại di động như camera, GPS, và bộ nhớ.

Ưu điểm:

- Phát triển ứng dụng di động đa nền tảng (Android, iOS) từ một mã nguồn duy nhất.
- Có thể tích hợp các plugin để truy cập các tính năng phần cứng của thiết bị.

Nhược điểm:

- Hiệu suất không tốt như các ứng dụng native, đặc biệt với các ứng dụng có yêu cầu xử lý đồ họa hoặc hiệu suất cao.
- Các plugin có thể không hoàn toàn tương thích hoặc không được duy trì tốt.

5. Flutter (cho Web)

- Flutter là một framework phát triển ứng dụng di động của Google, ban đầu được thiết kế để phát triển ứng dụng native cho Android và iOS. Tuy nhiên, với sự hỗ trợ của Flutter Web, bạn có thể phát triển ứng dụng web, bao gồm các ứng dụng web di động, với giao diện người dùng đẹp mắt và hiệu suất cao.

Ưu điểm:

- Phát triển ứng dụng di động và web từ cùng một mã nguồn.
- Tạo giao diện người dùng đẹp mắt và mượt mà.

Nhược điểm:

- Tối ưu hóa cho web không hoàn toàn mạnh mẽ như trên di động.
- Cộng đồng web của Flutter chưa phát triển mạnh mẽ như các framework web khác.

6. Bootstrap

- Bootstrap là một framework CSS phổ biến, hỗ trợ thiết kế giao diện người dùng (UI) responsive và mobile-first. Nó cung cấp một bộ công cụ phong phú, bao gồm các thành phần giao diện người dùng sẵn có như nút, bảng, biểu mẫu, v.v., giúp tạo các ứng dụng web tương thích với di động dễ dàng.

Ưu điểm:

- Dễ sử dụng và học, giúp thiết kế giao diện nhanh chóng.
- Tích hợp tốt với các ngôn ngữ web cơ bản như HTML và CSS.

Nhược điểm:

- Thiết kế có thể trông giống nhau giữa các ứng dụng, thiếu tính sáng tạo nếu không tùy chỉnh.

Câu 8:

1. **Kỹ năng lập** trình Native (Android & iOS)

- Android:
  - Java và Kotlin: Java từng là ngôn ngữ chủ yếu để phát triển ứng dụng Android, nhưng hiện nay Kotlin được coi là ngôn ngữ chính và được Google khuyến khích sử dụng do tính hiện đại và dễ sử dụng hơn.
  - Android SDK: Kiến thức vững về Android SDK (Software Development Kit) và các công cụ như Android Studio, Gradle là rất cần thiết để phát triển ứng dụng Android.

- iOS:
    - Swift và Objective-C: Swift là ngôn ngữ chủ yếu để phát triển ứng dụng iOS hiện đại, thay thế Objective-Ctrong hầu hết các dự án mới. Kiến thức về cả hai ngôn ngữ này sẽ rất hữu ích, đặc biệt khi làm việc với các dự án cũ.
    - Xcode: Là công cụ phát triển chính cho iOS, lập trình viên iOS cần có khả năng sử dụng thành thạo Xcode để lập trình, thử nghiệm và triển khai ứng dụng.

2. **Kỹ năng lập** trình Cross-Platform

- React Native: Đây là một framework mạnh mẽ giúp lập trình viên viết ứng dụng di động cho cả Android và iOS từ một mã nguồn duy nhất. Kỹ năng JavaScript và React là cần thiết để phát triển ứng dụng bằng React Native.
- Flutter: Được phát triển bởi Google, Flutter cho phép viết ứng dụng di động nhanh chóng và hiệu quả với Dart. Kỹ năng Flutter đang ngày càng được yêu cầu nhiều vì khả năng tạo ra ứng dụng đẹp và nhanh trên cả hai nền tảng.
- Xamarin: Là một framework của Microsoft, Xamarin cho phép lập trình viên viết ứng dụng di động bằng C# và .NET cho cả Android và iOS.

3. UI/UX Design

- **Thiết kế** giao **diện người** dùng (UI) và **trải nghiệm người** dùng (UX) là rất quan trọng khi phát triển ứng dụng di động, vì trải nghiệm người dùng sẽ quyết định thành công của ứng dụng. Lập trình viên di động không chỉ cần hiểu các nguyên lý lập trình mà còn cần có khả năng thiết kế giao diện mượt mà, dễ sử dụng.
- Các công **cụ thiết kế** như Sketch, Figma, Adobe XD sẽ là những công cụ hữu ích giúp lập trình viên phối hợp với các nhà thiết kế để tạo ra các ứng dụng di động có giao diện bắt mắt.

4. API và Backend Development

- Kỹ năng về API (Application Programming Interface) rất quan trọng vì hầu hết các ứng dụng di động cần kết nối với backend để truy xuất dữ liệu. Lập trình viên cần có hiểu biết về các loại API như RESTful API, GraphQL, và cách tích hợp chúng vào ứng dụng di động.
- Kiến thức về backend và các dịch vụ đám mây như Firebase, AWS, Google Cloud hoặc Azure cũng sẽ giúp lập trình viên di động xây dựng các ứng dụng mạnh mẽ hơn.

5. **Kiến thức về** Testing và Debugging

- Unit testing và UI testing rất quan trọng trong phát triển ứng dụng di động để đảm bảo ứng dụng hoạt động ổn định và không có lỗi.
- Các công cụ như JUnit, Espresso cho Android, và XCTest cho iOS là các công cụ thường được sử dụng để kiểm thử ứng dụng di động.

6. **Kiến thức về** Data Storage và Caching

- SQLite là cơ sở dữ liệu nhẹ được sử dụng phổ biến trên thiết bị di động để lưu trữ dữ liệu.
- Room Database (Android) và Core Data (iOS) cũng là các giải pháp lưu trữ dữ liệu quan trọng trong các ứng dụng di động.
- Các công cụ caching như Redis, Shared Preferences (Android), UserDefaults (iOS) giúp tối ưu hiệu suất ứng dụng.

7. **Tối ưu** hóa **hiệu suất** và **bảo mật**

- Lập trình viên di động cần tối ưu hóa hiệu suất của ứng dụng, bao gồm **tối ưu** hóa **bộ nhớ**, **quản** lý tài nguyên **hệ thống**, và **giảm thiểu độ trễ**.
- **Bảo mật** là một yếu tố quan trọng, đặc biệt khi ứng dụng xử lý dữ liệu nhạy cảm. Các kỹ năng bảo mật như mã hóa **dữ liệu**, **quản** lý **quyền** truy **cập** và an toàn **mạng** là cần thiết