**Pierwsze kroki z Android Studio**

W poprzedniej Części poznaliśmy historię platformy Android oraz jej główne cechy i zalety. Ta Część przeprowadzi Cię przez proces instalacji i konfiguracji środowiska programistycznego, tak abyś mógł postępować zgodnie z instrukcjami i przykładami zawartymi w tej książce. Najpierw przyjrzymy się procesowi instalacji i konfiguracji, przejrzymy skrypty Gradle Build i dowiemy się więcej o pracy z projektami Android Studio

**INSTALACJA I KONFIGURACJA**

Konfiguracja Android Studio zwykle zajmuje tylko kilka kliknięć. Przede wszystkim jednak upewnij się, że pobierasz najnowszą wersję 4.2.2 (czerwiec 2021) Android Studio z oficjalnej strony programisty: https://developer.android.com/studio.

**Windows**

Jeśli instalujesz Android Studio w systemie Windows, wykonaj następujące czynności: Jeśli pobrałeś plik .exe, kliknij dwukrotnie, aby go uruchomić. Jeśli pobrałeś plik .zip, musisz najpierw rozpakować plik ZIP, skopiować folder android-studio do folderu Program Files, a następnie otworzyć folder android-studio > bin i uruchomić studio64.exe (dla maszyn 64-bitowych) lub studio.exe (dla maszyn 32-bitowych). Możesz później skorzystać z kreatora konfiguracji w Android Studio i zainstalować inne zalecane przez niego pakiety. Gdy pojawią się nowe narzędzia i produkty, Android Studio poinformuje Cię o tym za pomocą wyskakującego powiadomienia lub zawsze możesz sprawdzić dostępność aktualizacji, klikając Pomoc > Sprawdź aktualizacje.

**Mac**

Jeśli jesteś użytkownikiem komputera Mac, proces konfiguracji jest dość prosty, ponieważ:

• Najpierw uruchom plik Android Studio DMG.

• Następnie przenieś i zlokalizuj Android Studio w folderze Aplikacje, po czym uruchom Android Studio.

• Wybierz, czy chcesz zaimportować poprzednie ustawienia Android Studio, czy nie, a następnie kliknij OK.

Kreator instalacji Android Studio poprowadzi Cię przez resztę konfiguracji, która obejmuje pobranie elementów zestawu programistycznego dla systemu Android, które są kluczowe dla zapewnienia lepszej wydajności. Gdy pojawią się nowe narzędzia i produkty, Android Studio poinformuje Cię o tym za pomocą wyskakującego powiadomienia lub zawsze możesz sprawdzić dostępność aktualizacji, otwierając Android Studio > Sprawdź aktualizacje. Dodatkowo, jeśli używasz Android Studio na MacOS Mojave, prawdopodobnie otrzymasz monit o zezwolenie zintegrowanemu środowisku programistycznemu (IDE) na przejrzenie Twojego kalendarza, kontaktów lub zdjęć. Ten monit jest wprowadzany przez nową technikę ochrony prywatności dla aplikacji, które uzyskują dostęp do plików w katalogu domowym. Tak więc, jeśli twój projekt ma pliki i biblioteki w twoim katalogu domowym i pojawi się ten monit, zawsze możesz wybrać opcję Nie zezwalaj.

**Linux**

Aby zainstalować Android Studio w systemie Linux, wykonaj następujące czynności: Rozpakuj pobrany plik .zip do odpowiedniej lokalizacji dla aplikacji, na przykład w /usr/local/ dla swojego profilu użytkownika lub /opt/ w celu udostępnienia innym. Jeśli używasz 64-bitowej wersji systemu Linux, musisz najpierw zainstalować wymagane biblioteki dla maszyn 64-bitowych. Ale jednocześnie, jeśli

używasz 64-bitowej wersji Ubuntu, powinieneś zainstalować kilka 32-bitowych bibliotek za pomocą następującego polecenia:

sudo apt-get zainstaluj libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386

Lub jeśli używasz 64-bitowej Fedory, polecenie to:

sudo mniam zainstaluj zlib.i686 ncurses-libs.i686 bzip2-libs.i686

Po uruchomieniu Android Studio otwórz terminal, przejdź do katalogu android-studio/bin/i aktywuj studio.sh. Wybierz, czy chcesz zaimportować poprzednie ustawienia Android Studio, czy nie, a następnie naciśnij OK. Kreator instalacji Android Studio przeprowadzi Cię przez resztę konfiguracji, która obejmuje pobranie kluczowych komponentów Androida, które są wymagane do rozwoju. Aby dodać Android Studio do listy aplikacji, przejdź do Narzędzia > Utwórz wpis pulpitu z paska menu Android Studio. Gdy pojawią się nowe narzędzia i produkty, Android Studio poinformuje Cię o tym za pomocą wyskakującego powiadomienia lub zawsze możesz sprawdzić dostępność aktualizacji, otwierając Pomoc > Sprawdź aktualizacje.

**System operacyjny Chrome**

Jeśli jesteś użytkownikiem Chrome OS, wykonaj następujące kroki, aby przeprowadzić pełną instalację:

• Jeśli jeszcze tego nie zrobiłeś, zainstaluj Linux dla Chrome OS.

• Otwórz aplikację Pliki i znajdź pobrany plik DEB (Pakiet oprogramowania Debiana) w folderze Pobrane w obszarze Moje pliki.

• Kliknij prawym przyciskiem myszy pakiet DEB i wybierz Zainstaluj z systemem Linux (Beta).

• Wybierz docelową lokalizację pliku pakietu DEB w systemie operacyjnym Chrome.

• Jeśli wcześniej zainstalowałeś Android Studio, wybierz, czy chcesz ponownie zaimportować podstawowe ustawienia Android Studio, czy nie, a następnie kliknij OK.

Kreator instalacji Android Studio przeprowadzi Cię przez resztę konfiguracji, która obejmuje również pobranie kluczowych komponentów programistycznych Androida. Po zakończeniu instalacji uruchom Android Studio z Launchera lub z terminala Chrome OS Linux, uruchamiając następujący plik studio.sh w domyślnym katalogu instalacyjnym:  /opt/android-studio/bin/studio.sh

Podobnie, gdy pojawią się nowe narzędzia, Android Studio poinformuje Cię o tym za pomocą wyskakującego okienka lub możesz zobaczyć te aktualizacje, klikając Pomoc > Sprawdź aktualizacje. Co więcej, Android Studio udostępnia kreatory i szablony, które wyjaśniają wymagania systemowe, takie jak Java Development Kit (JDK) i konfigurują ustawienia domyślne, takie jak zoptymalizowana domyślna emulacja Android Virtual Device (AVD) i zaktualizowane obrazy systemu. Później przyjrzymy się każdemu z tych dokumentów i dodatkowym ustawieniom konfiguracyjnym. Ale żeby przejść przez kilka podstawowych procedur konfiguracyjnych, Android Studio zapewnia dostęp do dwóch plików konfiguracyjnych poprzez menu Pomoc:

• studio.vmoptions: Służy do dostosowywania opcji wirtualnej maszyny Java (JVM) programu Studio, takich jak rozmiar sterty i rozmiar pamięci podręcznej. Zazwyczaj na komputerach z systemem Linux ten plik ma nazwę studio64.vmoptions, w zależności od używanej wersji Android Studio.

• idea.properties: Stosowany do dostosowywania właściwości Android Studio, takich jak ścieżka folderu wtyczek lub maksymalny obsługiwany rozmiar pliku. Oba pliki konfiguracyjne znajdują się w

folderze konfiguracyjnym Android Studio. Nazwa folderu zależy od wersji Studio. Oto lokalizacje dla Android Studio 4.1 i nowszych:

• Składnia systemu Windows: %APPDATA%\Google\<produkt><wersja>

Przykład: C:\Użytkownicy\NazwaUżytkownika\AppData\Roaming\Google\AndroidStudio4.1

• System operacyjny Mac

Składnia: ~/Library/Application Support/Google/<produkt><wersja>

Przykład: ~/Library/Application Support/Google/AndroidStudio4.1

• Linux

Składnia: ~/.config/Google/<produkt><wersja>

Przykład: ~/.config/Google/AndroidStudio4.1

W wersji Android Studio 4.0 i starszych pliki konfiguracyjne znajdują się w następujących miejscach:

• Windows: %USERPROFILE%\.CONFIGURATION_FOLDER

• MacOS: ~/Biblioteka/Preferencje/FOLDER_KONFIGURACJI

• Linux: ~/.FOLDER_KONFIGURACJI

Innym prostym sposobem na zlokalizowanie katalogu konfiguracyjnego jest przejście do Pomoc > Edytuj niestandardowe opcje maszyny wirtualnej w Android Studio. Ta opcja otwiera plik konfiguracyjny i pozwala sprawdzić ścieżkę pliku konfiguracyjnego, aby zobaczyć katalog konfiguracji. Możesz również użyć następujących zmiennych środowiskowych, aby dotrzeć do określonych plików zastępujących w innym miejscu:

• STUDIO_VM_OPTIONS: Ustaw nazwę i lokalizację pliku .vmoptions.

• STUDIO_PROPERTIES: Ustaw nazwę i lokalizację pliku .properties.

• STUDIO_JDK: Ustaw JDK, z którym ma być uruchamiane Studio.

**Dostosowywanie opcji maszyny wirtualnej**

Plik konfiguracyjny studio.vmoptions pozwala modyfikować opcje dla JVM Android Studio. Aby zwiększyć wydajność programu Studio, najczęstszym sposobem jest dostosowanie maksymalnego rozmiaru sterty, ale można również użyć pliku studio.vmoptions, aby zmienić inne domyślne ustawienia, takie jak początkowy rozmiar sterty, rozmiar pamięci podręcznej i przełączniki wyrzucania elementów bezużytecznych Java. Aby utworzyć nowy plik studio.vmoptions lub otworzyć istniejący, kliknij Pomoc > Edytuj niestandardowe opcje maszyny wirtualnej. Jeśli nigdy wcześniej nie edytowałeś opcji maszyny wirtualnej dla Android Studio, IDE wyświetli monit o otwarcie nowego pliku studio.vmoptions. Kliknij Tak, aby otworzyć plik. Plik studio.vmoptions będzie wtedy dostępny w oknie edytora Android Studio. Edytuj plik, aby dodać własne dostosowane opcje maszyny wirtualnej. Utworzony plik studio.vmoptions zostanie następnie dodany do domyślnego pliku studio.vmoptions, znajdującego się w bin/katalogu w folderze instalacyjnym Android Studio. Nie zaleca się jednak bezpośredniej edycji pliku studio.vmoptions znajdującego się w folderze programu Android Studio. Nawet jeśli możesz uzyskać dostęp do pliku, aby wyświetlić domyślne opcje maszyny wirtualnej Studio, edytowanie tylko własnego pliku studio.vmoptions gwarantuje, że nie zmienisz kluczowych ustawień domyślnych dla Android Studio. Dlatego w pliku studio.vmoptions zmodyfikuj tylko atrybuty, na

których Ci zależy, i zezwól Android Studio na dalsze używanie wartości domyślnych dla wszystkich elementów, których nie dotykałeś. Teraz domyślnie maksymalny rozmiar sterty Android Studio to 1280 MB. Jeśli pracujesz nad dużym projektem, a Twój system ma dużo pamięci RAM, możesz zwiększyć wydajność, zwiększając maksymalny rozmiar sterty dla procesów Android Studio, takich jak podstawowe IDE, demon Gradle i demon Kotlin. Ale jednocześnie Android Studio automatycznie sprawdzi możliwe optymalizacje rozmiaru sterty i powiadomi Cię, jeśli zauważy, że wydajność można poprawić. Jeśli używasz systemu 64-bitowego, który ma co najmniej 5 GB pamięci RAM, możesz również ręcznie zmienić rozmiary sterty projektu. Aby to zrobić, po prostu kliknij Plik > Ustawienia na pasku menu (lub Android Studio > Preferencje w systemie MacOS). Następnie przejdź do opcji Wygląd i zachowanie > Ustawienia systemu > Ustawienia pamięci. Tutaj możesz dostosować rozmiary stosów, aby podawać żądane ilości. Po zakończeniu kliknij Zastosuj. Jednocześnie, jeśli zmieniłeś rozmiar sterty dla IDE, powinieneś ponownie uruchomić Android Studio przed zapisaniem nowych ustawień pamięci. Pamiętaj też, że przydzielenie zbyt dużej ilości pamięci może najprawdopodobniej obniżyć ogólną wydajność. Jeśli chodzi o ustawienia IDE, możesz wyeksportować plik Settings.jar, który zawiera wszystkie lub segment preferowanych ustawień IDE dla projektu. Następnie możesz zaimportować plik Java Archive (JAR) do innych projektów i udostępnić plik JAR współpracownikom w celu włączenia go do ich projektów.

**Dostosowywanie właściwości IDE**

Aby dostosować właściwości IDE, zastosuj plik idea.properties, który pozwoli Ci dostosować właściwości IDE dla Android Studio, takie jak ścieżka do wtyczek zainstalowanych przez użytkownika i maksymalny rozmiar pliku obsługiwany przez IDE. Plik idea.properties jest następnie konsolidowany z domyślnymi właściwościami środowiska IDE, dzięki czemu można wybrać tylko właściwości zastępowania. Aby utworzyć nowy plik idea.properties lub uzyskać dostęp do istniejącego pliku, wykonaj następujące czynności: Kliknij Pomoc > Edytuj właściwości niestandardowe. Jeśli nie edytowałeś wcześniej właściwości IDE, Android Studio poprosi o utworzenie nowego pliku idea.properties. Po prostu kliknij Tak, aby utworzyć plik. Plik idea.properties stanie się wtedy dostępny w oknie edytora Android Studio. Edytuj plik, aby dodać własne dostosowane właściwości IDE.

**Konfigurowanie IDE dla maszyn o małej ilości pamięci**

Jeśli używasz Android Studio na komputerze z mniej niż zalecaną specyfikacją pamięci, nadal możesz dostosować IDE, aby usprawnić procesy na swoim komputerze, wykonując następujące czynności:

• Aktualizowanie Gradle i wtyczki Android dla Gradle: zaktualizuj najnowsze wersje Gradle i wtyczki Android dla Gradle, aby mieć pewność, że stosujesz najnowsze ulepszenia wydajności.

• Włączenie trybu oszczędzania energii: Włączenie trybu oszczędzania energii wyłącza wiele operacji w tle wymagających dużej ilości pamięci i baterii, w tym podświetlanie błędów i inspekcje w locie, automatyczne uzupełnianie kodu wyskakującego i automatyczną przyrostową kompilację w tle. Aby włączyć tryb oszczędzania energii, kliknij Plik > Tryb oszczędzania energii.

• Wyłączanie zbędnych kontroli kłaczków: Aby zmienić, które kontrole lint aktywuje Android Studio w kodzie, kliknij Plik > Ustawienia (w MacOS, Android Studio > Preferencje), aby otworzyć okno dialogowe Ustawienia. W lewym okienku rozwiń sekcję Edytor i kliknij Kontrole. Kliknij pola wyboru, aby zaznaczyć lub odznaczyć kontrole kłaczków zgodnie z Twoim projektem. Kliknij Zastosuj lub OK, aby zapisać zmiany.

• Debugowanie na urządzeniu fizycznym: Debugowanie na emulatorze zajmuje więcej pamięci niż debugowanie na urządzeniu fizycznym, dzięki czemu można zwiększyć ogólną wydajność Android Studio poprzez debugowanie na urządzeniu fizycznym.

• Uwzględnianie tylko niezbędnych usług Google Play jako zależności: Uwzględnienie w projekcie Usług Google Play jako zależności zwiększa ilość potrzebnej pamięci. Dlatego upewnij się, że uwzględniasz tylko niezbędne zależności, aby poprawić użycie pamięci i wydajność.

• Nie włączaj kompilacji równoległej: Android Studio zazwyczaj kompiluje niezależne moduły równolegle, ale jeśli masz system o małej ilości pamięci, nie powinieneś włączać tej funkcji. Aby zmienić to ustawienie, kliknij Plik > Ustawienia (w systemie MacOS, Android Studio > Preferencje), aby otworzyć okno dialogowe Ustawienia. W lewym okienku rozwiń Build, Execution, Deployment, a następnie kliknij Compiler. Tutaj upewnij się, że opcja Kompiluj niezależne moduły w opcji równoległej nie jest zaznaczona. Po dokonaniu modyfikacji kliknij Zastosuj lub OK, aby zmiana zaczęła obowiązywać.

**Ustawianie wersji JDK**

Kopia najnowszego OpenJDK jest zawarta w Android Studio 2.2 i nowszych. Jest to wersja JDK, którą zalecamy używać w projektach Androida. Aby aktywować i zastosować dołączony pakiet JDK, wykonaj następujące czynności:

Otwórz projekt w Android Studio i przejdź do Plik > Ustawienia > Kompilacja, wykonywanie, wdrażanie > Narzędzia do budowania > Gradle (Android Studio > Preferencje > Kompilacja, wykonywanie, wdrażanie > Narzędzia do budowania > Gradle na Macu). W sekcji Gradle JDK wybierz opcję Embedded JDK i kliknij przycisk OK.

Domyślnie wersja językowa Java używana do skryptowania projektu jest oparta na compileSdkVersion, ponieważ różne wersje Androida obsługują różne wersje Java. Dlatego w razie potrzeby możesz zmienić tę domyślną wersję Java, dodając następujący blok CompileOptions {} do pliku build.Gradle:

```
android {

compileOptions {

sourceCompatibility JavaVersion.VERSION\_1\_6

targetCompatibility JavaVersion.VERSION\_1\_6

}

}
```

**Ustawienia proxy**

Serwery proxy można zdefiniować jako pośredniczące punkty łącza między klientami HTTP a serwerami sieciowymi, które są odpowiedzialne za zapewnienie bezpieczeństwa i prywatności połączeń internetowych. W celu obsługi Android Studio za firewallem, konieczne jest ustawienie proxy odpowiedniego dla Android Studio IDE. Możesz użyć strony ustawień Android Studio IDE HTTP Proxy, aby skonfigurować ustawienia HTTP proxy dla Android Studio. Z paska menu przejdź do Plik > Ustawienia (w systemie MacOS kliknij Android Studio > Preferencje). W lewym okienku należy kliknąć Wygląd i zachowanie > Ustawienia systemu > Serwer proxy HTTP. Gdy pojawi się strona HTTP Proxy, wybierz opcję Automatyczne wykrywanie ustawień proxy, jeśli chcesz użyć adresu URL automatycznej

konfiguracji proxy dla ustawień proxy, lub kliknij Ręczna konfiguracja proxy, aby móc samodzielnie wprowadzić każde z ustawień. Kliknij Zastosuj lub OK, aby zapisać zmiany.

**Wtyczka Androida do ustawień serwera proxy Gradle http**

Podczas uruchamiania wtyczki Androida z wiersza poleceń lub na komputerach, na których nie zainstalowano jeszcze Android Studio, uzyskaj dostęp do wtyczki Androida dla ustawień proxy Gradle za pomocą pliku kompilacji Gradle. Jeśli chcesz uzyskać dostęp do ustawień serwera proxy HTTP specyficznych dla aplikacji, ustaw ustawienia proxy w pliku build.Gradle zgodnie z wymaganiami dla każdego modułu aplikacji. Jeśli chodzi o ustawienia proxy HTTP dla całego projektu, ustaw ustawienia proxy w pliku Gradle/Gradle.properties. Trzeba zrozumieć, że na wydajność Android Studio w systemie Windows mogą mieć wpływ różne okoliczności lub wkłady. Od teraz zobaczymy, jak można zoptymalizować ustawienia Android Studio, aby uzyskać jak najlepsze przewodnictwo w systemie Windows. Najłatwiejszą rzeczą, od której możesz zacząć, jest zminimalizowanie wpływu oprogramowania antywirusowego na szybkość kompilacji. Ogólnie rzecz biorąc, niektóre programy antywirusowe mają tendencję do zakłócania procesu kompilacji Android Studio, co powoduje, że kompilacje działają znacznie wolniej. Po uruchomieniu kompilacji w Android Studio Gradle kompiluje zasoby i kod źródłowy aplikacji, a następnie przechowuje skompilowane zasoby w pakiecie APK. Na tym etapie na komputerze tworzonych jest wiele plików. A jeśli oprogramowanie antywirusowe ma włączoną funkcję skanowania w czasie rzeczywistym, program antywirusowy może spowolnić proces kompilacji za każdym razem, gdy tworzony jest plik, podczas gdy program antywirusowy śledzi i skanuje ten plik. Aby temu zapobiec, należy wyłączyć skanowanie w czasie rzeczywistym niektórych katalogów w oprogramowaniu antywirusowym. Aby jednak zapewnić ochronę komputera przed złośliwymi atakami, nie wolno całkowicie wyłączać skanowania w czasie rzeczywistym ani oprogramowania antywirusowego. Poniżej znajduje się lista domyślnych lokalizacji każdego katalogu Android Studio, który należy wykluczyć ze skanowania w czasie rzeczywistym:

• Pamięć podręczna Gradle: %USERPROFILE%\.gradle

• Projekty Android Studio: %USERPROFILE%\AndroidStudioProjects

• Android SDK: %USERPROFILE%\AppData\Local\Android\SDK Pliki systemowe Android Studio:

• Składnia: %LOCALAPPDATA%\Google\<produkt>

<wersja>

• Przykład: C:\Użytkownicy\NazwaUżytkownika\AppData\Local\Google\AndroidStudio4.1

Jeśli napotkasz zasady grupy ograniczające liczbę katalogów, które możesz wykluczyć ze skanowania w czasie rzeczywistym na swoim komputerze, zawsze możesz przenieść swoje katalogi Android Studio do jednej z lokalizacji, które scentralizowane zasady grupy już wycinają. Tutaj zilustrowaliśmy, w jaki sposób można dostosować lokalizację każdego katalogu Android Studio, gdzie C:\WorkFolder to katalog, który jest już odrzucany przez zasady grupy:

• W przypadku pamięci podręcznej Gradle: Zdefiniuj zmienną środowiskową GRADLE_USER_HOME, aby odwołać się do C:\WorkFolder\.gradle.

• W przypadku projektów Android Studio: Przenieś lub wstaw nowe katalogi projektów do odpowiedniego podkatalogu C:\WorkFolder. Na przykład C:\WorkFolder\AndroidStudioProjects.

• Postępuj zgodnie z tymi instrukcjami dotyczącymi Android SDK: W Android Studio otwórz okno dialogowe Ustawienia (Preferencje w systemie MacOS), a następnie przejdź do opcji Wygląd i

zachowanie > Ustawienia systemu > Android SDK. Zmień wartość lokalizacji Android SDK na C:\WorkFolder\AndroidSDK. Aby powstrzymać się od ponownego pobierania SDK, należy po prostu skopiować istniejący katalog SDK, domyślnie zlokalizowany w

%USERPROFILE%\AppData\Local\Android\SDK, do nowej lokalizacji.

• W przypadku plików systemowych Android Studio: Otwórz Android Studio, a następnie przejdź do Pomoc > Edytuj właściwości niestandardowe. Android Studio poprosi Cię o utworzenie pliku idea.properties, jeśli jeszcze go nie masz. Tutaj upewnij się, że wstawiasz następujący wiersz do pliku idea.properties:

• idea.system.path=c:/workfolder/studio/cache/trunk-system

**KORZYSTANIE ZE SKRYPTÓW BUDOWANIA GRADLE**

Gradle to nazwa zautomatyzowanego zestawu narzędzi opartego na systemie Android, który umożliwia modyfikowanie i administrowanie sposobem tworzenia projektów za pomocą kombinacji różnych plików konfiguracyjnych. Obejmuje to określenie, w jaki sposób projekt ma być oskryptowany, jakie zależności muszą zostać wprowadzone, aby projekt został pomyślnie skompilowany i jak powinien wyglądać końcowy wynik procedury kompilacji. Główną zaletą Gradle jest możliwość dostosowania, którą oferuje do użytkownika. System Gradle pozycjonuje się jako samowystarczalne, oparte na wierszu poleceń ustawienie, które można włączyć do innych ustawień projektu za pomocą wtyczki. W przypadku Android Studio integracja z Gradle odbywa się za pośrednictwem aplikacji o podobnej nazwie Android Studio Plug-in. Chociaż wtyczka Android Studio zapewnia, że zadania Gradle są inicjowane i zarządzane z poziomu Android Studio, opakowanie wiersza poleceń Gradle może być nadal stosowane do uruchamiania projektów opartych na Android Studio na systemach, na których nie zainstalowano Android Studio. Zasady konfiguracji do rozpoczęcia projektu są regulowane przez pliki kompilacji Gradle, a skrypty oparte są na języku programowania Groovy. Gradle wnosi szereg wyjątkowych cech do tworzenia projektów aplikacji na Androida. Niektóre z głównych funkcji omówiono w kolejnych sekcjach.

**Rozsądne wartości domyślne**

Gradle opiera się na koncepcji zwanej konwencją ponad konfiguracją. Mówiąc najprościej, oznacza to, że Gradle ma z góry określoną liczbę rozsądnych domyślnych ustawień konfiguracji, które będą stosowane domyślnie, chyba że zostaną wyłączone przez ustawienia w plikach kompilacji. Oznacza to również, że kompilacje mogą być administrowane z minimalną konfiguracją dostosowaną przez programistę. Zmiany w plikach kompilacji są wymagane tylko wtedy, gdy konfiguracja domyślna nie spełnia wszystkich specyfikacji kompilacji.

**Zależności**

Kolejnym kluczowym elementem zapewniającym funkcjonalność Gradle są zależności. Na przykład moduł w projekcie Android Studio aktywuje ruch w celu załadowania innego modułu w projekcie. Pierwszy moduł jest zależny od drugiego modułu, ponieważ kompilacja aplikacji nie powiedzie się, jeśli drugiego modułu nie można znaleźć i uruchomić w czasie wykonywania. Tę zależność można zaobserwować w pliku kompilacji Gradle dla pierwszego modułu, tak że drugi moduł jest dołączony do kompilacji aplikacji, lub losowy błąd w przypadku, gdy nie można znaleźć lub zbudować drugiego modułu. Innymi przykładami zależności mogą być biblioteki i pliki JAR, na których projekt opiera się na kompilacji i uruchomieniu. Zazwyczaj zależności Gradle są klasyfikowane jako lokalne lub zdalne. Zależność lokalna oznacza element znajdujący się w lokalnym systemie plików komputera, na którym działa kompilacja. Z drugiej strony zależność zdalna odnosi się do elementu, który znajduje się na

serwerze zdalnym (nazywanym również repozytorium). W większości przypadków zdalne zależności są obsługiwane w projektach Android Studio za pośrednictwem innego produktu do zarządzania projektami o nazwie Maven. W przypadku ustanowienia zdalnej zależności w pliku kompilacji Gradle przy użyciu składni Maven, zależność ta zostanie automatycznie pobrana ze zidentyfikowanego repozytorium i dodana do procesu kompilacji. Poniższa deklaracja zależności, na przykład, powoduje włączenie biblioteki AppCompat do projektu z repozytorium Google: Implementation „com.android.support:appcompat-v7:26.0.2”.

**Warianty kompilacji**

Oprócz wspomnianych wcześniej narzędzi, Gradle posiada również wsparcie dla wariantów kompilacji dla projektów Android Studio. Pozwala to na tworzenie wielu odmian aplikacji z jednego projektu. A ponieważ Android działa na wielu różnych urządzeniach obejmujących różne typy procesorów i rozmiary ekranu, ważne jest aby wybrać jak najwięcej typów i rozmiarów urządzeń, aby móc zbudowaćwiele różnych wariantów aplikacji (z interfejsem użytkownika dla telefonów jak również dla ekranów wielkości tabletów). Teraz, dzięki szerokiemu wykorzystaniu Gradle, jest to możliwe w Android Studio.

**Wpisy jawne**

Uznaje się za dopuszczalne, aby do każdego projektu Android Studio dołączony był plik AndroidManifest.xml zawierający dane konfiguracyjne dotyczące aplikacji. W plikach kompilacji Gradle można zdefiniować wiele wpisów manifestu, które są następnie automatycznie generowane w pliku manifestu podczas wykonywania projektu. Ta pojemność jest ściśle dodatkowa do funkcji wariantów kompilacji, umożliwiając różne modyfikowanie funkcji, takich jak numer wersji aplikacji, identyfikator aplikacji i informacje o wersji zestawu SDK dla każdej wersji kompilacji.

**Wsparcie ProGuard**

ProGuard to narzędzie dostarczane z Android Studio, które optymalizuje, łączy i kompiluje kod bajtowy Java, aby uczynić go bardziej produktywnym, ale jednocześnie trudniejszym do inżynierii wstecznej (metoda, dzięki której inni mogą zrozumieć sens aplikacji poprzez analiza skompilowanego kodu bajtowego Java). Pliki kompilacji Gradle umożliwiają określenie, czy funkcja ProGuard jest aktywowana w aplikacji podczas jej tworzenia. Gradle to system typu open source, który służy również do automatyzacji tworzenia, testowania i wdrażania projektów. Build.gradle to skrypty, w których możesz zoptymalizować zadania. Każdy projekt Androida potrzebuje gradle do generowania apk z plików .java i .xml w projekcie. Mówiąc najprościej, gradle pobiera wszystkie pliki źródłowe (java i XML), konwertuje je na pliki dex i kompresuje je wszystkie w jeden plik znany jako apk. Istnieją dwa typy skryptów build.gradle: build.gradle najwyższego poziomu i build.gradle na poziomie modułu. Najwyższego poziomu build.gradle znajduje się w głównym katalogu projektu, a jego głównym celem jest określenie, które konfiguracje kompilacji mają być zastosowane do wszystkich modułów w projekcie. Najwyższego poziomu build.gradle obsługuje następujące konfiguracje kompilacji:

• Buildscript: używany do konfigurowania repozytoriów i zależności dla Gradle.

• Zależności: Ten blok w buildscript jest używany do konfigurowania zależności, które Gradle musi zbudować podczas projektu.

• Wszystkie projekty: jest to blok, w którym można modyfikować wtyczki i biblioteki innych firm. W przypadku świeżo tworzonych projektów Android Studio zawiera domyślnie JCenter i Google maven repozytorium.

• Czyszczenie zadania(type:Delete): Służy do usuwania katalogu za każdym razem, gdy projekt jest aktywowany. W ten sposób projekt pozostaje czysty, gdy ktoś edytuje niektóre pliki konfiguracyjne, takie jak settings.gradle, które wymagają całkowicie czystego środowiska.

build.gradle na poziomie modułu znajduje się w katalogu project/module projektu. W tym konkretnym skrypcie oceny definiuje się wszystkie zależności i weryfikowane są wersje SDK. Skrypt ma wiele funkcji w projekcie, w tym dodatkowe typy kompilacji i modyfikacje ustawień w głównym manifeście aplikacji lub pliku build.gradle najwyższego poziomu. build.gradle na poziomie modułu obsługuje następujące konfiguracje kompilacji

• Android: Ten blok służy do konfigurowania konkretnych opcji kompilacji systemu Android.compileSdkVersion - służy do ustawiania poziomu interfejsu API aplikacji, aby aplikacja mogła korzystać z funkcji tego i niższego poziomu.

• defaultConfig: applicationId - jest używany do identyfikacji unikalnego identyfikatora do publikowania app.

minSdkVersion - definiuje minimalny poziom interfejsu API wymagany do uruchomienia application.

targetSdkVersion - definiuje poziom interfejsu API używany do testowania app.

versionCode - definiuje kod wersji aplikacji. Za każdym razem, gdy aplikacja będzie wymagać aktualizacji, kod wersji musi zostać zwiększony o 1 lub więcej.

versionName - definiuje nazwę wersji aplikacji.

• buildTypes(release): minifyEnabled - służy do włączania zmniejszania kodu dla wydania build.

proguardFiles - służy do określania pliku ustawień projektu.

Zależności - służy do określania zależności wymaganych do skompilowania projektu.

Zarówno pliki najwyższego poziomu, jak i build.gradle na poziomie modułu powinny być postrzegane jako główne pliki skryptowe do automatyzacji zadań w projekcie android i wykorzystywane przez Gradle do generowania APK z plików źródłowych. A jeśli jesteś nowy w Gradle, oto kilka konfiguracji do zarządzania modułami projektu i ich źródłami

**Zmiana domyślnych konfiguracji zestawu źródeł**

Możliwe jest użycie bloku sourceSets w pliku build.gradle na poziomie modułu, aby zmienić miejsce, w którym Gradle odnosi się do zbierania plików dla każdego komponentu zestawu źródłowego. W przypadku projektów, które administrują wieloma modułami, przydatne może być zdefiniowanie właściwości na poziomie projektu i przydzielenie ich do wszystkich modułów. Możesz to również zrobić, dodając dodatkowe właściwości do bloku ext w pliku build.gradle najwyższego poziomu.

**Zarządzanie bibliotekami i zależnościami**

Ponadto Gradle ma również solidny mechanizm zarządzania zależnościami, niezależnie od tego, czy są to zdalne biblioteki, czy lokalne moduły biblioteczne. Jeśli chcesz ustalić zależność tylko dla określonego zestawu źródłowego wariantów kompilacji lub testowego zestawu źródłowego, po prostu upewnij się, że nazwa konfiguracji zależności jest pisana wielkimi literami i poprzedza ją nazwą wariantu kompilacji lub testowego zestawu źródłowego.

**Tworzenie różnych wersji Twojej aplikacji**

Gradle i wtyczka Androida pozwalają tworzyć różne wersje aplikacji z jednego modułu, konfigurując warianty kompilacji. Dzięki wtyczce do systemu Android możesz uruchomić wiele pakietów APK, które są przeznaczone dla każdego interfejsu binarnego aplikacji (ABI) lub gęstości ekranu, i skorzystać z obsługi wielu pakietów APK w Google Play. Jeśli jednak chcesz utworzyć oddzielne pakiety APK dla różnych gęstości ekranu, dodaj blok android.splits.density do pliku build.Gradle modułu. Aby utworzyć oddzielne pakiety APK dla każdego ABI, musisz dodać blok android.splits.abi do pliku build.Gradle modułu. Domyślnie, gdy Gradle uruchamia pakiety APK dla Twojego projektu, każdy plik APK ma te same informacje o wersji, co w pliku build.gradle na poziomie modułu. Ponieważ Sklep Google Play nie zezwala na wiele pakietów APK dla tej samej aplikacji, które mają te same informacje o wersji, przed przesłaniem go do Sklepu Play musisz upewnić się, że każdy plik APK ma swój unikalny kod wersji. Możesz to zrobić za pomocą niestandardowej logiki kompilacji, która dystrybuuje inny kod wersji do każdego pakietu APK w czasie kompilacji.

**Konfigurowanie ustawień manifestu oprzyrządowania**

Gdy Gradle tworzy testowy pakiet APK, automatycznie konfiguruje plik AndroidManifest.xml i konfiguruje go z węzłem < Instrumentation >. Istnieje możliwość zmiany niektórych ustawień tego węzła przez wygenerowanie innego pliku manifestu w testowym zestawie źródeł lub skonfigurowanie pliku build.gradle na poziomie modułu. Pierwotnie wszystkie testy były uruchamiane względem typu kompilacji debugowania. Możesz zmienić to na inny typ kompilacji, dodając właściwość testBuildType w pliku build.gradle na poziomie modułu. Aby to zilustrować, jeśli chcesz przeprowadzić testy dla typu kompilacji „produkcyjnej”, zmodyfikuj plik w następujący sposób:

```
android {

…

testBuildType "production"

}
```

Konfigurowanie opcji testu Gradle

Jeśli chcesz określić, które konkretnie opcje zmieniają sposób uruchamiania przez Gradle wszystkich twoich testów, skonfiguruj blok testOptions w module build.gradle. Ponadto możesz również skorzystać z poniższych konfiguracji, aby przyspieszyć pełne i przyrostowe kompilacje.

Shrink Your Code: Android Studio używa narzędzia R8, które konwertuje nasz kod bajtowy Java na zoptymalizowany kod dex. Przechodzi przez całą aplikację i optymalizuje nieużywane klasy i metody. Działa w czasie kompilacji i zmniejsza rozmiar kompilacji, aby była bardziej solidna. W przypadku nowych projektów Android Studio używa domyślnego pliku ustawień (proguard-android.txt) z folderu narzędzi/proguard/folderu Android SDK. Tak więc, jeśli potrzebujesz większego zmniejszenia kodu niż zapewnia R8, spróbuj zmodyfikować plik proguard-android-optimize.txt:

```
android {

buildTypes {

release {

minifyEnabled true

proguardFiles getDefaultProguardFile

('proguard-android-optimize.txt'),
```

'proguard-rules.pro'

}

}

…

Podpisz swoją aplikację : mimo że Android Studio zapewnia prostą ścieżkę do konfiguracji podpisywania dla kompilacji wydania z poziomu interfejsu użytkownika, możesz również ręcznie edytować blok signingConfigs w pliku build.gradle modułu. Wszystkie konfiguracje podpisywania są zwykle przechowywane jako zwykły tekst w pliku build.gradle modułu. Jeśli więc pracujesz z zespołem lub projektem open source, możesz przenieść informacje, które mogą być poufne z plików kompilacji, postępując w następujący sposób: Najpierw utwórz plik o nazwie keystore.properties w katalogu głównym swojego projekt i zawierać następujące informacje:

storePassword=myStorePassword

keyPassword=myKeyPassword4

keyAlias=myKeyAlias

storeFile=myStoreFileLocation

Second, in your build.gradle file, you need to load the keystore.properties file in the following manner (this has to be before the android block):

//Creates a variable called keystorePropertiesFile, and initializes it to the

//keystore.properties file.

def keystorePropertiesFile = rootProject

.file("keystore.properties")

//Initializes a new Properties() object called keystoreProperties.

def keystoreProperties = new Properties()

//Loads the keystore.properties file into the keystoreProperties object.

keystoreProperties.load(new FileInputStream

(keystorePropertiesFile))

android {

…

}

Gdy skończysz, po prostu wprowadź informacje o podpisie przechowywane w obiekcie keystoreProperties:

android {

signingConfigs {

config {

keystoreProperties ['keyAlias'] keyPassword keystoreProperties ['keyPassword'] storeFile file(keystoreProperties ['storeFile']) storePassword keystoreProperties ['storePassword']

}

}

...

Na koniec kliknij Synchronizuj teraz na pasku powiadomień.

Uprość tworzenie aplikacji Aby uprościć tworzenie aplikacji, zaleca się współdzielenie niestandardowych pól i wartości zasobów z kodem aplikacji. Dlatego w czasie kompilacji Gradle generuje klasę BuildConfig, dzięki czemu kod aplikacji może potencjalnie sprawdzić wszystkie dane dotyczące bieżącej kompilacji. W tym miejscu możesz również wstawić niestandardowe pola do klasy BuildConfig z pliku konfiguracyjnego kompilacji Gradle za pomocą metody buildConfigField() i uzyskać dostęp do tych wartości w kodzie wykonawczym aplikacji. Podobnie, możesz również uwzględnić wartości zasobów aplikacji za pomocą resValue().Share Właściwości z Manifestem W kilku przypadkach może być konieczne zadeklarowanie tej samej właściwości zarówno w manifeście, jak i w kodzie (na przykład podczas deklarowania uprawnień dla FileProvider ). Zamiast aktualizować tę samą właściwość w wielu lokalizacjach w celu odzwierciedlenia zmiany, możesz zidentyfikować pojedynczą właściwość w pliku build.gradle modułu, aby udostępnić ją zarówno w manifeście, jak i w kodzie. Domyślnie każdy projekt Android Studio zawiera narzędzie Gradle wrapper w celu autoryzacji zadań Gradle, które mają być żądane z wiersza poleceń. To narzędzie znajduje się w katalogu głównym każdego folderu projektu. Mimo że to opakowanie jest wykonywalne w systemach Windows, nadal musi mieć włączone uprawnienia w systemach Linux i MacOS, zanim będzie można z niego korzystać. Aby włączyć takie uprawnienie, musisz uzyskać dostęp do okna terminala i zmienić katalog na folder projektu, dla którego potrzebne jest opakowanie, i wykonać następujące polecenie:

chmod +x gradlew

Gdy plik ma uprawnienia do wykonywania, lokalizacja pliku będzie musiała być uwzględniona w zmiennej środowiskowej $PATH lub nazwa poprzedzona ./ w celu uruchomienia. Na przykład: zadania ./gradlew. Ogólnie rzecz biorąc, Gradle traktuje tworzenie projektów w kategoriach wielu różnych zadań. Pełną listę zadań dostępnych dla bieżącego projektu można pobrać, aktywując następujące polecenie z katalogu projektu (upewnij się, że poprzedziłeś polecenie przedrostkiem ./, jeśli działa w systemie MacOS lub Linux): zadania gradlew. Aby zbudować wersję debugową projektu odpowiednią do testowania urządzeń lub emulatorów, należy zastosować opcję assembleDebug: gradlew assembleDebug. Alternatywnie, aby zbudować wersję wydania aplikacji, możesz wstawić: gradlew assembleRelease. W większości Android Studio jest w stanie w pełni zarządzać kompilacjami aplikacji w tle bez jakiejkolwiek interwencji ze strony programisty. Ten proces kompilacji działa przy użyciu systemu Gradle, który jest przeznaczony do przeglądania sposobów kompilowania projektów w celu skonfigurowania ich za pomocą zestawu plików konfiguracji kompilacji. I chociaż standardowe zachowanie Gradle jest wystarczające dla wielu podstawowych wymagań dotyczących budowania projektu, to w przypadku bardziej skomplikowanych projektów potrzeba modyfikacji procesu budowania jest nieunikniona.

**PRACA Z PROJEKTAMI**

Projekt w Android Studio zawiera wszystko, co definiuje Twój obszar roboczy dla aplikacji; obejmuje to kod źródłowy, zasoby, kod testowy i konfiguracje kompilacji. Kiedy zaczynasz nowy projekt, Android Studio tworzy niezbędną kompozycję dla wszystkich twoich plików i czyni je widocznymi w oknie

projektu (poprzez View > Tool Windows > Project). Ta strona zwykle zawiera cały przegląd kluczowych komponentów w twoim projekcie.

**Moduły**

Jednym z kluczowych elementów projektu jest moduł. Oznacza zbiór plików źródłowych i ustawień kompilacji, które pozwalają podzielić projekt na oddzielne jednostki wykonania. Twój projekt może mieć jeden lub wiele modułów, a jeden moduł może używać innego modułu jako zależności. Niemniej jednak możesz mieć pewność, że każdy moduł może być niezależnie budowany, testowany i debugowany. Dodatkowe moduły są często używane do tworzenia różnych bibliotek kodu w ramach projektu lub jeśli potrzebujesz wielu zestawów kodu i zasobów dla różnych typów urządzeń, takich jak telefony i urządzenia do noszenia, ale jednocześnie upewnij się, że wszystkie pliki są objęte zakresem ten sam projekt i udostępnij jakiś kod. Możesz dodać nowy moduł do swojego projektu, klikając Plik > Nowy > Nowy moduł. Android Studio ma kilka innych odrębnych typów modułów:

**Moduł aplikacji na Androida**

Działa jako kontener dla kodu źródłowego aplikacji, plików zasobów i ustawień na poziomie aplikacji, takich jak plik kompilacji na poziomie modułu i plik manifestu Androida. Podczas tworzenia nowego projektu domyślna nazwa modułu to „app”. W oknie Utwórz nowy moduł Android Studio ma do wyboru następujące typy modułów aplikacji:

• Moduł telefonu i tabletu

• Wear OS Moduł

• Moduł Android TV

• Moduł Glass

Każdy z nich przechowuje pliki kluczy i szablony kodów, które są odpowiednie dla odpowiedniej aplikacji lub typu urządzenia.

**Moduł funkcji**

Ten typ modułu reprezentuje zmodularyzowaną funkcję Twojej aplikacji, która jest ściśle powiązana z dostarczaniem funkcji Play. Na przykład, dzięki modułom funkcji, możesz zapewnić użytkownikom określone funkcje na żądanie lub natychmiastowe doświadczenia w Google Play Instant.

**Moduł biblioteczny**

Zapewnia lokalizację kodu wielokrotnego użytku, który można zastosować jako zależność w innych modułach aplikacji lub użyć z innymi projektami. Sądząc ściśle ze struktury, moduł biblioteki bardzo przypomina moduł aplikacji, jednak po zbudowaniu daje plik archiwum kodu zamiast APK, więc nie można go zainstalować na urządzeniu. W oknie Create New Module Android Studio prezentuje następujące moduły biblioteczne:

• Biblioteka systemu Android: ten typ biblioteki może zawierać wszystkie typy plików obsługiwane w projekcie systemu Android, w tym kod źródłowy, zasoby i pliki manifestu. Wynikiem końcowym jest plik Android Archive (AAR), który można dołączyć jako zależność dla modułów aplikacji systemu Android.

• Biblioteka Java: Ten typ biblioteki może przechowywać tylko pliki źródłowe Java. Wynikiem kompilacji jest plik JAR, który można dołączyć jako zależność dla modułów aplikacji systemu Android lub innych projektów Java.

**Moduł Google Cloud**

Moduł Google Cloud to idealny kontener na kod backendu Google Cloud. Ten moduł zawiera wymagany kod i zależności dla backendu Java App Engine, który używa prostego protokołu HTTP, Cloud Endpoints i Cloud Messaging do łączenia się z Twoją aplikacją. Dzięki temu możesz rozwijać swój backend, aby świadczyć usługi w chmurze dla swojej aplikacji. Zastosowanie Android Studio do tworzenia i zarządzania modułem Google Cloud pozwala regulować kod aplikacji i kod zaplecza w tym samym projekcie. Możesz także uruchamiać i testować swój kod zaplecza lokalnie, a także wdrażać moduł Google Cloud za pośrednictwem Android Studio. Uważa się za normę postrzeganie modułów jako podprojektów, ponieważ Gradle odnosi się do modułów jako projektów. Na przykład, gdy tworzysz moduł biblioteki i chcesz go dodać jako zależność do modułu aplikacji Android, powinieneś zadeklarować go jako projekt implementacji w następujący sposób:

dependencies {

implementation project

(':my-library-module')

}

**Pliki projektu**

Domyślnie Android Studio przechowuje pliki projektu w widoku Androida. Jednak ten widok nie odzwierciedla rzeczywistej struktury plików na dysku, ale jest po prostu zorganizowany według modułów i typów plików, aby umożliwić nawigację między kluczowymi plikami źródłowymi projektu lub usunąć niektóre pliki lub katalogi, które nie są często używane. Niektóre zmiany strukturalne w porównaniu do struktury na dysku obejmują:

• Możesz przeglądać wszystkie pliki konfiguracyjne związane z kompilacją projektu w grupie najwyższego poziomu Gradle Script.

• Możesz zobaczyć wszystkie pliki manifestu dla każdego modułu w grupie na poziomie modułu (jeśli masz różne pliki manifestu dla różnych produktów i typów kompilacji).

• Możesz uzyskać dostęp do wszystkich alternatywnych plików zasobów w jednej grupie, zamiast w osobnych folderach na kwalifikator zasobów. Na przykład wszystkie wersje gęstości ikony programu uruchamiającego były widoczne obok siebie.

Aby przejrzeć rzeczywistą strukturę plików projektu, w tym wszystkie pliki ukryte w widoku Androida, wybierz Projekt z listy rozwijanej u góry okna Projekt. Po kliknięciu Projekt będziesz mógł uzyskać dostęp do najważniejszych plików i katalogów w następujący sposób:

• build/: Zawiera dane wyjściowe kompilacji.

• libs/: Zawiera prywatne biblioteki.

• src/: Zawiera cały kod i pliki zasobów modułu.

• androidTest/: zawiera kod do testów oprzyrządowania uruchamianych na urządzeniu z systemem Android.

• main/: Zawiera „główne” pliki zestawu źródłowego: Kod i zasoby Androida współdzielone przez wszystkie warianty kompilacji (pliki innych wariantów kompilacji znajdują się w katalogach równorzędnych, takich jak src/debug/ dla typu kompilacji debug).

• AndroidManifest.xml: opisuje charakter aplikacji i każdy z jej składników.

• java/: Zawiera źródła kodu Java.

• jni/: Zawiera kod natywny korzystający z interfejsu Java Native Interface (JNI).

• gen/: Zawiera pliki Java wygenerowane przez Android Studio, takie jak plik R.java i interfejsy utworzone z plików języka definicji interfejsu Androida (AIDL).

• res/: Zawiera zasoby aplikacji, takie jak pliki do rysowania, pliki układu i ciągi interfejsu użytkownika.

• asset/: Zawiera plik, który powinien zostać skompilowany do pliku .apk bez zmian. Możesz poruszać się po tym katalogu w taki sam sposób, jak w typowym systemie plików, używając jednolitych identyfikatorów zasobów (URI) i czytać pliki jako strumień bajtów za pomocą AssetManager. Na przykład jest to dobra lokalizacja dla tekstur i danych gry.

• test/: zawiera kod do testów lokalnych uruchamianych na maszynie JVM hosta.

• build.gradle (moduł): Definiuje konfiguracje kompilacji specyficzne dla modułu.

• build.gradle (projekt): Definiuje konfigurację kompilacji, która ma zastosowanie do wszystkich modułów. Ten plik jest integralną częścią projektu, więc powinieneś mieć nad nim kontrolę nad wszystkimi pozostałymi kodami źródłowymi.

**Ustawienia struktury projektu**

Aby zmienić różne ustawienia projektu Android Studio, otwórz okno dialogowe Struktura projektu, klikając Plik > Struktura projektu. Posiada następujące główne sekcje:

• Lokalizacja SDK: Ustawia lokalizację JDK, Android SDK i Android NDK, których używa Twój projekt.

• Projekt: koordynuje wersję dla Gradle i wtyczkę Android dla Gradle oraz nazwę lokalizacji repozytorium.

• Moduły: pozwalają edytować konfiguracje kompilacji specyficzne dla modułu, w tym docelowy i minimalny zestaw SDK, sygnaturę aplikacji i zależności bibliotek. Sekcja Ustawienia modułów umożliwia zmianę opcji konfiguracyjnych dla każdego modułu projektu. Strona ustawień każdego modułu składa się z następujących zakładek:

- Właściwości: Określa wersje zestawu SDK i narzędzi do kompilacji, które mają być używane podczas kompilowania modułu.

- Podpisywanie: identyfikuje certyfikat, którego należy użyć do podpisania pakietu APK.

- Charakterystyczne cechy: umożliwia tworzenie wielu wariantów kompilacji, w których każda cecha określa zestaw ustawień konfiguracyjnych, takich jak minimalna i docelowa wersja SDK modułu oraz kod i nazwa wersji. Na przykład możesz zdefiniować jedną cechę, który ma minimum SDK 15 i docelowy SDK 21, a inna cecha ma minimum 19 i docelowy SDK 23.

- Typy kompilacji: Pozwala tworzyć i modyfikować konfiguracje kompilacji. Domyślnie każdy moduł ma typy kompilacji debugowania i wydania, ale w razie potrzeby można zdefiniować więcej.

- Zależności: Wyświetla listę zależności biblioteki, pliku i modułu dla tego modułu. W tej sekcji możesz dodawać, modyfikować i usuwać zależności.

Teraz spróbujmy przejść przez tworzenie nowego projektu Androida za pomocą Android Studio. Procedura jest dość prosta i wymaga wykonania następujących prostych kroków. Po zainstalowaniu najnowszej wersji Android Studio w oknie Witamy w Android Studio kliknij Utwórz nowy projekt. Jeśli masz już otwarty projekt, wybierz Nowy projekt. W oknie Wybierz szablon projektu wybierz Puste działanie i kliknij Dalej. Jeśli chcesz umieścić projekt w innym folderze, zmień jego lokalizację Zapisz. Wybierz Java lub Kotlin z menu rozwijanego Język i w polu Minimalny pakiet SDK wybierz najniższą wersję Androida, jaką ma obsługiwać Twoja aplikacja. Będziesz także mieć dostęp do linku Pomóż mi wybrać w oknie dialogowym Dystrybucja platformy Android. To okno dialogowe zawiera informacje o różnych wersjach Androida, które są dystrybuowane na urządzenia. Kluczową kwestią do rozważenia jest odsetek urządzeń z Androidem, które musisz obsługiwać, a także ilość pracy, aby utrzymać aplikację na każdej z różnych wersji, na których działają te urządzenia. Na przykład, jeśli zdecydujesz się na zapewnienie kompatybilności swojej aplikacji z wieloma różnymi wersjami Androida, zwiększysz wysiłek wymagany do zachowania zgodności między najstarszą a najnowszą wersją. Jeśli Twoja aplikacja wymaga standardowej obsługi starszych bibliotek, zaznacz pole wyboru Użyj starszych bibliotek android.support. Pozostałe opcje pozostaw bez zmian i kliknij Zakończ. Po pewnym czasie przetwarzania pojawi się główne okno Android Studio, w którym możesz poświęcić chwilę na przejrzenie najważniejszych plików. Tutaj upewnij się, że okno projektu jest otwarte, a widok Androida jest wybrany z listy rozwijanej u góry tego okna. W tym momencie powinieneś być w stanie zobaczyć następujące pliki:

app > java > com.example.myfirstapp > MainActivity

To jest główny plik aktywności. Zawiera punkt wejścia dla Twojej aplikacji. System automatycznie uruchamia instancję tego działania i ładuje jego układ podczas tworzenia i uruchamiania aplikacji.

app > res > layout > activity_main.xml

Ten plik XML definiuje układ interfejsu użytkownika działania. Zawiera standardowy element TextView z tekstem „Hello, World!”

app > manifests > AndroidManifest.xml

Ten plik manifestu określa podstawowe cechy aplikacji i definiuje każdy z jej elementów.

Gradle Scripts > build.gradle

Istnieją dwa pliki o tej nazwie: jeden dla projektu „Project: My_First_App” i jeden dla modułu aplikacji „Module: My_First_App.app”. Każdy moduł ma swój własny plik build.gradle, ale zasadniczo możesz użyć pliku build.Gradle każdego modułu, aby regulować sposób, w jaki wtyczka Gradle buduje Twoją aplikację.

**Wybór Twojego projektu**

Jeśli nie masz jeszcze otwartego projektu, Android Studio wyświetla ekran powitalny, na którym możesz otworzyć nowy projekt, klikając Uruchom nowy projekt Android Studio. Następnie należy postępować zgodnie z kreatorem Utwórz nowy projekt, który pozwala wybrać typ projektu, który chcesz utworzyć, i wypełnić go kodem oraz zasobami, aby rozpocząć. Na ekranie Wybierz projekt, który pojawia się wraz z kreatorem, możesz wybrać typ projektu, który chcesz utworzyć, z kategorii wymiarów urządzenia, które są prezentowane jako zakładki w górnej części kreatora. Na pierwszym ekranie kreatora wybierz typ projektu, który chcesz utworzyć. Wybierając typ projektu, który chcesz utworzyć, Android Studio może dołączyć przykładowy kod i zasoby, które poprowadzą Cię dalej. Po podjęciu decyzji po prostu kliknij Dalej.

**Konfiguracja Twojego projektu**

Następnym krokiem byłoby skonfigurowanie niektórych ustawień i utworzenie nowego projektu. Konfiguracja nowego projektu wymaga tylko kilku ustawień. Upewnij się, że podałeś nazwę swojego projektu oraz nazwę pakietu. Domyślnie ta nazwa pakietu staje się również identyfikatorem Twojej aplikacji, który zawsze możesz zmienić później. Musisz również określić lokalizację Zapisz, w której musisz lokalnie umieścić swój projekt. Ponadto wybierz język, który chcesz zastosować w Android Studio podczas tworzenia przykładowego kodu dla nowego projektu. Pamiętaj jednak, że nie jesteś ograniczony do używania tylko tego języka tworząc projekt. Następnie wybierz Minimalny poziom interfejsu API, który ma obsługiwać Twoja aplikacja. Po wybraniu niższego poziomu interfejsu API Twoja aplikacja może polegać na mniejszej liczbie nowoczesnych interfejsów API systemu Android. Jednak większy odsetek urządzeń z Androidem może uruchomić Twoją aplikację. Odwrotna sytuacja jest również prawdą przy wyborze wyższego poziomu API. Jeśli chcesz, aby Twój projekt domyślnie korzystał z bibliotek AndroidX, które oznaczają ulepszoną wersję bibliotek Android Support, zaznacz pole obok pozycji Użyj AndroidX. Kiedy będziesz gotowy do stworzenia swojego projektu, po prostu kliknij Zakończ. Android Studio może stworzyć Twój nowy projekt przy użyciu tylko podstawowego kodu i minimalnych zasobów. Jeśli jednak później zdecydujesz się dodać obsługę innego współczynnika kształtu urządzenia, możesz później dołączyć moduł do swojego projektu. A jeśli chcesz udostępniać kod i zasoby między modułami, możesz to zrobić, ustanawiając bibliotekę Androida.

**Importuj istniejący projekt**

Aby zaimportować istniejący, lokalny projekt do Android Studio, wykonaj następujące czynności: Przejdź do Plik i kliknij Importuj projekt. W wyświetlonym oknie przejdź do katalogu głównego projektu, który chcesz zaimportować i kliknij OK. Android Studio następnie uruchamia projekt w nowym oknie IDE i indeksuje jego zawartość. Jeśli importujesz projekt z kontroli wersji, użyj opcji Projekt z menu Kontrola wersji.

**Konfiguracja kompilacji modułu funkcji**

Kiedy tworzysz nowy moduł funkcji za pomocą Android Studio, IDE stosuje następującą wtyczkę Gradle do pliku build.Gradle modułu: Zastosuj wtyczkę: „com.android.dynamic-feature”. Większość właściwości dostępnych dla standardowej wtyczki aplikacji jest również dostępna dla Twojego modułu funkcji. Istnieją jednak pewne właściwości, których nie zaleca się uwzględniać w konfiguracji kompilacji modułu funkcji. Ponieważ każdy moduł funkcji zależy od modułu podstawowego, dziedziczy również pewne konfiguracje. Dlatego powinieneś pominąć następujące elementy w pliku build.gradle modułu funkcji:

• Właściwość minifyEnabled: Możesz aktywować zmniejszanie kodu dla całego projektu aplikacji tylko z konfiguracji kompilacji modułu podstawowego. Dlatego warto pominąć tę właściwość w modułach funkcji. Możesz jednak dodać określone reguły ProGuard dla każdego modułu funkcji.

• Konfiguracje podpisywania: Pakiety aplikacji są podpisywane przy użyciu konfiguracji podpisywania określonych na początku w module podstawowym.

• versionCode i versionName: podczas tworzenia pakietu aplikacji Gradle używa informacji o wersji aplikacji udostępnianych przez moduł podstawowy. Dlatego nie ma potrzeby podawania dodatkowych szczegółów przy użyciu tych właściwości.

**Nawiąż relację z modułem podstawowym**

Gdy Android Studio zarządza modułem funkcji, upewnia się, że jest on widoczny dla modułu podstawowego, dodając właściwość android.dynamicFeatures do pliku build.gradle modułu podstawowego, jak pokazano poniżej:

```
//W pliku build.Gradle modułu podstawowego.

android {

…

//Określa moduły funkcji, które są zależne od

//tego modułu podstawowego.

dynamicFeatures = [":dynamic_feature", ":dynamic_feature2"]

}
```

Ponadto Android Studio zawiera również moduł podstawowy jako zależność modułu funkcji w następujący sposób:

```
//W pliku build.Gradle modułu funkcji:

…

dependencies {

…

//Deklaruje zależność od modułu podstawowego „:app”.

implementation project(':app')

}
```

Określ dodatkowe zasady ProGuard

Chociaż tylko konfiguracja kompilacji modułu podstawowego może autoryzować i regulować zmniejszanie kodu dla projektu aplikacji, możesz ustawić niestandardowe reguły ProGuard dla każdego modułu funkcji przy użyciu właściwości proguardFiles, jak pokazano tutaj:

```
android.buildTypes {

release {

// You must use the following property to specify additional ProGuard

// rules for feature modules.

proguardFiles 'proguard-rules-dynamic-features.pro'

}

}
```

Należy pamiętać, że te reguły ProGuard są łączone z regułami z innych modułów (w tym modułu podstawowego) w czasie kompilacji. Oznacza to, że chociaż każdy moduł funkcji może definiować nowy zestaw reguł, reguły te dotyczą wszystkich modułów w projekcie aplikacji.

**Wdróż swoją aplikację**

Podczas konfigurowania aplikacji z różną obsługą modułów funkcji możliwe jest również wdrożenie jej na podłączonym urządzeniu, tak jak zwykle, klikając funkcję Uruchom na pasku menu. Jednak w przypadku, gdy projekt aplikacji zawiera jeden lub więcej modułów funkcji, możesz wybrać, które funkcje dodać podczas wdrażania aplikacji, modyfikując istniejącą konfigurację uruchamiania/debugowania w następujący sposób: Przejdź do opcji Uruchom > Edytuj konfiguracje z paska menu i przejrzyj opcję Uruchom/ Okno dialogowe konfiguracji debugowania na lewym panelu, aby wybrać żądaną konfigurację aplikacji na Androida. W obszarze Funkcje dynamiczne do wdrożenia na karcie Ogólne przejdź przez pole obok każdego modułu funkcji, który chcesz uwzględnić podczas wdrażania aplikacji. Gdy już się zdecydujesz, po prostu kliknij OK. Jednocześnie Android Studio zazwyczaj nie wdraża Twojej aplikacji przy użyciu pakietów aplikacji do wdrażania aplikacji. Zamiast tego IDE tworzy i instaluje na Twoim urządzeniu pakiety APK zoptymalizowane pod kątem szybkości wdrażania, a nie rozmiaru APK.

**Użyj modułów funkcji do niestandardowej dostawy**

Wielką zaletą modułów funkcji jest możliwość dostosowania sposobu i czasu pobierania różnych funkcji aplikacji na urządzenia z systemem Android 5.0 (poziom API 21) lub nowszym. Na przykład, aby zmniejszyć początkowy rozmiar pobieranej aplikacji, możesz zmodyfikować niektóre funkcje, aby były pobierane w razie potrzeby na żądanie lub tylko przez urządzenia, które obsługują określone funkcje, takie jak możliwość robienia zdjęć lub obsługują cechy rzeczywistości rozszerzonej. Nawet jeśli domyślnie otrzymujesz wysoce zoptymalizowane pliki do pobrania, gdy przesyłasz swoją aplikację jako pakiet aplikacji, bardziej ulepszone i odpowiednie opcje dostarczania funkcji wymagają większej konfiguracji i modularyzacji funkcji aplikacji za pomocą modułów funkcji. Oznacza to, że moduły funkcji stanowią cegiełki do tworzenia funkcji modułowych, które można później skonfigurować, aby można było je pobrać w razie potrzeby. Weźmy aplikację, z której chcesz, aby Twoi użytkownicy kupowali i sprzedawali towary na rynku online. Dzięki modułom funkcji możesz skutecznie zmodularyzować każdą z następujących funkcji aplikacji w osobne moduły funkcji:

•Logowanie i tworzenie konta

•Przeglądanie rynku

•Wystawienie przedmiotu na sprzedaż

•Przetwarzanie płatności

Tworzenie identyfikatora URI dla zasobu

Czasami może być potrzebny dostęp do zasobu przechowywanego w module funkcji. Najłatwiej to zrobić za pomocą identyfikatora URI przy użyciu Uri.Builder():

```
val uri = Uri.Builder()

.scheme(ContentResolver.SCHEME_ANDROID_RESOURCE)

.authority(context.getPackageName()) // Look up the resources in the application with its splits loaded

.appendPath(resources.getResourceTypeName(resId))

.appendPath(String.format("%s:%s",

resources.getResourcePackageName(resId), // Look up the dynamic resource in the split namespace.

resources.getResourceEntryName(resId)
```

))

.build()

Każdy segment tej ścieżki do zasobu jest tworzony w czasie wykonywania, co zapewnia utworzenie właściwej przestrzeni nazw po załadowaniu podzielonych pakietów APK. Jako przykład sposobu generowania identyfikatora URI załóżmy, że masz aplikację i moduły funkcji o następujących nazwach:

•Nazwa pakietu aplikacji: com.example.my_app_package

•Nazwa pakietu zasobów funkcji: com.example.my_app_package.my_dynamic_feature

Jeśli identyfikator resId w kodzie fragment powyżej odnosi się do surowego zasobu pliku o nazwie „moje_wideo" w module funkcji, a powyższy kod Uri.Builder() spowoduje następujący wynik: android.resource://com.example.my_app_package/raw/com.example.my_app_package.my_dynamic_feature:my_videoTen identyfikator URI może być następnie wykorzystany przez aplikację w celu uzyskania dostępu do zasobów modułu funkcji. Aby zweryfikować ścieżki w identyfikatorze URI, możesz użyć narzędzia APK Analyzer do śledzenia pliku APK modułu funkcji i określenia nazwy pakietu lub użyć narzędzia APK Analyzer do sprawdzenia nazwy pakietu w skompilowanym pliku zasobów.

**Uwagi dotyczące modułów funkcji**

Korzystając z modułów funkcji, można znacznie poprawić szybkość kompilacji i prędkość projektowania oraz znacznie dostosować dostarczanie funkcji aplikacji, aby zmniejszyć jej rozmiar. Ale jednocześnie istnieją pewne ograniczenia i ograniczenia, o których należy pamiętać podczas korzystania z modułów funkcji:

• Zainstalowanie 50 lub więcej modułów funkcji na jednym urządzeniu, poprzez dostawę warunkową lub na żądanie, może spowodować poważne awarie operacyjne. Moduły czasu instalacji, które nie są traktowane jako wymienne, są automatycznie dołączane do modułu podstawowego i liczą się tylko jako jeden moduł funkcji na każdym urządzeniu.

• - Upewnij się, że liczba modułów, które modyfikujesz jako wymienne w celu dostarczenia w czasie instalacji, została ograniczona do dziesięciu. W przeciwnym razie czas pobierania i instalacji aplikacji może gwałtownie wzrosnąć.

• Należy pamiętać, że tylko urządzenia z systemem Android 5.0 (poziom API 21) lub nowszym obsługują pobieranie i instalowanie funkcji na żądanie.

• Moduły funkcji nie powinny oznaczać działań w swoim manifeście z ustawieniem android:exported na true. Dzieje się tak głównie dlatego, że nie ma gwarancji, że urządzenie pobrało moduł funkcji, gdy inna aplikacja spróbuje uruchomić aktywność. Oznacza to, że Twoja aplikacja powinna sprawdzić, czy funkcja została pobrana, zanim spróbuje uzyskać dostęp do jej kodu i zasobów.

• Podczas wprowadzania nowego modułu funkcji za pomocą Android Studio, IDE dodaje większość atrybutów manifestu, których moduł wymaga, aby zachowywał się jak moduł funkcji.

Ponadto niektóre atrybuty są uwzględniane przez system kompilacji w czasie kompilacji, więc nie ma potrzeby samodzielnego ich określania lub edytowania. Poniższa lista przedstawia atrybuty manifestu, które są ważne dla modułów funkcji:

1. <manifest…: To jest standardowy blok <manifestu>.

2. xmlns:dist=http://schemas.android.com/apk/distribution: Określa nową przestrzeń nazw dist: XML.

3. split="split_name": Definiuje nazwę modułu, którą aplikacja określa podczas żądania modułu na żądanie za pomocą biblioteki Play Core. Gdy Android Studio tworzy pakiet aplikacji, domyślnie dodaje ten atrybut. Dlatego nie ma potrzeby samodzielnego dołączania lub modyfikowania tego atrybutu. Mówiąc prościej, kiedy tworzysz moduł funkcji za pomocą Android Studio, IDE używa nazwy określonej jako nazwa modułu, aby zidentyfikować moduł jako podprojekt Gradle w pliku ustawień Gradle. A kiedy tworzysz pakiet aplikacji, Gradle stosuje ostatni element ścieżki podprojektu, aby wstawić ten atrybut manifestu w manifeście modułu. Aby to zilustrować, jeśli utworzysz nowy moduł funkcji w MyAppProject/features/katalog i określisz „basic_feature1" jako nazwę modułu, IDE doda „:features:basic_feature1" jako podprojekt w twoim pliku settings.Gradle. Podczas tworzenia pakietu aplikacji Gradle wstawia <manifest split="basic_feature1"> do manifestu modułu.

4. android:isFeatureSplit="true | false">: określa, że ten moduł jest modułem funkcji. Gdy Android Studio tworzy pakiet aplikacji, domyślnie dołącza ten atrybut. Dlatego nie należy ręcznie dodawać ani modyfikować tego atrybutu.

5. <dist:module: Ten nowy element XML definiuje atrybuty, które określają, w jaki sposób moduł powinien być pakowany i dystrybuowany jako pakiety APK.

6. dist:instant="prawda | false": określa, czy moduł powinien być dostępny w wyszukiwarce Google Play jako natychmiastowa. Jeśli Twoja aplikacja zawiera co najmniej jeden moduł funkcji z obsługą natychmiastową, musisz również włączyć moduł podstawowy. Podczas korzystania z Android Studio 3.5 lub nowszego IDE robi to za Ciebie podczas tworzenia modułu funkcji z natychmiastową obsługą. Nie możesz ustawić tego elementu XML na true, jednocześnie ustawiając < dist:on-demand/ >. Niemniej jednak nadal możesz zażądać pobrania na żądanie modułów funkcji z funkcją natychmiastową jako natychmiastowych doświadczeń, korzystając z biblioteki Play Core. Gdy użytkownik pobierze i zainstaluje Twoją aplikację, urządzenie domyślnie pobiera i instaluje moduły funkcji aplikacji z obsługą natychmiastową wraz z podstawowym pakietem APK.

7. dist:title=„@string/feature_name": Określa tytuł modułu skierowany do użytkownika. Na przykład urządzenie może pokazywać ten tytuł, gdy zażąda potwierdzenia pobrania. Może być konieczne uwzględnienie zasobu ciągu dla tego tytułu w pliku module_root/src/source_set/res/values/strings.xml modułu podstawowego.

8. < /dist:module >: Określa, czy uwzględnić moduł w wielu pakietach APK przeznaczonych dla urządzeń z systemem Android 4.4 (poziom interfejsu API 20) lub niższym. Dodatkowo, jeśli zastosujesz narzędzie bundletool do generowania pakietów APK z pakietu aplikacji, tylko moduły funkcji, dla których ta właściwość ma wartość true, są uwzględniane w uniwersalnym pakiecie APK, który jest monolitycznym pakietem APK zawierającym kod i zasoby dla wszystkich konfiguracji urządzeń obsługiwanych przez Twoją aplikację.

9. < dist:delivery >: Używany do enkapsulacji opcji, które dostosowują dostarczanie modułów. Należy pamiętać, że każdy moduł funkcji musi skonfigurować tylko jeden rodzaj tych niestandardowych opcji dostawy.

10. < dist:install-time >: Określa, że moduł powinien być dostępny w czasie instalacji. Jest to domyślne zachowanie modułów funkcji, które nie określają innego typu niestandardowej opcji dostarczania. Może również określać warunki, które ograniczają moduł do urządzeń spełniających określone wymagania, takie jak funkcje urządzenia, kraj użytkownika lub minimalny poziom interfejsu API.

11. < dist:removable dist:value="prawda | false"/ >: Gdy nieskonfigurowana lub ustawiona na false, bundletool zlokalizuje moduły czasu instalacji w module podstawowym podczas generowania

podzielonych pakietów APK z pakietu. Ponieważ w wyniku przeniesienia będzie mniej podzielonych pakietów APK, to ustawienie może nieco zwiększyć wydajność Twojej aplikacji. Jednak gdy wymienny jest ustawiony na true: moduły czasu instalacji nie zostaną umieszczone w module podstawowym. Powinieneś ustawić to na true tylko wtedy, gdy chcesz odinstalować moduły w przyszłości. Ale jednocześnie skonfigurowanie zbyt wielu modułów do usunięcia może wydłużyć czas instalacji aplikacji. Czasami, gdy domyślnie ma wartość false, konieczne jest ustawienie tej wartości w manifeście, jeśli chcesz wyłączyć relokację dla modułu funkcji. Należy jednak pamiętać, że ta funkcja jest dostępna tylko podczas korzystania z wtyczki Android Gradle 4.2 lub podczas korzystania z narzędzia bundletool v1.0 z wiersza poleceń.

12.<dist:on-demand/>: Określa, że moduł powinien być dostępny do pobrania na żądanie. Oznacza to, że moduł nie jest dostępny w czasie instalacji, ale aplikacja może wymagać późniejszego pobrania. W tym rozdziale dowiedziałeś się, jak zainstalować i przygotować środowisko programistyczne do procesu konfiguracji Android Studio. Pokazaliśmy również, jak stosować skrypty Gradle Build oraz uruchamiać i nawigować w projektach Android Studio. Powinieneś teraz mieć podstawy wymagane do przeglądania i analizowania interfejsu użytkownika Android Studio z jego określonymi warunkami, różnymi folderami, układami i ciągami.