# Vaje 05

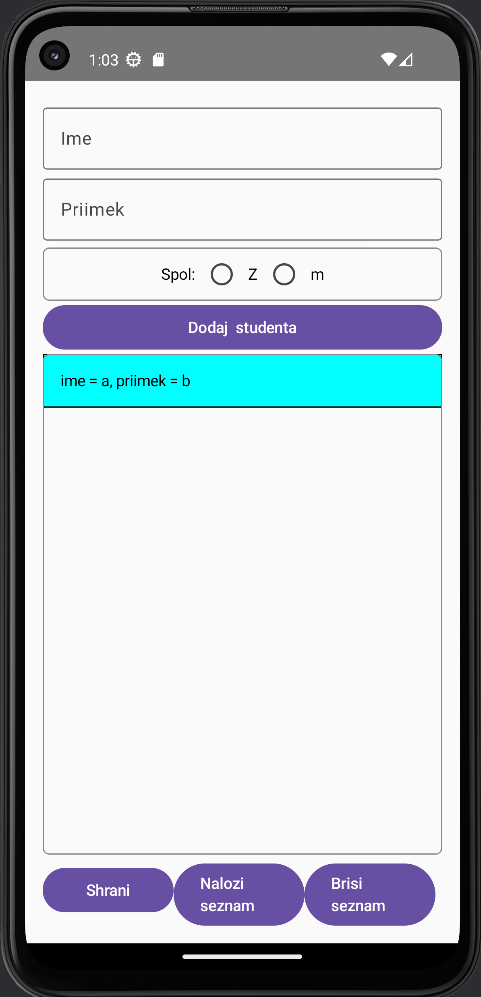
Cilj:

* Hranjenje podatkov v datoteko (JSON) na telefonu v lokal

**Naloga01** - Vaja05nal01SeznamStudentov

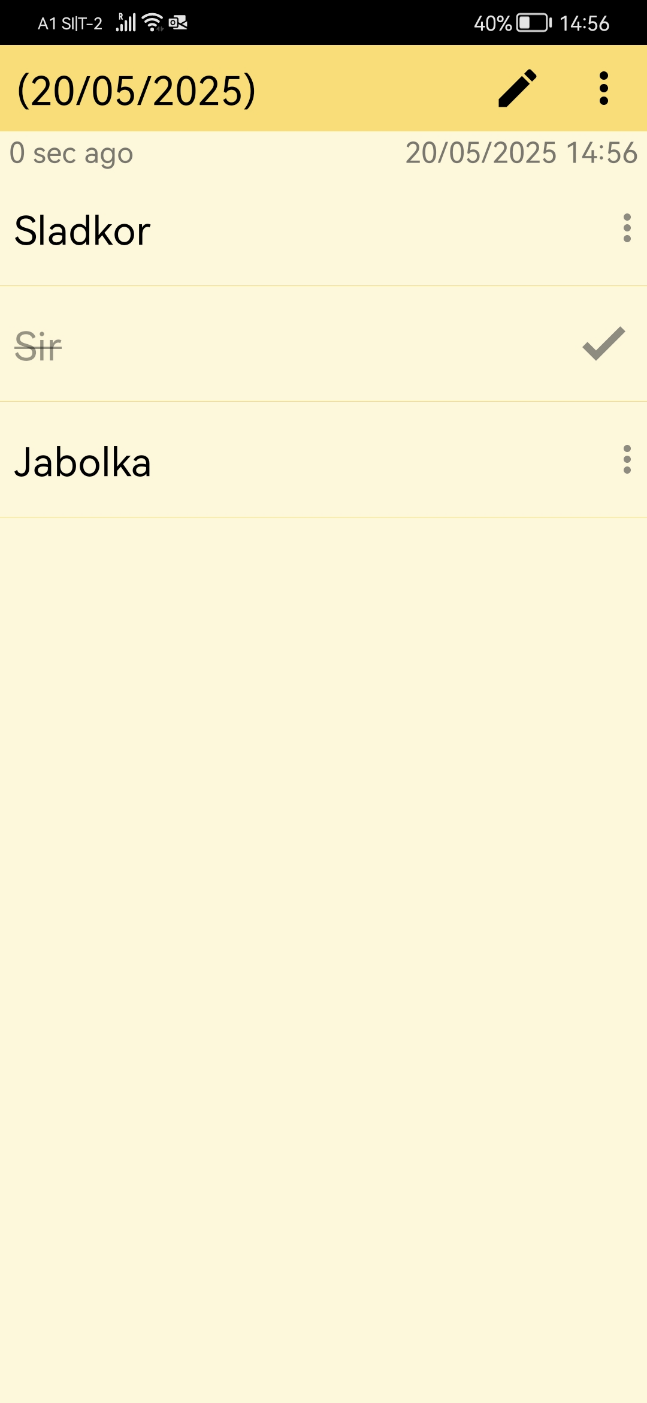
Razširi vajo tako, da podatke shranimo v datoteko.

Podatki se ob odpiranju programa iz datoteke shranijo v seznam objektov. Ob kliku na gumb SHRANI se podatki iz seznam objektov (Student) shranijo v JSON datoteko. Izdelaj še dva gumba za Izbriši podatke iz seznama.



**Naloga 02 -** Vaja05nal02SeznamNakup

Izdelaj seznam za nakup.



# Kaj in kako?

Dve funkciji:

1. Shrani seznam študentov v interno datoteko – ko želimo (lahko ob pritisku na gumb, lahko ob zapiranju programa). Imenovali jo bomo saveStudentToFile.
2. Napolni seznam študentov iz interne datoteke – ko želimo (najbolje ob zagonu programa, lahko tudi ob pritisku na bumb). Imenovali jo bomo

## Function Explanation

|  |
| --- |
| fun saveStudentToFile(context: Context, studentiList: List<Student>) {  val jsonString = Json.encodeToString(studentiList)  val file = File(context.filesDir, "student\_list.json")  file.writeText(jsonString)  } |

|  |  |  |
| --- | --- | --- |
| Json.encodeToString(studentiList) | Pretvori seznam (Študentov) v JSON String. | * Razred Student mora imeti določilo (anotation) @Serializable * Vključi knjižnico kotlinx.serialization |
| File(context.filesDir, "student\_list.json") | Ustvari datoteko v interni mapi za aplikacijo. | * Zasebna mapa samo za to aplikacijo. * Izbriše, ko odstranimo app. |
| file.writeText(jsonString) | V datoteko vpiše String. | * Če datoteka že obstaja, se prepiše. |

|  |
| --- |
| fun loadStudentsListFromFile(context: Context): List<Student> {  val file = File(context.filesDir, "student\_list.json")  return if (file.exists()) {  val jsonString = file.readText()  Json.decodeFromString(jsonString)  } else {  emptyList()  }  } |

|  |  |  |
| --- | --- | --- |
| File(context.filesDir, "student\_list.json") | Odpre datoteko. |  |
| file.readText() | V String shrani vsebino JSON datoteke. |  |
| Json.decodeFromString(jsonString) | Pretvori JSON v seznam študentov. |  |

### Json.encodeToString(studentiList)

* This **converts** your list of Student objects into a **JSON-formatted string**.
* Example output:

[

{"id":1,"ime":"Ana","priimek":"Novak","spol":"Z"},

{"id":2,"ime":"Luka","priimek":"Kranjc","spol":"M"}

]

✅ You need to make sure that:

* Your Student data class is annotated with @Serializable.
* You are using the kotlinx.serialization library.

### 🔹 2. val file = File(context.filesDir, "student\_list.json")

* This creates a File object that points to a file named student\_list.json in your app’s **internal storage directory**.
  + **What is an app’s internal storage directory?**

It’s a **private folder on the device** where your Android app can save files (like .json, images, or text) that:

* + - Are **only accessible to your app**
    - Are **automatically deleted** when your app is uninstalled
    - Do not require any permissions
* context.filesDir is a safe place to store private app data.
  + Path to internal directory
  + No permissions needed
  + File is deleted if the app is uninstalled

### 🔹 3. file.writeText(jsonString)

* This writes the JSON string into the file.
* If the file already exists, it **overwrites** it.

## What to include in your project

### 1. Kotlinx Serialization dependency

In build.gradle.kts (module):

dependencies {

implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.3")

}

If you're using version catalogs (libs.versions.toml), you can declare:

[libraries]

kotlinx-serialization-json = { module = "org.jetbrains.kotlinx:kotlinx-serialization-json", version = "1.6.3" }

And then use:

implementation(libs.kotlinx.serialization.json)

### 2. Apply Kotlin Serialization plugin

**In build.gradle.kts (module):**

plugins {

id("org.jetbrains.kotlin.plugin.serialization") version "1.9.0"

}

Or using version catalogs:

[plugins]

kotlin-serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version = "1.9.0" }

Then in the plugin block:

plugins {

alias(libs.plugins.kotlin.serialization)

}

### 3. Your Student class must be serializable

import kotlinx.serialization.Serializable

@Serializable

data class Student(

val id: Int,

val ime: String,

val priimek: String,

val spol: String

)

### 🔧 Optional: Toast on success

If you want to notify the user when the file is saved:

Toast.makeText(context, "Shranjeno!", Toast.LENGTH\_SHORT).show()

## ✅ Summary

| **Requirement** | **Why it's needed** |
| --- | --- |
| @Serializable | To make Student serializable to JSON |
| Kotlin plugin | Enables serialization compiler support |
| JSON library | Provides Json.encodeToString() |
| Context argument | To access internal storage for file writing |

# Context

## ✅ What is Context?

In Android, Context is an object that gives you access to **global information about your app and the system** — like:

* App resources (strings, images)
* File storage directories (like filesDir)
* Launching activities or services
* Getting system services (e.g. Wi-Fi manager)
* Showing Toasts

Think of Context as the **gateway to system features** and app-level information.

## ✅ Why your function needs Context

In this function:

fun saveStudentToFile(context: Context, studentiList: List<Student>) {

val jsonString = Json.encodeToString(studentiList)

val file = File(context.filesDir, "student\_list.json")

file.writeText(jsonString)

}

### 🔍 context.filesDir

This line:

context.filesDir

… accesses the **internal storage directory** for your app. You **can’t access this without Context**, because the file path is different for each app and managed by Android.

## 🧠 Without Context, the function would not know:

* Where to save the file
* What app it belongs to
* What resources it can use (e.g. Toasts)

## 🧩 Where do you get Context?

* In an Activity or Fragment, you can use this or requireContext()
* In Jetpack Compose, you use:

val context = LocalContext.current

## ✅ Summary

| **Feature you want** | **Needs Context?** | **Why?** |
| --- | --- | --- |
| Save file to storage | ✅ Yes | context.filesDir gives correct path |
| Show Toast | ✅ Yes | Needs to know which app to display in |
| Get resources (R.string) | ✅ Yes | Context holds app's resource map |
| Launch Activity/Service | ✅ Yes | System needs context of your app |

Let me know if you want a **diagram** or metaphor to visualize what Context really is — it's one of the most important building blocks in Android!

# Gradle?

**Gradle** is a **build system** — a tool that compiles, packages, and runs your Android app.

It handles:

* Which libraries you use (like Jetpack Compose, Kotlin serialization)
* What plugins you apply (like Kotlin Android support)
* Which SDK versions your app supports
* How the app is built, tested, and packaged

## You have **two Gradle files** in every Android project:

### 1. build.gradle.kts (Top-Level Project)

* Configures Gradle itself and version catalogs (like libs.versions.toml)
* You **don’t edit this often** unless adding plugin versions

### 2. build.gradle.kts (Module-level: :app)

This is **where all your app settings live** — let’s explain your key parts:

## EXPLAINED: Your Module-level build.gradle.kts

### 🔹plugins { ... }

plugins {

alias(libs.plugins.android.application)

alias(libs.plugins.kotlin.android)

alias(libs.plugins.kotlin.compose)

alias(libs.plugins.kotlin.serialization)

}

This activates the following features:

| **Plugin** | **What it does** |
| --- | --- |
| android.application | Lets you build Android APKs |
| kotlin.android | Lets you use Kotlin in Android |
| kotlin.compose | Enables Jetpack Compose support |
| kotlin.serialization | Lets you use @Serializable for saving/loading JSON |

You are using **aliases** from libs.versions.toml, which is modern and clean. ✅

### 🔹 android { ... }

android {

namespace = "com.example.vaja04naloga02seznamstudentovdemo"

compileSdk = 35

defaultConfig {

applicationId = "com.example.vaja04naloga02seznamstudentovdemo"

minSdk = 24

targetSdk = 35

versionCode = 1

versionName = "1.0"

testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"

}

buildTypes {

release {

isMinifyEnabled = false

proguardFiles(

getDefaultProguardFile("proguard-android-optimize.txt"),

"proguard-rules.pro"

)

}

}

compileOptions {

sourceCompatibility = JavaVersion.VERSION\_11

targetCompatibility = JavaVersion.VERSION\_11

}

kotlinOptions {

jvmTarget = "11"

}

buildFeatures {

compose = true

}

}

### ✅ What each part does:

| **Section** | **Meaning** |
| --- | --- |
| namespace | Your app's internal package name |
| compileSdk | The Android SDK version used to compile your app |
| defaultConfig | Basic app info (ID, min/max SDK, version code/name) |
| buildTypes | Defines how your app is built for release/debug |
| compileOptions | Java version compatibility |
| kotlinOptions | Kotlin JVM version (11 here) |
| buildFeatures.compose = true | Enables Jetpack Compose ✅ |

### 🔹 dependencies { ... }

dependencies {

implementation(libs.androidx.core.ktx)

implementation(libs.androidx.lifecycle.runtime.ktx)

implementation(libs.androidx.activity.compose)

implementation(platform(libs.androidx.compose.bom))

implementation(libs.androidx.ui)

implementation(libs.androidx.ui.graphics)

implementation(libs.androidx.ui.tooling.preview)

implementation(libs.androidx.material3)

implementation(libs.kotlinx.serialization.json) // ✅ JSON save/load support

testImplementation(libs.junit)

androidTestImplementation(libs.androidx.junit)

androidTestImplementation(libs.androidx.espresso.core)

androidTestImplementation(platform(libs.androidx.compose.bom))

androidTestImplementation(libs.androidx.ui.test.junit4)

debugImplementation(libs.androidx.ui.tooling)

debugImplementation(libs.androidx.ui.test.manifest)

}

### ✅ What these libraries are:

| **Library** | **Purpose** |
| --- | --- |
| core.ktx | Kotlin extensions for Android basics |
| lifecycle.runtime.ktx | ViewModel/lifecycle handling |
| activity.compose | Enables Compose in activities |
| compose.bom | BOM = Bill of Materials → aligns all Compose versions |
| material3 | Material Design 3 UI components |
| serialization.json | Convert objects to/from JSON (@Serializable) |
| junit, espresso, ui.test | Testing frameworks |
| ui.tooling.preview | Enables preview in Android Studio |
| ui.tooling, ui.test.manifest | Extra tools/debug features for Compose previews |

## ✅ What You Need to Make It Work (Checklist)

| **What you’re doing** | **What you need** |
| --- | --- |
| Jetpack Compose UI | buildFeatures.compose = true, Compose libraries |
| Kotlin | kotlin.android plugin |
| JSON save/load | kotlinx.serialization plugin + dependency |
| Internal storage | Use context.filesDir (no permission needed) |
| Project settings | compileSdk, targetSdk, minSdk in android block |

## ✅ Summary

* Gradle defines **how your app is built and what it uses**
* You have:
  + Plugins: turn on Android, Kotlin, Compose, Serialization
  + Android config: sets SDKs, versioning, release types
  + Dependencies: tell Gradle what libraries to include
* You’re using **modern best practices** (version catalogs with libs.plugins and libs.androidx ✅)

Since your Gradle uses **version catalogs** (with alias(libs.plugins...)), let me show you **exactly what you need to add** using that method — just **one clean solution**, tailored to your setup.

## 1. Update libs.versions.toml

This file is in the gradle folder of your project.

Add this to enable the **Kotlin serialization plugin** and the required library:

[plugins]

kotlin-serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version = "1.9.0" }

[libraries]

kotlinx-serialization-json = { module = "org.jetbrains.kotlinx:kotlinx-serialization-json", version = "1.6.3" }

Make sure the version of Kotlin (1.9.0) matches your kotlin.compilerVersion.

## 2. Use aliases in build.gradle.kts (app-level)

Modify your plugin and dependency sections like this:

plugins {

alias(libs.plugins.android.application)

alias(libs.plugins.kotlin.android)

alias(libs.plugins.kotlin.compose)

alias(libs.plugins.kotlin.serialization) // ✅ enables @Serializable

}

dependencies {

implementation(libs.kotlinx.serialization.json) // ✅ for Json.encodeToString()

// your other dependencies...

}

## Then:

* Sync Gradle (Android Studio will prompt)
* Rebuild the project

## Why this works:

| **Feature** | **Required** | **How you're enabling it** |
| --- | --- | --- |
| @Serializable | Serialization plugin | alias(libs.plugins.kotlin.serialization) |
| Json.encodeToString() | JSON serialization library | libs.kotlinx.serialization.json |
| Compatibility | Kotlin version match | Declared in .toml file |

# What is **Serialization**?

**Serialization** means **converting an object into a format that can be saved or sent** — usually into a text format like **JSON** or **XML**.

### And the reverse process?

It’s called **deserialization**: converting that text **back into the original object**.

## Why is this useful?

Because in apps, we often want to:

* Save data to a file (e.g. List<Student> → file)
* Send data over the network (e.g. to a server)
* Store app settings
* Persist objects between app restarts

But we can’t store complex objects like Student directly — we need to **serialize** them into plain text first (like JSON).

## Example: Kotlin + JSON

Given this class:

@Serializable

data class Student(val id: Int, val ime: String, val priimek: String, val spol: String)

### Serialize to JSON:

val student = Student(1, "Ana", "Novak", "Z")

val json = Json.encodeToString(student)

json will now contain:

{"id":1,"ime":"Ana","priimek":"Novak","spol":"Z"}

### Deserialize from JSON:

val loaded = Json.decodeFromString<Student>(json)

Now loaded is again a real Student object.

## What do you need in Kotlin?

### 1. Annotate your data class with:

@Serializable

### 2. Use the Kotlinx Serialization library:

implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.3")

### 3. Apply the plugin:

plugins {

id("org.jetbrains.kotlin.plugin.serialization") version "1.9.0"

}

## Summary

| **Concept** | **Meaning** |
| --- | --- |
| Serialization | Object → JSON/text |
| Deserialization | JSON/text → Object |
| Needed for | Saving to file, network communication, shared preferences |
| Tool used in Kotlin | kotlinx.serialization |