File Format

Last edited by **Priya Ashok Sardhara** 2 days ago

File Format

For storing and managing data in our project, we have selected **JSON** as our primary file format. JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for both humans to read and write, and easy for machines to parse and generate. It provides a structured way to store key-value pairs and hierarchical data, making it ideal for our project's requirements.

Why JSON?

- **Human-readable and lightweight**: JSON files are easy to understand and edit manually if needed.
- **Supports hierarchical data**: JSON naturally supports nested data structures, which align well with the object-oriented nature of our project.
- **Compatible with Java libraries**: There are multiple well-supported Java libraries for parsing and writing JSON, such as:
    - org.json (Java's built-in JSON library)
    - Gson (Google's JSON library)
    - Jackson (Fast and flexible JSON parser)

Structure of the JSON File

We will use JSON to store relevant project data, such as user profiles, game state, settings, and preferences. The structure of the JSON file is as follows:

json

CopyEdit

```
{
 "users": [
  {
   "id": 1,
   "username": "PlayerOne",
   "preferences": {
    "difficulty": "Hard"
   },
   "progress": {
    "level": 5,
    "score": 3200,
    "achievements": ["First Win", "Sharp Shooter"]
   }
  },
  {
   "id": 2,
   "username": "PlayerTwo",
   "preferences": {
    "difficulty": "Medium"
   },
   "progress": {
    "level": 3,
    "score": 1500,
    "achievements": ["Beginner"]
   }
  }
 ]
}
```

Handling JSON in Java

We will use **Gson** to handle JSON serialization and deserialization in Java. Below is a sample Java implementation for reading and writing JSON:

**Writing Data to JSON:
**java
CopyEdit

```java
import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import java.io.FileWriter;

import java.io.IOException;

public class JSONWriter {

    public static void saveData(Object data, String filePath) {

        Gson gson = new GsonBuilder().setPrettyPrinting().create();

        try (FileWriter writer = new FileWriter(filePath)) {

            gson.toJson(data, writer);

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

Reading Data from JSON:
**java
CopyEdit

```java
import com.google.gson.Gson;

import java.io.FileReader;

import java.io.IOException;

public class JSONReader {

    public static <T> T loadData(String filePath, Class<T> clazz) {

        Gson gson = new Gson();

        try (FileReader reader = new FileReader(filePath)) {

            return gson.fromJson(reader, clazz);

        } catch (IOException e) {

            e.printStackTrace();

            return null;

        }

    }

}
```

JSON is the best fit because it balances **readability, ease of use, and compatibility with Java**.