UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

LABORATORIO DE BASES DE DATOS 2

SEGUNDO SEMESTRE 2023

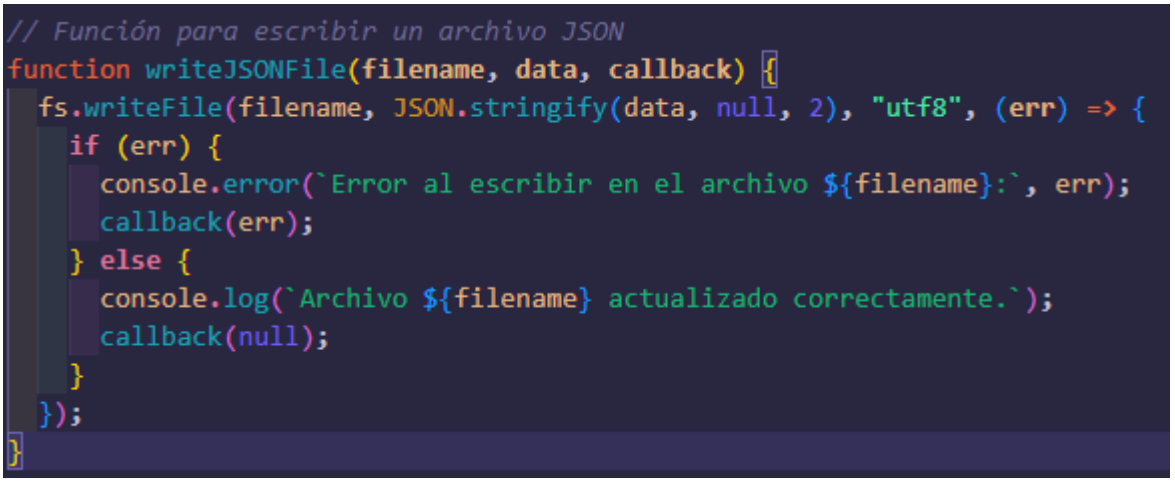
PROYECTO 2

GRUPO 18

Nombre	**Carnet**
Javier Alejandro Gutierrez de León	202004765
Enrique Alejandro Pinula Quiñonez	202004707
Gerson Rubén Quiroa del Cid	202000166

La función readJSONFile se utiliza para leer un archivo JSON desde el sistema de archivos y luego parsear su contenido.

La función utiliza el módulo fs (sistema de archivos) de Node.js para leer el contenido del archivo especificado. Si la lectura tiene éxito, se intenta parsear el contenido del archivo como JSON utilizando JSON.parse. Si la operación de análisis tiene éxito, se llama a la función de retorno con null como error y el objeto JSON parseado como datos.



De la misma manera para escribir un archivo json.

```
// Función para procesar los géneros
function processGenres(genresData, gameData, genreData) {
  genresData.Genres.forEach((genre) => {
    const foundGame = gameData.Game.find((g) => g.game === genre.game);
    if (foundGame) {
      const foundGenre = genreData.Genre.find((g) => g.genre === genre.genre);

      if (foundGenre) {
        foundGame.genres = foundGame.genres || [];
        foundGame.genres.push(foundGenre);
        console.log(
          `Género con ID ${genre.genre} agregado al juego con ID ${genre.game}`
        );
      } else {
        console.log(`No se encontró el género con ID ${genre.genre}`);
      }
    } else {
      console.log(`No se encontró el juego con ID ${genre.game}`);
    }
  });

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "genres" agregado a game.json correctamente.');
    }
  });
}
```

Recorremos la tabla de muchos a muchos entre games y genres para poder así relacionar los géneros con los juegos y agregarle a json de games un atributo llamado genres con todos los géneros de cada juego.

```
// Función para procesar los involvedcompany
function processInvolvedCompany(involvedCompanyData, companyData) {
  involvedCompanyData.InvolvedCompany.forEach((involvedCompany) => {
    const foundCompany = companyData.Company.find(
      (c) => c.company === involvedCompany.company
    );
    if (foundCompany) {
      involvedCompany.company = foundCompany;
      console.log(
        `Compañía con ID ${foundCompany.company} agregada al InvolvedCompany con ID ${inv
      );
    } else {
      console.log(`No se encontró el juego con ID ${genre.game}`);
    }
  });

  writeJSONFile(
    "./outputJsons/involvedcompany.json",
    involvedCompanyData,
    (err) => {
      if (!err) {
        console.log(
          'Campo "company" agregado a involvedcompany.json correctamente.'
        );
      }
    }
  );
}
```

Recorremos la tabla de involved companies para poder agregar un nuevo atributo en dicha tabla con el nombre de company, de esta manera cada involved company tiene un company dentro del json.

```
function processInvolvedCompanies(
  involvedCompaniesData,
  gameData,
  involvedCompanyData
) {
  involvedCompaniesData.InvolvedCompanies.forEach((i) => {
    const foundGame = gameData.Game.find((g) => g.game === i.game);
    if (foundGame) {
      const foundInvolvedCompany = involvedCompanyData.InvolvedCompany.find(
        (ic) => ic.involved_company === i.involved_company
      );

      if (foundInvolvedCompany) {
        foundGame.involved_companies = foundGame.involved_companies || [];
        foundGame.involved_companies.push(foundInvolvedCompany);
        console.log(
          `InvolvedCompany con ID ${i.involved_company} agregado al juego con ID ${i.game}`
        );
      } else {
        console.log(
          `No se encontró el InvolvedCompany con ID ${i.involved_company}`
        );
      }
    } else {
      console.log(`No se encontró el juego con ID ${i.game}`);
    }
  });

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log(
        'Campo "involved_company" agregado a game.json correctamente.'
```

Luego, con la tabla de muchos a muchos entre Games y InvolvedCompanies relacionamos las compañías desarrolladoras con los juegos. Agregamos al json de games un atributo nuevo llamado involved_companies con todos las compañías desarrolladoras de cada juego.

```
// Función para procesar los collections
function processCollection(gameData, collectionData) {
  gameData.Game.forEach((game) => {
    const foundCollection = collectionData.Collection.find(
      (c) => c.collection === game.collection
    );
    if (foundCollection) {
      game.collection = foundCollection;
      console.log(
        `Colleccion con ID ${foundCollection.collection} agregado al Game con ID ${game.game}`
      );
    } else {
      console.log(`No se encontró la coleccion con ID ${game.collection}`);
    }
  });

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "collection" agregado a game.json correctamente.');
    }
  });
}
```

Con collectión, como cada juego tenía su id de una colección, solo se iteraron los juegos, se buscó cada id de colección en el json de Colletion y se añadió dicha colección en un atributo llamado collection en cada juego.

```
// Función para procesar los platforms
function processPlatforms(platformsData, gameData, platformData) {
  platformsData.Platforms.forEach((platforms) => {
    const foundGame = gameData.Game.find((g) => g.game === platforms.game);
    if (foundGame) {
      const foundPlatform = platformData.Platform.find(
        (p) => p.platform === platforms.platform
      );

      if (foundPlatform) {
        foundGame.platforms = foundGame.platforms || [];
        foundGame.platforms.push(foundPlatform);
        console.log(
          `Plataforma con ID ${platforms.platform} agregado al juego con ID ${platforms.game}`
        );
      } else {
        console.log(
          `No se encontró la plataforma con ID ${platforms.platform}`
        );
      }
    } else {
      console.log(`No se encontró el juego con ID ${platforms.game}`);
    }
  });

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "platforms" agregado a game.json correctamente.');
    }
  });
```

con la tabla de muchos a muchos entre Games y Platform relacionamos las plataformas con los juegos. Agregamos al json de games un atributo nuevo llamado platforms con todas las plataformas disponibles en cada juego.

```
// Función para procesar los region enum
function processRegionEnum(releaseDateData, regionEnumData) {
  releaseDateData.ReleaseDate.forEach((release) => {
    const foundRegionEnum = regionEnumData.RegionEnum.find(
      (r) => r.region_enum === release.region_enum
    );
    if (foundRegionEnum) {
      release.region_enum = foundRegionEnum;
      console.log(
        `RegionEnum con ID ${release.region_enum.region_enum} agregado al releasedate con ID
      );
    } else {
      console.log(
        `No se encontró el region enum con ID ${release.region_enum}`
      );
    }
  });

  writeJSONFile("./outputJsons/regionenum.json", releaseDateData, (err) => {
    if (!err) {
      console.log(
        'Campo "region_enum" agregado a regionenum.json correctamente.'
      );
    }
  });
}
```

Para la tabla ReleaseDate se tuvo que crear un nuevo atributo llamado region_enum donde dicha información se encontraba en la tabla RegionEnum, mediante la cual la pudimos relacionar ya que cada release date tenía un id de region_enum.

```
// Función para procesar los ReleaseDate
function processReleaseDate(releaseDateData, gameData) {
  releaseDateData.ReleaseDate.forEach((release) => {
    const foundGame = gameData.Game.find((g) => g.game === release.game);
    if (foundGame) {
      foundGame.release_date = foundGame.release_date || [];
      foundGame.release_date.push(release);
      console.log(
        `ReleaseDate con ID ${release.release_date} agregado al Juego con ID ${release.game}`
      );
    } else {
      console.log(`No se encontró el Juego con ID ${release.game}`);
    }
  });

  gameData.Game[0];
  getChar();

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "release_date" agregado a game.json correctamente.');
    }
  });
}
```

Cada release_date tenía también un id de juego, por lo cual mediante esta manera se pudo relacionar cada release_date con un juego. Se creó en el json de games, un nuevo atributo llamado release_date con todos las fechas de lanzamiento que contiene un juego.

```
function processPlayerPerspective(
  playerPerspectivesData,
  playerPerspectiveData
) {
  playerPerspectivesData.PlayerPerspectives.forEach((pps) => {
    const foundPlayerPerspective = playerPerspectiveData.PlayerPerspective.find(
      (pp) => pp.player_perspective === pps.player_perspective
    );
    if (foundPlayerPerspective) {
      pps.player_perspective = foundPlayerPerspective;
      console.log(
        `PlayerPerspective con ID ${pps.player_perspective.player_perspective} agregado al player_perspe
      );
    } else {
      console.log(
        `No se encontró el PlayerPerspective con ID ${pps.player_perspective}`
      );
    }
  });

  console.log(playerPerspectivesData.PlayerPerspectives[0]);
  getChar();

  writeJSONFile(
    "./outputJsons/playerperspectives.json",
    playerPerspectivesData,
    (err) => {
      if (!err) {
        console.log(
          'Campo "player_perspective" agregado a playerperspectives.json correctamente.'
        );
      }
```

Luego para cada player_perfective se tuvo que relacionar con un player_perfectives por lo que se agregó un atributo nuevo llamado player_perpective para cada perspectiva de juego.

```
// Función para procesar los PlayerPerspectives
function processPlayerPerspectives(playerPerspectivesData, gameData) {
  playerPerspectivesData.PlayerPerspectives.forEach((pp) => {
    const foundGame = gameData.Game.find((g) => g.game === pp.game);
    if (foundGame) {
      foundGame.player_perspectives = foundGame.player_perspectives || [];
      foundGame.player_perspectives.push(pp);
      console.log(
        `PlayerPerspectives con ID ${pp.player_perspectives} agregado al Juego con ID ${pp.game}`
      );
    } else {
      console.log(`No se encontró el Juego con ID ${pp.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log(
        'Campo "player_perspectives" agregado a game.json correctamente.'
      );
    }
  });
}
```

Para relacionar cada perspectiva de juego se utilizó el id de juego que tenía cada player_perfective para relacionarlo así con un juego y en el json de games se agregó un nuevo atributo llamado player_perfecives con cada una de las perspectivas de juego que contiene dicho juego.

```
// Función para procesar los Themes
function processThemes(themesData, gameData, themeData) {
  themesData.Themes.forEach((theme) => {
    const foundGame = gameData.Game.find((g) => g.game === theme.game);
    if (foundGame) {
      const foundTheme = themeData.Theme.find((t) => t.theme === theme.theme);

      if (foundTheme) {
        foundGame.themes = foundGame.themes || [];
        foundGame.themes.push(foundTheme);
        console.log(
          `Theme con ID ${theme.theme} agregado al juego con ID ${theme.game}`
        );
      } else {
        console.log(`No se encontró el Theme con ID ${theme.theme}`);
      }
    } else {
      console.log(`No se encontró el juego con ID ${theme.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "themes" agregado a game.json correctamente.');
    }
  });
}
```

con la tabla de muchos a muchos entre Games y Theme relacionamos los temas con los juegos. Agregamos al json de games un atributo nuevo llamado themes con todos los temas de cada juego.

```
// Función para procesar los Region
function processRegion(gameLocalizationData, regionData) {
  gameLocalizationData.GameLocalization.forEach((gl) => {
    const foundRegion = regionData.Region.find((r) => r.region === gl.region);
    if (foundRegion) {
      gl.region = foundRegion;
      console.log(
        `Region con ID ${gl.region.region} agregado al GameLocalization con ID ${gl.game_localization}`
      );
    } else {
      console.log(`No se encontró el Region con ID ${gl.region}`);
    }
  });

  console.log(gameLocalizationData.GameLocalization[0]);
  getChar();

  writeJSONFile(
    "./outputJsons/gamelocalization.json",
    gameLocalizationData,
    (err) => {
      if (!err) {
        console.log(
          'Campo "region" agregado a gamelocalization.json correctamente.'
        );
      }
    }
  );
}
```

Se agregó para cada game_localization su respectivo región por medio del id de región contenido en cada game_localization, de esta manera se creó un nuevo atributo con el nombre de game_localization.

```
// Función para procesar los GameLocalization
function processGameLocalization(gameLocalizationData, gameData) {
  gameLocalizationData.GameLocalization.forEach((gl) => {
    const foundGame = gameData.Game.find((g) => g.game === gl.game);
    if (foundGame) {
      foundGame.game_localization = foundGame.game_localization || [];
      foundGame.game_localization.push(gl);
      console.log(
        `GameLocalization con ID ${gl.game_localization} agregado al Juego con ID ${gl.game}`
      );
    } else {
      console.log(`No se encontró el Juego con ID ${gl.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game.json", gameData, (err) => {
    if (!err) {
      console.log(
        'Campo "game_localization" agregado a game.json correctamente.'
      );
    }
  });
}
```

Se relacionó cada game_localization por medio del id de juego que contiene cada localización de juego. De esta manera se agregó un nuevo atributo en el json games para guardar la información de cada game_localization de cada juego.

```
// Función para procesar los LanguageSupportType y Language
function processLanguage(
  languageSupportData,
  languageSupportTypeData,
  languageData
) {
  languageSupportData.LanguageSupports.forEach((ls) => {
    const foundLanguageSupportType =
      languageSupportTypeData.LanguageSupportType.find(
        (lsp) => lsp.language_support_type === ls.language_support_type
      );
    if (foundLanguageSupportType) {
      ls.language_support_type = foundLanguageSupportType;
      console.log(
        `LanguageSupportType con ID ${ls.language_support_type.language_support_type} agregado al
      );
    } else {
      console.log(
        `No se encontró el LanguageSupportType con ID ${ls.language_support_type}`
      );
    }
    const foundLanguage = languageData.Languages.find(
      (l) => l.language === ls.language
    );
    if (foundLanguage) {
      ls.language = foundLanguage;
      console.log(
        `Language con ID ${ls.language.language} agregado al LanguageSupport con ID ${ls.language_
      );
    } else {
      console.log(`No se encontró el Language con ID ${ls.language}`);
    }
```

Para relacionar el lenguaje de soporte y el tipo de lenguaje se utilizó la tabla LenguajeSupport que contiene el id de ambos objetivos. De esta manera se añadió un nuevo atributo al json de LenguajeSupport con language_support_type y language.

```
// Función para procesar los LanguageSupport
function processLanguageSupport(languageSupportData, gameData) {
  languageSupportData.LanguageSupports.forEach((ls) => {
    const foundGame = gameData.Game.find((g) => g.game === ls.game);
    if (foundGame) {
      foundGame.language_support = foundGame.language_support || [];
      foundGame.language_support.push(ls);
      console.log(
        `LanguageSupport con ID ${ls.language_support} agregado al Juego con ID ${ls.game}`
      );
    } else {
      console.log(`No se encontró el Juego con ID ${ls.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  // Dividir el archivo JSON en dos partes
  const halfwayPoint = Math.ceil(gameData.Game.length / 2);
  const part1 = { Game: gameData.Game.slice(0, halfwayPoint) };
  const part2 = { Game: gameData.Game.slice(halfwayPoint) };
  writeJSONFile("./outputJsons/game_part1.json", part1, (err) => {
    if (!err) {
      console.log('Campo "language_support" agregado a game_part1.json correctamente.');
    }
  });
  writeJSONFile('./outputJsons/game_part2.json', part2, (err) => {
    if (!err) {
      console.log('Campo "language_support" agregado a game_part2.json correctamente.');
    }
  });
```

Se añadieron los lenguajes de soporte en cada juego tomando como referencia los id de juego que contiene cada language_support. De esta manera se añadió un nuevo atributo en cada juego llamado language_support.

```
// Función para procesar los AlternativeName
function processAlternativeName(alternativeNameData, gameData) {
  alternativeNameData.AlternativeName.forEach((an) => {
    const foundGame = gameData.Game.find((g) => g.game === an.game);
    if (foundGame) {
      foundGame.alternative_name = foundGame.alternative_name || [];
      foundGame.alternative_name.push(an);
      console.log(
        `AlternativeName con ID ${an.alternative_name} agregado al Juego con ID ${an.g
      );
    } else {
      console.log(`No se encontró el Juego con ID ${an.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game_part2.json", gameData, (err) => {
    if (!err) {
      console.log(
        'Campo "alternative_name" agregado a game.json correctamente.'
      );
    }
  });
}
```

para los nombres alternativos se utilizó el id de cada alternative_name para relacionarlo con cada juego. De esta manera se añadió un nuevo atributo al json game con el nombre de alternative_name con todos su nombres alternativos.

```
// Función para procesar los GameModes
function processGameModes(gameModesData, gameData, gameModeData) {
  gameModesData.GameModes.forEach((gm) => {
    const foundGame = gameData.Game.find((g) => g.game === gm.game);
    if (foundGame) {
      const foundGameMode = gameModeData.GameMode.find(
        (g) => g.game_mode === gm.game_mode
      );

      if (foundGameMode) {
        foundGame.game_modes = foundGame.game_modes || [];
        foundGame.game_modes.push(foundGameMode);
        console.log(
          `GameMode con ID ${gm.game_mode} agregado al juego con ID ${gm.game}`
        );
      } else {
        console.log(`No se encontró el GameMode con ID ${gm.game_mode}`);
      }
    } else {
      console.log(`No se encontró el juego con ID ${gm.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game_part2.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "game_modes" agregado a game.json correctamente.');
    }
  });
}
```

con la tabla de muchos a muchos entre Games y GameMode relacionamos los modos de juegos con los juegos. Agregamos al json de games un atributo nuevo llamado game_modes con todos los modos de juego disponible de cada juego.

```
// Función para procesar los GameEngines
function processGameEngines(gameEnginesData, gameData, gameEngineData) {
  gameEnginesData.GameEngines.forEach((ge) => {
    const foundGame = gameData.Game.find((g) => g.game === ge.game);
    if (foundGame) {
      const foundGameEngine = gameEngineData.GameEngine.find(
        (g) => g.game_engine === ge.game_engine
      );

      if (foundGameEngine) {
        foundGame.game_engines = foundGame.game_engines || [];
        foundGame.game_engines.push(foundGameEngine);
        console.log(
          `GameEngine con ID ${ge.game_engine} agregado al juego con ID ${ge.game}`
        );
      } else {
        console.log(`No se encontró el GameEngine con ID ${ge.game_engine}`);
      }
    } else {
      console.log(`No se encontró el juego con ID ${ge.game}`);
    }
  });

  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game_part2.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "game_engines" agregado a game.json correctamente.');
    }
```

con la tabla de muchos a muchos entre Games y GameEngines relacionamos las tecnologías de juegos con los juegos. Agregamos al json de games un atributo nuevo llamado game_engines con todos las tecnologías de juego disponible de cada juego.

```
// Función para procesar los Franchises
function processFranchises(franchisesData, gameData, franchiseData) {
  franchisesData.Franchises.forEach((frs) => {
    const foundGame = gameData.Game.find((g) => g.game === frs.game);
    if (foundGame) {
      const foundFranchise = franchiseData.Franchise.find(
        (g) => g.franchise === frs.franchise
      );

      if (foundFranchise) {
        foundGame.franchises = foundGame.franchises || [];
        foundGame.franchises.push(foundFranchise);
        console.log(
          `Franchise con ID ${frs.franchise} agregado al juego con ID ${frs.game}`
        );
      } else {
        console.log(`No se encontró el Franchise con ID ${frs.franchise}`);
      }
    } else {
      console.log(`No se encontró el juego con ID ${frs.game}`);
    }
  });
                        any
  console.log(gameData.Game[0]);
  getChar();

  writeJSONFile("./outputJsons/game_part2.json", gameData, (err) => {
    if (!err) {
      console.log('Campo "franchises" agregado a game.json correctamente.');
    }
  });
}
```

con la tabla de muchos a muchos entre Games y Franchise relacionamos las franquicias con los juegos. Agregamos al json de games un atributo nuevo llamado franchises con todas las franquicias de cada juego.

```
// Función para procesar los géneros
function processGamesGenres(genresData, gameData, genreData) {
  genresData.Genres.forEach((genre) => {
    const foundGame = gameData.Game.find((g) => g.game === genre.game);
    if (foundGame) {
      const foundGenre = genreData.Genre.find((g) => g.genre === genre.genre);

      if (foundGenre) {
        const gamesInput = { game: foundGame.game, name: foundGenre.name, platforms: foundGa
        foundGenre.games = foundGenre.games || [];
        foundGenre.games.push(gamesInput);
        console.log(
          `Juego con ID ${genre.game} agregado al genero con ID ${genre.genre}`
        );
      } else {
        console.log(`No se encontró el género con ID ${genre.genre}`);
      }
    } else {
      console.log(`No se encontró el juego con ID ${genre.game}`);
    }
  });

  writeJSONFile("./outputJsons/genres.json", genreData, (err) => {
    if (!err) {
      console.log('Campo "games" agregado a genres.json correctamente.');
    }
  });
}
```

Añadimos un nuevo json, esto por razones de las consultas, en cada genero del json Genres se agregó un atributo llamado games, donde se encuentra todos los juegos relacionados con cada género.

Por último, importamos los jsons que generamos para guardarlos en colecciones, en este caso solo se crearon 3 colecciones: Games, Genres y Platforms. Utilizamos el siguiente comando:

- mongoimport –db videojuegos –collection games –file games.json
- mongoimport –db videojuegos –collection genres –file genres.json
- mongoimport –db videojuegos –collection platforms –file platforms.json