Prison Database Management

Name: Abdulmohaimin Bashir

Student Number: B2205.010021

T**he Project’s Purpose:**

The main goal of the project is to create a reliable system that organises the management system of a prison. Placing the appropriate data into the table, and keeping track of information. This project seeks to provide a robust and scalable solution for managing diverse aspects of a prison facility, ensuring effective data handling and streamlined operations.

The project will consist of a single database containing 4 tables, each having their own fields and establishing relationships with their appropriate tables. These tables are as follows, “*Prisoners*” handling the prisoners’ information “*Labour*” handling the prisoners' assigned labour tasks, “*Cells*” managing information about different cell types and their associations with prisoners, and “*Districts*” handling the organization of cells within specific districts.

**The Project’s Files:**

Prisoner Prisoner Repository Prisoner Controller

Labour Labour Repository Labour Controller

Cell Cell Repository Cell Controller

District District Repository District Controller

The project uses **Wampserver** to establish a database in **MySQL Workbench**. The connection is established within the “*application.properties*”. The dependencies are:

1. **Spring Boot DevTools:** Provides many features among which, live reloading and automatic application restart
2. **Spring Web:** Offers features to simply web development, such as, managing web complement, HTTP requests, building RESTful services
3. **Spring Data JPA:** Provides high-level of abstraction for accessing files Java application.
4. **MySQL Driver:** Offers data manipulation and retrieval between Java applications and MYSQL database.

**Entities:**

**District:**

@Entity

public class District {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    Long id;

    @Column(name = "District Name")

    String districtName;

    @Column(name = "District Location")

    String districtLocation;

    @OneToMany(mappedBy = "district", cascade = CascadeType.ALL)

    @JsonIgnore

    List<Cell> cell;

    // Getters and Setters

}

**Cell:**

@Entity

public class Cell {

    @Id

    @GeneratedValue (strategy = GenerationType.IDENTITY)

    Long id;

    @Column (name = "Cell Type")

    String cellType;

    @OneToOne(mappedBy = "cell", cascade = CascadeType.ALL)

    @JsonIgnore

    Prisoner prisoner;

    @ManyToOne

    @JoinColumn(name = "district\_id")

    District district;

    // Getters and Setters

}

**Prisoner:**

@Entity

public class Prisoner {

    @Id

    @GeneratedValue (strategy = GenerationType.IDENTITY)

    Long id;

    @Column (name = "Name")

    String name;

    @Column (name = "Surname")

    String surname;

    @Column (name = "Age")

    int age;

    @Column (name = "YearsOfImprisonment")

    int yearsOfImprisonment;

    @Column (name = "CrimeType")

    String crimeType;

    @ManyToMany

    @JoinTable(name = "prisoner\_labour",

            joinColumns = @JoinColumn(name = "prisoner\_id"),

            inverseJoinColumns = @JoinColumn(name = "labour\_id"))

    List<Labour> labour;

    @OneToOne

    Cell cell;

    // Getters and Setters

}

**Labour:**

@Entity

public class Labour {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    Long id;

    @Column(name = "Labour Type")

    String labourType;

    @Column(name = "Labour Hours")

    int labourHours;

    @ManyToMany(mappedBy = "labour", cascade = CascadeType.ALL)

    @JsonIgnore

    List<Prisoner> prisoner;

    // Getters and Setters

}

**Annotations Explainations:**

* **@Entity**: Marks a class as a JPA entity, representing a table in a relational database.
* **@Column(name = "Name")**: Specifies the mapping of a field or property to a column with the given name in the database.
* **(mappedBy = "entity", cascade = CascadeType.ALL):** specifying that the attribute "entity" in the associated entity is the owning side of the relationship, and defining cascade operations for the relationship.
* **@ManyToOne & @OneToMany**: Defines a many-to-one & one-to-many relationship between entities.
* **@OneToOne**: Indicates a one-to-one relationship between entities.
* **@ManyToMany**: Defines a many-to-many relationship between entities.
* **@JoinTable**: Specifies the join table for the many-to-many relationship, including the table name and the columns used for joining.
* **@JsonIgnore**: Instructs the serialization process to ignore the annotated field, preventing circular references or unwanted data from being included in the JSON representation.

**Repository:**

“**District**” and “**Labour**” have no special functions in “**Repository**”:

public interface DistrictRepository extends JpaRepository<District, Long> {

}

public interface LabourRepository extends JpaRepository<Labour, Long>{

}

**Cell:**

public interface CellRepository extends JpaRepository<Cell, Long>{

    List<Cell> findByDistrictId(Long id);

}

**Prisoner:**

public interface PrisonerRepository extends JpaRepository<Prisoner, Long>{

    List<Prisoner> findByCellId(Long id);

    List<Prisoner> findByLabourId(Long id);

}

The repositories extend from “**JpaRepository”** which contain multiple prebuilt functions.  
The “**findByEntityId”** method name follows the Spring Data JPA naming convention for query derivation

**Controller:**

All 4 controllers contain have the following CRUD operations:

* Returns a list of all entities’ record.
* Returns the entities’ record with the specified ID.
* Adds a new entity record to the database.
* Adds a list of entities records to the database.
* Updates an existing entity record with the specified ID.
* Deletes the entity’s record with the specified ID.
* Deletes all entity records in the database.

**District:**

@RestController

@RequestMapping("district")

public class DistrictController {

    @Autowired

    DistrictRepository districtRep;

    @GetMapping

    public List<District> getDistricts() {

        return districtRep.findAll();

    }

    @GetMapping("{id}")

    public Optional<District> getDistrict(@PathVariable Long id) {

        return districtRep.findById(id);

    }

    @PostMapping(value = "addDistrict", consumes = "application/json")

    public String addDistrict(@RequestBody District district) {

        districtRep.save(district);

        return "District Added";

    }

    @PostMapping(value = "addDistricts", consumes = "application/json")

    public String addDistricts(@RequestBody List<District> district) {

        districtRep.saveAll(district);

        return "Districts Added";

    }

@PutMapping("updateDistrict/{id}")

    public String updateDistrict(@RequestBody District district, @PathVariable Long id) {

        District existingDistrict = districtRep.findById(id).get();

        existingDistrict.setDistrictName(district.getDistrictName());

        existingDistrict.setDistrictLocation(district.getDistrictLocation());

        districtRep.save(existingDistrict);

        return "District Updated";

    }

    @DeleteMapping("deleteDistrict/{id}")

    public String deleteDistrict(@PathVariable Long id) {

        districtRep.deleteById(id);

        return "District Deleted";

    }

    @DeleteMapping("deleteAllDistricts")

    public String deleteAllDistricts() {

        districtRep.deleteAll();

        return "All Districts Deleted";

    }

}

**District custom function:**

This function retrieves all prisoners depending on the District ID

@GetMapping("getPrisonerByDistrict/{id}")

public StringBuilder getPrisonerByDistrict(@PathVariable Long id) {

   StringBuilder result = new StringBuilder();

   districtRep.findById(id).get().getCell().stream()

            .forEach(c -> result.append(c.getPrisoner().getName() + " " +

c.getPrisoner().getSurname() + " " c.getPrisoner().getAge() + " " + c.getPrisoner().getYearsOfImprisonment() + " " + c.getPrisoner().getCrimeType() + "<br>"));

   return result;

}

**Cell custom function:**

This function adds a cell to the Cell entity preassigned to a District ID

@PostMapping(value = "addCellWithDistrict/{id}", consumes = "application/json")

    public String addCellWithDistrict(@RequestBody Cell cell, @PathVariable Long id) {

        cell.setDistrict(districtRep.findById(id).get());

        cellRep.save(cell);

        return "Cell Added Along With Its Assigned District";

    }

This function adds a list of cells to the Cell entity preassigned to a District ID

@PostMapping(value = "addCellsWithDistrict/{id}", consumes = "application/json")

    public String addCellWithDistricts(@RequestBody List<Cell> cells, @PathVariable Long id) {

        for (Cell cell : cells) {

            cell.setDistrict(districtRep.findById(id).get());

        }

        cellRep.saveAll(cells);

        return "Cells Added Along With Its Assigned District";

    }

This function assigns a District ID to a cell in Cell entity

@GetMapping("assignDistrict/{cellId}/{disId}")

    public String assignDistrict(@PathVariable Long cellId, @PathVariable Long disId) {

        Cell cell = cellRep.findById(cellId).get();

        cell.setDistrict(districtRep.findById(disId).get());

        cellRep.save(cell);

        return "Cell Assigned To District";

    }

This function retrieves all cells based on District ID

@GetMapping("getCellsByDistrict/{id}")

    public List<Cell> getCellsByDistrict(@PathVariable Long id){

        List<Cell> result = cellRep.findByDistrictId(id);

        return result;

    }

**Prisoner customer functions:**

Add prisoner to Prisoner entity with a preassigned Labour ID

@PostMapping(value = "addPrisonerWithLabour/{id}", consumes = "application/json")

    public String addPrisonerWithLabour(@RequestBody Prisoner prisoner, @PathVariable Long id){

        if (prisoner.getLabour() == null) {

            prisoner.setLabour(new ArrayList<Labour>());

        }

        prisoner.getLabour().add(labourRep.findById(id).get());

        prisonerRep.save(prisoner);

        return "Prisoner Added With Their Corresponding Labour";

    }

Add prisoner to Prisoner entity with a preassigned Cell ID

    @PostMapping(value = "addPrisonerWithCell/{id}", consumes = "application/json")

    public String addPrisonerWithCell(@RequestBody Prisoner prisoner, @PathVariable Long id) {

        prisoner.setCell(CellRep.findById(id).get());

        prisonerRep.save(prisoner);

        return "Prisoner Added With Their Corresponding Cell";

    }

Add prisoner to Prisoner entity with a preassigned Cell ID and Labour ID

@PostMapping(value = "addPrisonerWithCellAndLabour/{cellId}/{labId}", consumes =

"application/json")

public String addPrisonerWithCellAndLabour(@RequestBody Prisoner prisoner, @PathVariable

Long cellId, PathVariable Long labId) {

        prisoner.setCell(CellRep.findById(cellId).get());

        if (prisoner.getLabour() == null) {

            prisoner.setLabour(new ArrayList<Labour>());

        }

        prisoner.getLabour().add(labourRep.findById(labId).get());

        prisonerRep.save(prisoner);

        return "Prisoner Added With Their Corresponding Cell And Labour";

    }

Add a list prisoner to Prisoner entity with a preassigned Labour ID

@PostMapping(value = "addPrisonersWithLabour/{id}", consumes = "application/json")

public String addPrisonersWithLabour(@RequestBody List<Prisoner> prisoner, @PathVariable

Long id) {

        for (Prisoner p : prisoner) {

            if (p.getLabour() == null) {

                p.setLabour(new ArrayList<Labour>());

            }

            p.getLabour().add(labourRep.findById(id).get());

        }

        prisonerRep.saveAll(prisoner);

        return "Prisoner Added With Their Corresponding Labour";

    }

Assign a Cell ID to prisoner

    @GetMapping("assignCell/{priId}/{cellId}")

    public String assignCell(@PathVariable Long priId, @PathVariable Long cellId) {

        Prisoner prisoner = prisonerRep.findById(priId).get();

        prisoner.setCell(CellRep.findById(cellId).get());

        prisonerRep.save(prisoner);

        return "Prisoner Assigned To Cell";

    }

Assign a Labour ID to prisoner

    @GetMapping("assignLabour/{priId}/{labId}")

    public String assignLabour(@PathVariable Long priId, @PathVariable Long labId) {

        Prisoner prisoner = prisonerRep.findById(priId).get();

        prisoner.getLabour().add(labourRep.findById(labId).get());

        prisonerRep.save(prisoner);

        return "Prisoner Assigned To Labour";

    }

Display all of the prisoners that are assigned to labour

@GetMapping("getPrisonersRelatedToLabour")

    public StringBuilder getPrisonersRelatedToLabour() {

        StringBuilder result = new StringBuilder();

        prisonerRep.findAll().stream().filter(p -> !p.getLabour().isEmpty())

.forEach(p -> result.append("Name: " + p.getName() + " " + p.getSurname() +

". Age:" + p.getAge() + ". Years of imprisonment: "

p.getYearsOfImprisonment() + ". Crime Type: " + p.getCrimeType() +

". <br>" + "Labour Type:" + p.getLabour().get(0).getLabourType() +

", " + p.getLabour().get(0).getLabourHours()

+ " Hours. <br>"));

        return result;

    }

Retrieves all prisoners based on Cell ID

    @GetMapping("getPrisonersByCell/{id}")

    public List<Prisoner> getPrisonersByCell(@PathVariable Long id) {

        List<Prisoner> result = prisonerRep.findByCellId(id);

        return result;

    }

Retrieves all prisoners based on Labour ID

    @GetMapping("getPrisonersByLabour/{id}")

    public List<Prisoner> getPrisonersByLabour(@PathVariable Long id) {

        List<Prisoner> result = prisonerRep.findByLabourId(id);

        return result;

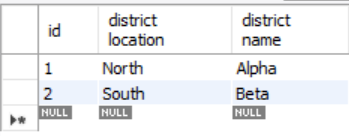
    }

**Labour has no custom functions.**

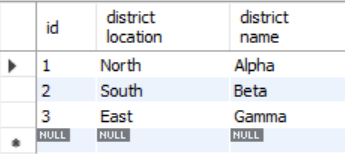
**Usage of CRUD and Custom Functions:**

**Create: “localhost:8088/district/addDistrict” || “localhost:8088/district/addDistricts”**

**Old:**

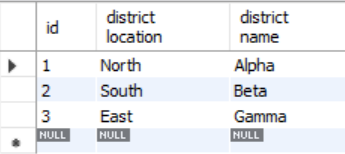
****

**New:**

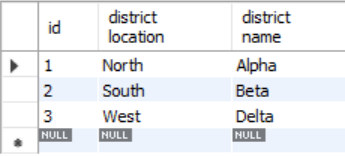
****

**Update: “localhost:8088/district/updateDistrict”**

**Old:**



**New:**

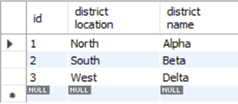
****

**Read: “localhost:8088/district” || “localhost:8088/district/{id}”**

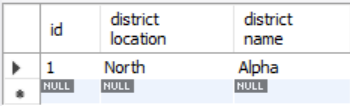
****

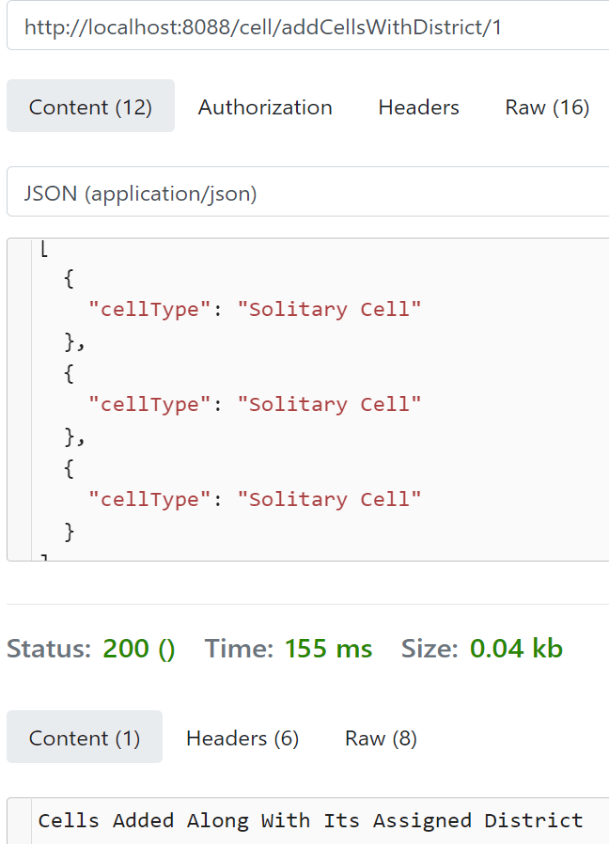
**Delete: “localhost:8088/district/deleteDistrict/{id}” || “localhost:8088/district/deleteAllDistricts”**

**Old:**



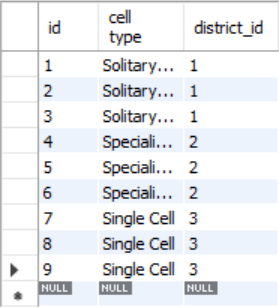
**New:**



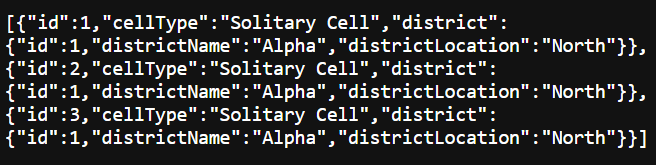
**Cell:**

in this cell custom function we are able to a list of cells into the database with a preassigned distirct to them.

**“http://localhost:8088/cell/addCellsWithDistrict/1”**

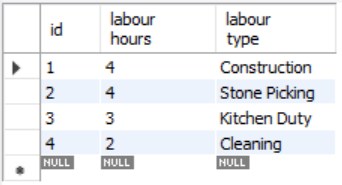


In this custom function we are able to view all of the cell depending on the distirct id.  
**“localhost:8088/cell/getCellsByDistrict/{id}”**



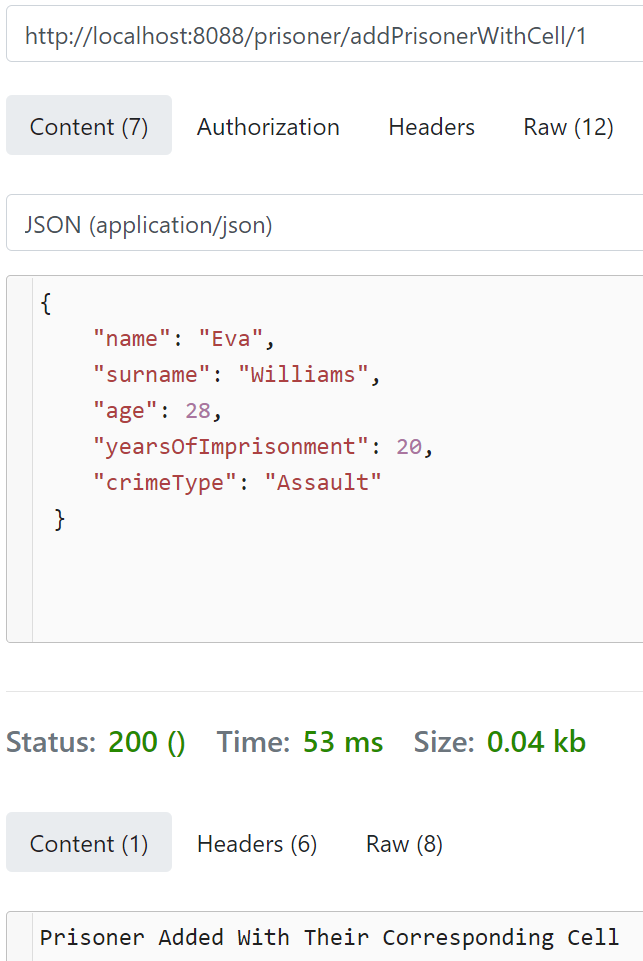
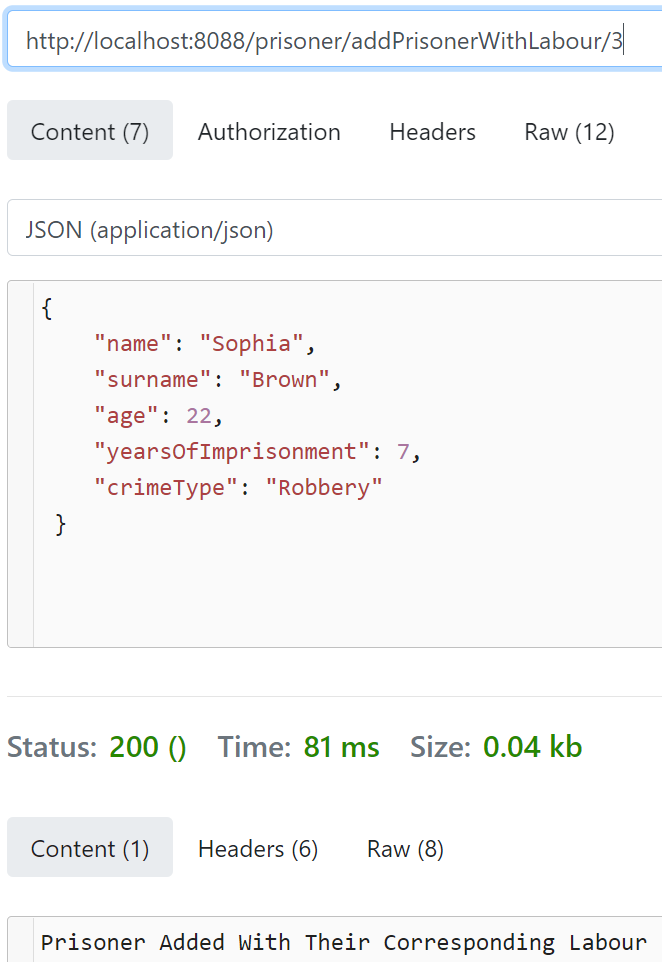
**Labour:**Adding a list of labour: **“localhost:8088/labour/addLabours”**

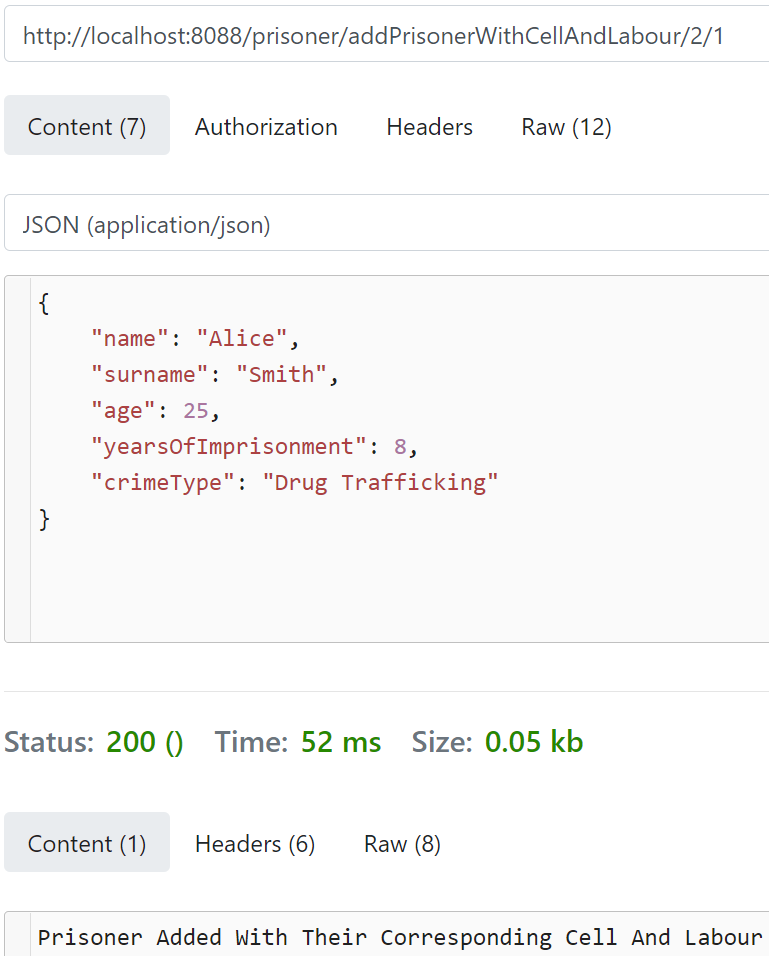


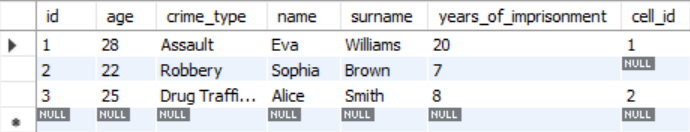


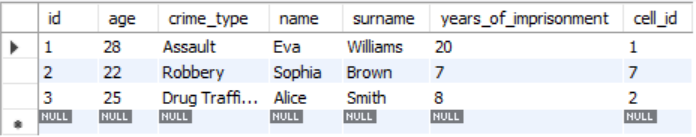
**Prisoner:**

Similar to the Cell Controller, in this Prisoner Controller we are able to add prisoner to a preassigned Labour or Cell or Labour & Cell at once. We are also able to assign multiple Prisoners to a Labour at once.

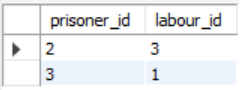


Assigning a cell to a prisoner: **“localhost:8088/prisoner/assignCell/{PrisonerID}/{CellID}”**

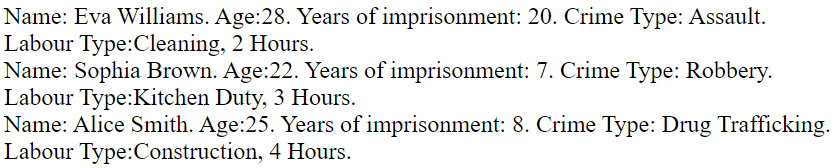
Before:  


After:  


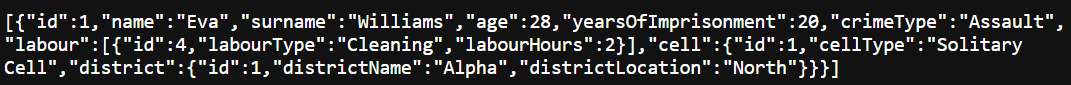
Assigning a labour to prisoner: **“localhost:8088/prisoner/assignLabour/{Prisoner}”**

Before:  


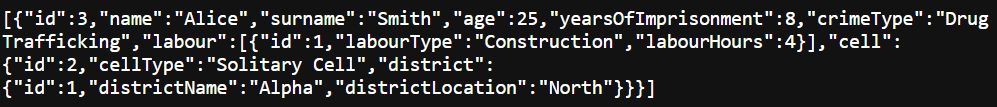
After:  


Displaying all of the Prisoners that have a relation with Labour: **“localhost:8088/prisoner/getPrisonersRelatedToLabour”**

Display prisoners by cell id: **“localhost:8088/prisoner/getPrisonersByCell/{id}”**



Display prisoners by labour id: **“localhost:8088/prisoner/getPrisonersByLabour/{id}”**



Thank you

Happy New Year