1. **Explanation of Models Folder in the Code**

In the Model code files, there are two main types of classes: DTO (Data Transfer Object) classes and Entity classes. Understanding these will help clarify how data is managed and transferred within the application.

* DTO (Data Transfer Object) Classes: DTO classes are used to transfer data between different parts of the application or between the application and a database. They often include attributes that represent the data but do not include complex logic or methods. The main purpose of DTOs is to encapsulate the data and ensure it is transferred efficiently and safely.
* Entity Classes: Entity classes extend DTO classes and typically include additional methods and logic that operate on the data. These classes are more interactive and are used within the application to manipulate and manage data, perform calculations, and provide utility functions.

1. **How They Work**

* DTO Classes: These are simple containers for data. They define the structure and types of data that will be used.
* Entity Classes: These extend DTOs, meaning they inherit the data structure from DTOs but also add more functionality. They include methods to:
  + Deserialize data from DTOs.
  + Convert the data to JSON format.
  + Provide additional utility methods specific to the data they manage.

1. **Entities and Their Summaries**

* Beds
  + DTO: `BedsDTO` includes properties like `id`, `name`, `location`, `gardenId`, `archived`, `userId`, and other related attributes.
  + Entity: `Beds` extends `BedsDTO` and includes methods for deserialization (`deserialize`), converting to JSON (`toJSON`), and displaying formatted date (`displayFinishDate`).
* Crops
  + DTO: `CropsDTO` includes properties like `id`, `name`, `image`, and `\_keywords`.
  + Entity: `Crops` extends `CropsDTO` and includes methods for deserialization (`deserialize`) and converting to JSON (`toJSON`).
* Gardens
  + DTO: `GardensDTO` includes properties like `id`, `userId`, `name`, `addressLine1`, `addressLine2`, `city`, `country`, `\_user`, `\_keywords`, `\_withPlague`, `archived`, and `description`.
  + Entity: `Gardens` extends `GardensDTO` and includes methods for deserialization (`deserialize`), converting to JSON (`toJSON`), and displaying the address (`displayAddress`).
* Harvest
  + - DTO: `HarvestDTO` includes properties like `id`, `date`, `quantity`, `unit`, `\_plantationObject`, `\_cropObject`, `\_bedObject`, and `\_gardenObject`.
  + - Entity: `Harvest` extends `HarvestDTO` and includes methods for deserialization (`deserialize`) and converting to JSON (`toJSON`).
* Image
  + - Interface: `IImage` defines properties like `id`, `url`, `permissions`, `image`, `product`, and `order`.
  + - Entity: `Image` includes methods for deserialization (`deserialize`), converting to JSON (`toJSON`), and getting properties (`getProperties`).
* Plagues
  + DTO: `PlaguesDTO` includes properties like `id`, `name`, and `\_keywords`.
  + Entity: `Plagues` extends `PlaguesDTO` and includes methods for deserialization (`deserialize`) and converting to JSON (`toJSON`).
* Plantations
  + DTO: `PlantationsDTO` includes properties like `id`, `cropId`, `bedId`, `order`, `startCultureDate`, `seed`, `numberOfCrop`, `behindGrowthDay`, `isDestroyed`, `isDestroyedPlague`, `lastUpdate`, `destroyedDate`, `harvestPlant`, `\_bedObject`, `\_gardenObject`, `\_userObject`, `\_cropObject`, `\_withPlague`, and `\_plagueObject`.
  + Entity: `Plantations` extends `PlantationsDTO` and includes methods for deserialization (`deserialize`), converting to JSON (`toJSON`), getting ID (`getId`), getting status (`getStatus`), and getting plague start date (`getPlagueStartDate`).
* Reports
  + DTO: `ReportsDTO` includes properties like `id`, `plagueId`, `image`, `plantationId`, `dateReport`, `isDestroyed`, `status`, `archived`, `comment`, `hasDoneActions`, `doneActions`, `gardenId`, `bedId`, `cropId`, `cropName`, `\_plagueObject`, and `\_userObject`.
  + Entity: `Reports` extends `ReportsDTO` and includes methods for deserialization (`deserialize`), converting to JSON (`toJSON`), getting report date (`getReportDate`), and getting status (`getStatus`).
* Tasks
  + DTO: `TasksDTO` includes properties like `id`, `cropsId`, `plagueId`, `cultureDay`, `taskName`, `taskDescription`, and `updateTask`.
  + Entity: `Tasks` extends `TasksDTO` and includes methods for deserialization (`deserialize`) and converting to JSON (`toJSON`).
* User
  + DTO: `UserDTO` includes properties like `name`, `userId`, `email`, `isAdmin`, `keywords`, `displayIsAdmin`, and `archived`.
  + Entity: `User` extends `UserDTO` and includes methods for deserialization (`deserialize`) and converting to JSON (`toJSON`).

**Summary**

In summary, the code is structured to have simple DTO classes for data representation and more complex Entity classes that extend these DTOs to include additional functionalities like deserialization, JSON conversion, and other utility methods. This separation of concerns ensures clean and efficient data handling, making the application easier to maintain and extend.