IRepositoryEntity

Complaint

- id: String
- title: String
- description: String
- cook: User
- client: User
- isArchived: boolean

+ create(...): Complaint
+ unarchiveAll(): void
+ getTitle(): String
+ getDescription(): String
+ getCook(): User
+ getClient(): User
+ isArchived(): boolean
+ archive(): void
+ getId(): String
+ serialise(): Map<String, Object>
+ deserialise(map:Map<String,Object>): void

0..* complaining client
0..* reported cook

A User will not have a role if admin is true.

IRepositoryEntity

User

- firstName:String
- lastName:String
- email:String
- passwordHash:String
- address:String
- admin:boolean
- role:UserRole

+ createNewUser(...): User
+ getUserByEmail(email:String): User
+ getters
+ getPasswordHash(): String
+ getId(): String
+ serialise(): Map<String, Object>
+ deserialise(map:Map<String,Object>): void

Maintains global state and saves logged in User across sections

<<singleton>>
MealerSystem (extends Application)

- system: MealerSystem
- usersession: SharedPreferences
- currentUser: User
- repository: IRepository

+ getSystem(): MealerSystem
+ getCurrentUser(): User
+ getRepository(): IRepository
+ onCreate(): void
+ restoreSession(): void
+ logoff(): void
+ tryLogin(email:String, password:String): boolean
- login(user:User): void
~ hashPassword(password:String): String

current user

0..1

1

repository instance

1

Generalization of a queryable permanent storage system

<<interface>>
IRepository

(T extends IRepositoryEntity)

+ hasId(cls:Class<T>, id:String): boolean
+ getById(cls:Class<T>, id:String): T
+ set(cls:Class<T>, entity:T): void
+ update(cls:Class<T>, id:String, properties:Map<String,Object>): void
+ delete(cls:Class<T>, entity:T): void
+ list(cls:Class<T>): List<T>
+ query(cls:Class<T>, predicate:Predicate<T>): List<T>
+ getAutoID(cls:Class<T>): String

1 cook

1 user

ISerialisableEntity

role 0..1

IRepositoryEntity

Meal

- id: String
- name: String
- type: String
- cuisine: String
- ingredients: String
- allergens: String
- description: String
- price: int
- cook: User
- isOffered: boolean

+ createMeal(...): Meal
+ updateMeal(...): Meal
+ getters
+ flipIsOffered(): void
+ getId(): String
+ serialise(): Map<String, Object>
+ deserialise(map:Map<String,Object>): void

0..*

<>
UserRole

~ user: User

+ getUser(): User

<<interface>>
ISerialisableEntity

+ serialise(): Map<String, Object>
+ deserialise(map:Map<String,Object>): void

<<interface>>
IRepositoryEntity

~ tablesNames: Map<Class<? extends IDatabaseEntity>,String>

+ getTableName(cls:Class<? extends IDatabaseEntity>): String
~ getId(): String

Represents a Entity that can be stored in a table/document in permanent storage
Can store other ISerialisableEntity and is responsible for calling their serialise and deserialise methods.

CloudFirestoreRepository

- db:FirestoreFirebase

(T extends IRepositoryEntity)

+ hasId(cls:Class<T>, id:String): boolean
+ getById(cls:Class<T>, id:String): T
+ set(cls:Class<T>, entity:T): void
+ update(cls:Class<T>, id:String, properties:Map<String,Object>): void
+ delete(cls:Class<T>, entity:T): void
+ list(cls:Class<T>): List<T>
+ query(cls:Class<T>, predicate:Predicate<T>): List<T>
+ getAutoID(cls:Class<T>): String

ClientRole

- cardNumber:String

+ getCardNumber(): String
+ setCardNumber(num:String): void
+ serialise(): Map<String, Object>
+ deserialise(map:Map<String,Object>): void

CookRole

- description:String
- banExpiration:long
- banReason:String
- totalRatings:long
- numRatings:long
- mealsSold:long

+ getDescription(): String
+ getBanExpiration(): long
+ getBanReason(): String
+ getAverageRating(): float
+ getMealsSold(): long
+ setRole(desc:String,exp:long,reason:String): void
+ setDescription(desc:String): void
+ setBan(exp:long,reason:String): void
+ rate(rating:int): void
+ soldMeal(): void
+ resetRatingsAndMealsSold(): void
+ serialise(): Map<String, Object>
+ deserialise(map:Map<String,Object>): void

A ban is represented by banExpiration being a non-negative number.
=-1 -> not banned
=0 -> permanent ban;
>0 -> Unix millisecond timestamp for expiration;