Design Patterns:

Singleton:

We are going to use singleton design pattern when dealing with businesses. So once a business is registered, we are going to initialize its object and each time a customer or admin wants to access this business we are going to return the same object that we just created. For instance, if a customer books a business, we are going to modify the business object data (for example the map consisting of free-dates) and we are not going to re-create a new business object each time a customer makes a booking.

Adapter Pattern:

We will use adapter patter in order to get data in the correct format as database accepts. So, we will use adapter pattern in order to produce a 32-digit encrypted password. So, when a new user is being registered, he provides a password that should have a length of 8 minimum, should have an uppercase letter and a number. When user clicks submit, we will call the method found in adapter object, give as parameter the given password and receive as return the encrypted password. Then we will call the function that takes as parameter all user details (name, surname, phone Nr and encrypted password which we got from adapter object) and will save the given information in the database.

Flyweight Design Pattern:

There are some data that will be part of each actor in our system. So, costumers, businesses and planners will have their personal information such as name, surname, email, phone_nr and address. Instead of giving these data as attributes in their classes we can use Flyweight pattern by creating a class called 'Particle' and giving all these data as class members. Now in each of classes mentioned above (costumers, businesses and planners) we can add a data type of 'Particle' and a method called addParticle(). So except from detailed information that each of actors will have, they will be able to get general information by calling the method addParticle(). So we avoid redundancy of writing the same data types in almost 3 or 4 different classes.