Memo

Managing user actions in the project:

Decisions made in architecture design:

1. The code should respect the SOLID, KISS, DRY and YAGNI principles
   1. **Single responsibility principle**: every class has only one responsibility well defined (eg: UserRepository manages only crud operations on the db tables that contain or are related to User objects)
   2. **Open closed principle**: Any other operation needed to be done to the User object (not crud operations) happens in UserService, which extends UserRepository
   3. **Interface segregation principle**: ViewModels implement different interfaces for each operation (eg: ILogInViewModel and IRegisterViewModel)
   4. **Dependency inversion principle**: All the models used are stored in another project in order to be accessible for all projects (for WPF app and Web app)
   5. **Keep it stupid simple**: While planning the application the tasks were broke down to the simplest, avoiding usage of libraries that required a lot of setup (making the code unreadable or hard to understand),
   6. **Don’t repeat yourself**: While dev the project, all the code that could be used in multiple places was put into a method in the according place.
   7. **You aren’t gonna need it**: avoiding over engineering in all possible situation: while implementing the project functionalities were developed only if needed, not keeping in mind future possible functionalities.
2. Naming conventions: respecting the c# naming convention: using PascalCasing for class, methods and public fields in a class; using camelCase for parameters, local variables in a method and private fields in a class
3. Corner cases:
   1. Exception when the API server it is not connected to the internet or does not respond
   2. The user tries to register with an email already used.
   3. The user does not provide the correct combination of username/password
   4. The username on which the user tries to log in does not exist

Exception safety level

*Class UserRepository*:

1. public User Add (User user)
   1. parameter: An User object that will be added to db
   2. return: the user added to the db with de id binded
   3. exception safety level: Strong guarantee (the SaveChanges is the only one that can throw an exception and if it fails the operation will rollback)
2. public IList<User> FindAll ()
   1. parameter:
   2. return: A List of users
   3. exception safety level: Strong guarantee: there is no data affected by this code
3. public IList<User> FindAllAdmins ()
   1. parameter:
   2. return: A List of users
   3. exception safety level: Strong guarantee: there is no data affected by this code
4. public User FindByEmail (string email)
   1. parameter: a string containing the email
   2. return: The user with the matching email // null if the user is not found
   3. exception safety level: Strong guarantee: there is no data affected by this code
5. public User UpdateIsEmailVerified (User userToUpdate)
   1. parameter: An User object that has not verified his account yet
   2. return: The user with the field “IsEmailVerified” set to true
   3. exception safety level: Strong guarantee (the SaveChanges is the equivalent of “commit transaction” and it happends at the end of the method. If anything fails before that the changes will not be saved))
6. public User UpdateActivationCode (User updatedUser)
   1. parameter: An User object that will be updated
   2. return: the user updated
   3. exception safety level: Strong guarantee (the SaveChanges is the equivalent of “commit transaction” and it happends at the end of the method. If anything fails before that the changes will not be saved)

*Class TripRepository*:

1. public Trip Add (Trip trip)
   1. parameter: A Trip object that will be added to db
   2. return: the trip added to the db with de id binded
   3. exception safety level: Strong guarantee (the SaveChanges is the equivalent of “commit transaction” and it happends at the end of the method. If anything fails before that the changes will not be saved)
2. public void Remove (Trip trip)
   1. parameter: A Trip object that will be removed from db
   2. return:
   3. exception safety level: Strong guarantee (the SaveChanges is the equivalent of “commit transaction” and it happends at the end of the method. If anything fails before that the changes will not be saved)
3. public IList<Trip> GetAll()
   1. parameter:
   2. return: A list of trips
   3. exception safety level: Strong guarantee: there is no data affected by this code
4. public Trip Update (Trip updatedTrip)
   1. parameter: A trip
   2. return: the trip updated
   3. exception safety level: Strong guarantee (the SaveChanges is the equivalent of “commit transaction” and it happends at the end of the method. If anything fails before that the changes will not be saved)