

The background is a dark navy blue. On the left, there are two overlapping parallelogram shapes, one blue and one light green, both tilted at an angle. In the bottom left, there is a circular inset showing a detailed, grayscale image of a printed circuit board (PCB) with various electronic components. In the top right corner, there is a faint, grayscale image of a complex circuit board layout with many traces.

# Draw.IO

## Code case study

by K.Kostenkov for Voodoo

# Table of contents

[Profile save\load](#)

[Login](#)

[Server data check](#)

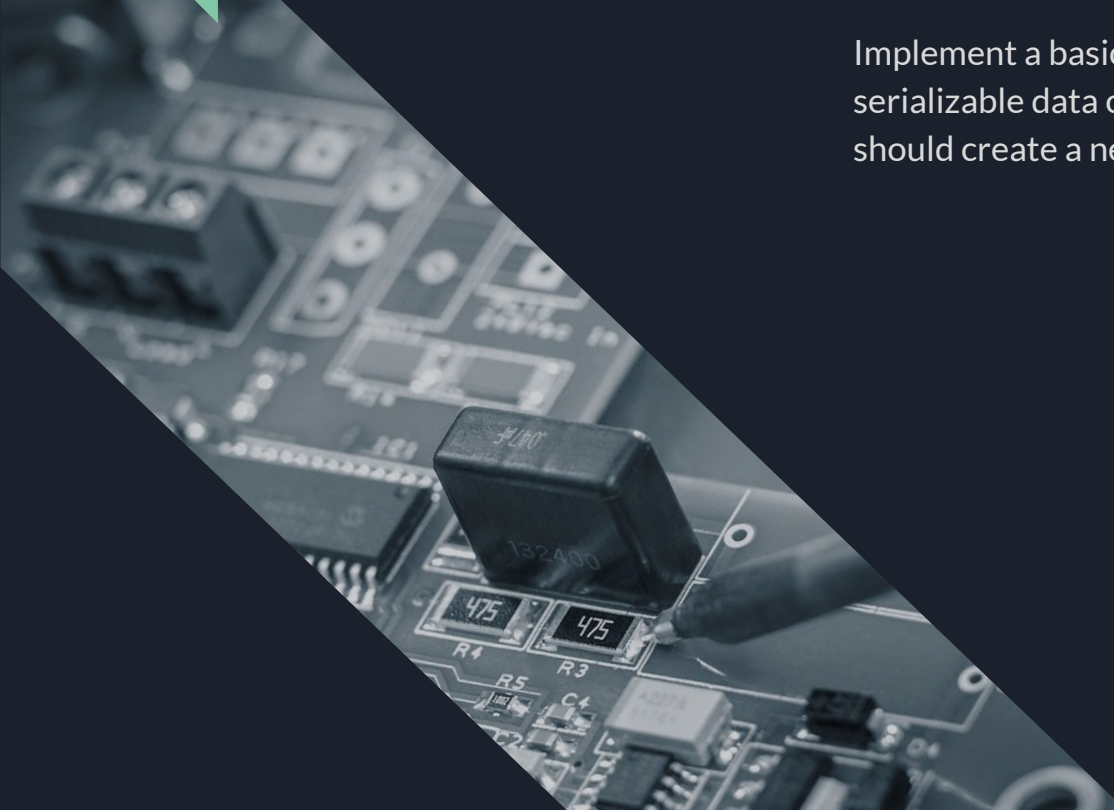




# Profile save\load

Task description:

Implement a basic init loop to save in a separate serializable data class the save of the player. The game should create a new player when no save exists





# Profile save\load

Implementation comments:

Entry point is `ProfileService.cs`

Game supports two types of saves: Immediate (data update is important, worth saving right away, like IAP, rewards) and deferred that will happen some time in the future not to overload the I/O (view settings)

`ProfileService` is expected to be used by the interfaces that serve only needed data to consumers (interface segregation principle)

Matching the project's code style was considered unnecessary since there is no code style.

Due to poor project lifetime management and missing DI an attribute `[DefaultExecutionOrder(-9)]` was added to `GameManager` and synchronous Profile data loading method is used. Implementing an object that initializes managers and views in a predefined order or Adding a bootstrap scene will allow getting rid of this solution.

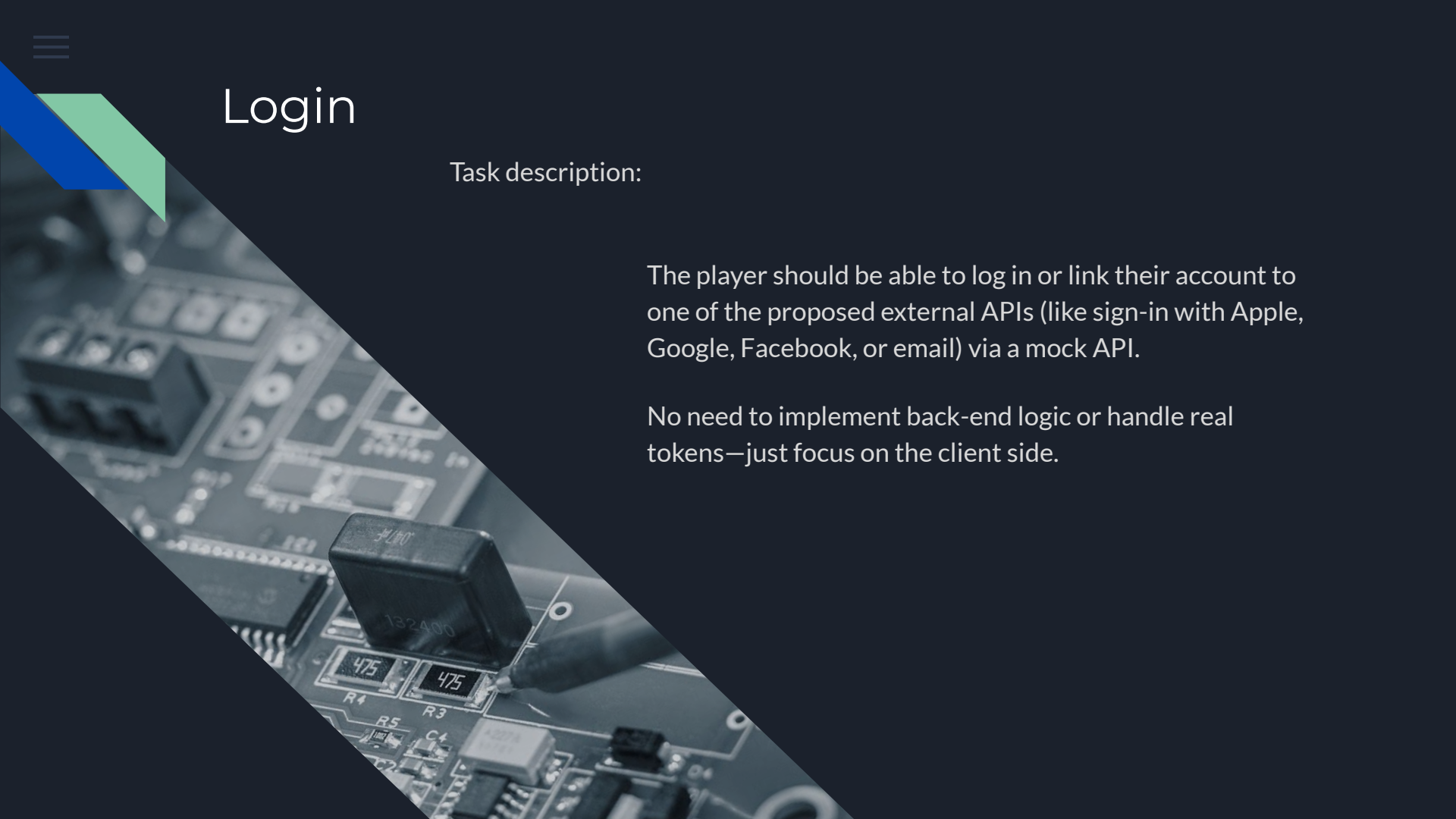


# Login

Task description:

The player should be able to log in or link their account to one of the proposed external APIs (like sign-in with Apple, Google, Facebook, or email) via a mock API.

No need to implement back-end logic or handle real tokens—just focus on the client side.





# Login

## Implementation comments:

Entry point is `LoginService.cs`

To imitate API calls delay `CoroutineRunner` class was added

The difference of behaviour between social providers could be implemented in their respective classes. Implementation of Apple and email was considered a waste of time and insignificant to show my kills.

The solution does not

- cover multiple active login providers
- cover case of failed `ExpressLogin`

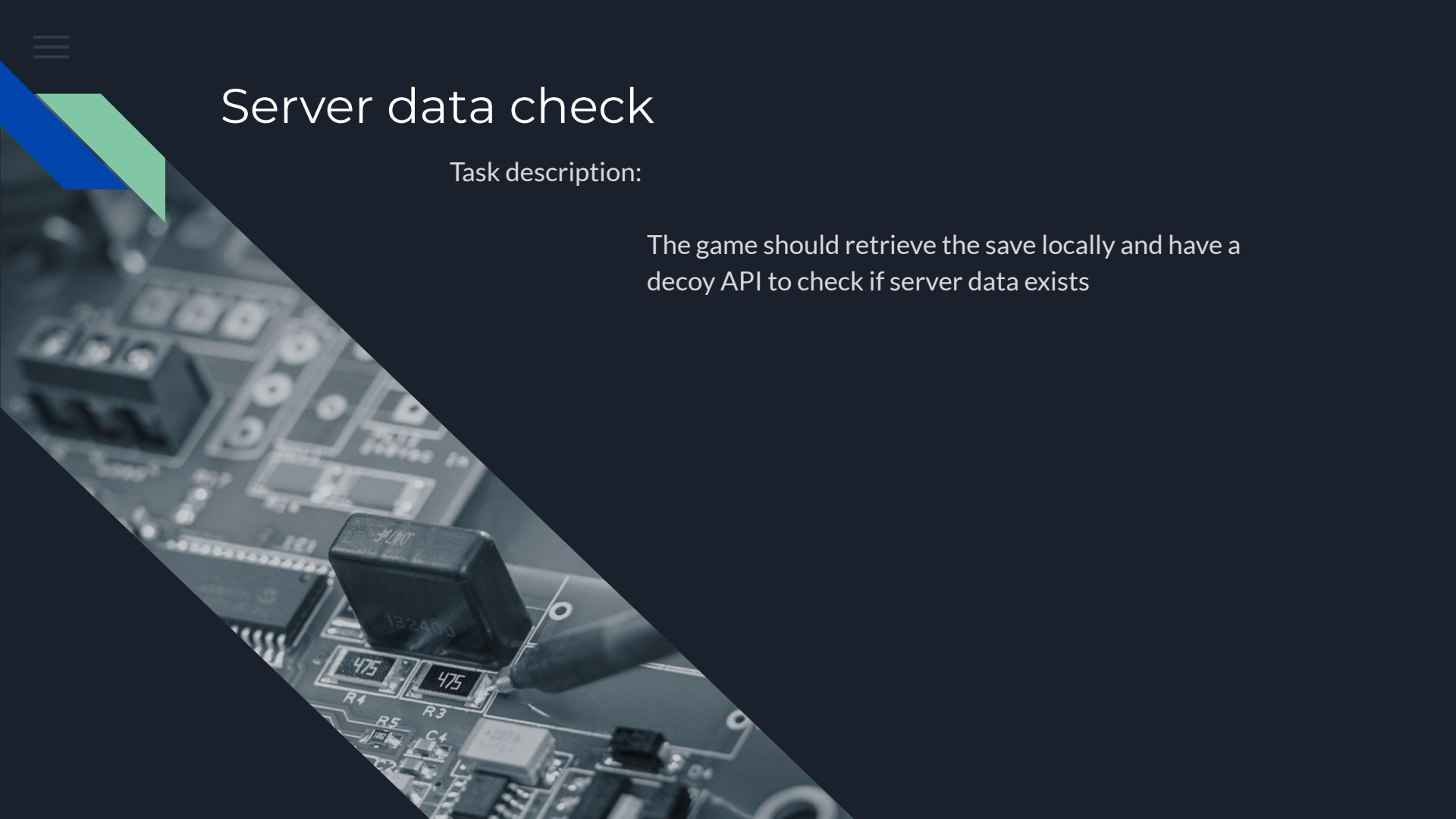
Social data storing is done via `PlayerPrefs`, but could be easily encapsulated.



# Server data check

Task description:

The game should retrieve the save locally and have a decoy API to check if server data exists





# Server data check

Implementation comments:

Entry point is `RemoteProfileFetcher`

`ProfileSaveDetails` class is for determining the date of the save and the necessity of json data migrations. It is also expected to be used next to the regular local `ProfileData` saves