# Assignment Instructions

The contract you are working on is a Provider-Subscriber system. It models a marketplace where entities, referred to as "Providers", can offer some services for a monthly fee. These services are consumed by entities known as "Subscribers". The Providers and Subscribers interact with each other using a specific ERC20 token as the medium of payment. Both Providers and Subscribers have their balances maintained within the contract.

The contract maintains a registry of Providers and Subscribers, identified by their respective IDs. Each Provider has its list of Subscribers. A Subscribed should use a certain number of Providers. Additionally, a Provider can be in one of two states: active or inactive, depending on whether it can provide services. A subscription can be paused, so the Subscriber is not charged by Providers.

## Key Functionalities

**Provider Registration**: Providers can register by specifying a registration key and a fee. The system prevents a Provider from registering using the same key more than once. There is also a maximum limit on the number of Providers that can be registered (200). The system should check that the minimum fee amount is worth at least $50 based on the current token price from Chainlink.

**Provider Removal**: Providers can be removed from the system, but only by their respective owners. The balance held in the contract is returned to the owner upon removal.

**Subscriber Registration**: Subscribers can register with one or more active Providers. The system should check that the deposited amount is worth at least $100 based on the current token price from Chainlink.

**Increase the subscription deposit**: Subscribers can increase the balance of subscriptions by transferring funds to the contract.

**Withdraw Provider Earnings**: Providers can withdraw their earnings from the contract, which are calculated based on their subscriber count and the fees they charge. The calculation is made every month. When withdrawing, emit an event that includes both the token amount withdrawn and its USD equivalent.

**Update Provider State**: The state of the Providers (active or inactive) can be updated. Only the contract owner can call this function.

**View functions**: Read-only functions are a key part of this system. Implement these:

- Get the state of a provider by id: returns number of subscribers, fee, owner, balance, and state.
- Get the provider earnings by id.
- Get the state of a subscriber by id: owner, balance, plan, and state.
- Implement a function getSubscriberDepositValueUSD(uint256 subscriberId) that returns the current USD value of a subscriber's deposit based on the latest Chainlink price data.

## Technical requirements

**Upgradeability**: Allow the contract to be upgradeable, with the possibility of making it non-upgradeable in the future.

**Authorization**: Consider the use of auth mechanisms if needed.

This should give a general idea of what your contract is designed to accomplish. To complete the system, you'll need to make sure to follow the best practices of smart contract development, especially concerning **security** and **gas efficiency**.

In the process of refining this contract, you're encouraged to adjust the data structures and types as necessary to optimize efficiency. However, ensure you're still adhering to the contract's core functionality.

**Don't get blocked by the completion of a feature** after expending reasonable effort. Please comment/document your solution to present it later and continue.

Don’t hesitate to reach out to marco@ssvlabs.io or phoebe@ssvlabs.io for any question or to extend the time.

## Bonus Section

Here are some additional questions that delve deeper into the contract's functionality and its potential improvements:

**Balance Management**: Currently, the contract operates monthly, meaning that subscribers need to deposit at least two months' worth of fees when they register. Could this process be improved or made more precise? Consider whether allowing subscribers to pay for services on a daily or even hourly basis would be more efficient. How could such a feature be implemented?
How would you modify the system to allow subscribers to deposit a USD-pegged amount rather than a specific token amount? What challenges might this introduce?

**System Scalability**: The current system restricts the maximum number of providers to 200. How could this system be changed to become more scalable and remove such a limitation? Are there changes to the data structures or other modifications that would allow the system to handle a theoretically unlimited number of providers?

**Changing Provider Fees**: Currently, providers set their fees upon registration. What if a provider needs to change their fee after registration? How can the system ensure that the correct amount is charged to subscribers, mainly if the fee change occurs partway through a billing cycle? Consider how such a feature could be implemented while maintaining fairness for both providers and subscribers.

Once ready, please upload the updated code to your GitHub account.