```solidity
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/utils/Context.sol";

contract TokenMarket is Context {
// Token addresses
address public houseToken;
address public gardenToken;
address public mortgageToken;
address public loanToken;
address public greenAssetToken;
address public investmentToken;
address public insuranceToken;

// Mapping to keep track of token prices
mapping(address => uint256) public tokenPrices;

// Event for token purchase
event TokenPurchased(address buyer, address token, uint256 amount);

// Modifier to ensure only token owners can sell their tokens
modifier onlyTokenOwner(address token, uint256 tokenId) {
require(
IERC721(token).ownerOf(tokenId) == _msgSender(),
"TokenMarket: Only token owner can sell"
);
_;
}

constructor(
address _houseToken,
address _gardenToken,
address _mortgageToken,
address _loanToken,
address _greenAssetToken,
address _investmentToken,
address _insuranceToken
) {
houseToken = _houseToken;
gardenToken = _gardenToken;
mortgageToken = _mortgageToken;
loanToken = _loanToken;
greenAssetToken = _greenAssetToken;
investmentToken = _investmentToken;
insuranceToken = _insuranceToken;
}

// Function to set the price for a token
function setTokenPrice(address token, uint256 price) external {
require(price > 0, "TokenMarket: Price must be greater than zero");
tokenPrices[token] = price;
}

// Function to buy a token
function buyToken(address token, uint256 amount) external payable {
uint256 totalCost = tokenPrices[token] * amount;
require(
msg.value >= totalCost,
"TokenMarket: Insufficient payment for token"
);
require(
IERC20(token).balanceOf(address(this)) >= amount,
"TokenMarket: Insufficient token balance"
);

IERC20(token).transfer(_msgSender(), amount);

emit TokenPurchased(_msgSender(), token, amount);
}

// Function to sell a token
function sellToken(address token, uint256 amount) external {
require(amount > 0, "TokenMarket: Amount must be greater than zero");
require(
IERC20(token).allowance(_msgSender(), address(this)) >= amount,
"TokenMarket: Token allowance not set"
);

IERC20(token).transferFrom(_msgSender(), address(this), amount);

payable(_msgSender()).transfer(tokenPrices[token] * amount);

emit TokenPurchased(_msgSender(), token, amount);
}
}
```

This contract allows token owners to set prices for their tokens, buyers can purchase tokens using the `buyToken` function, and sellers can sell tokens using the `sellToken` function. The contract also emits an event `TokenPurchased` whenever a token is bought or sold.

Please note that this is a basic implementation, and further checks and conditions can be added according to your specific requirements. Remember to deploy this contract on the RSK testnet and provide the correct addresses of the tokens you want to sell and purchase.