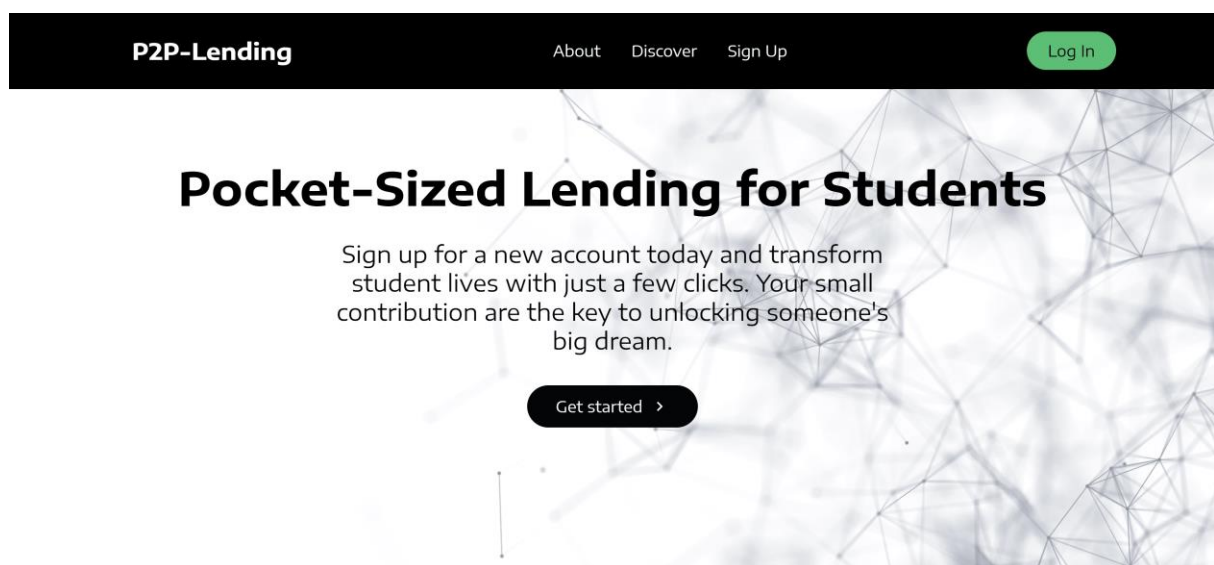


# Marketplace Dynamics

The infrastructure for the peer-to-peer lending page is available on our [GitHub](#) repository. The installation instructions can be seen in the [README](#) file of the web platform. The [smart contract](#) can be accessed as well as the corresponding [README](#) file.

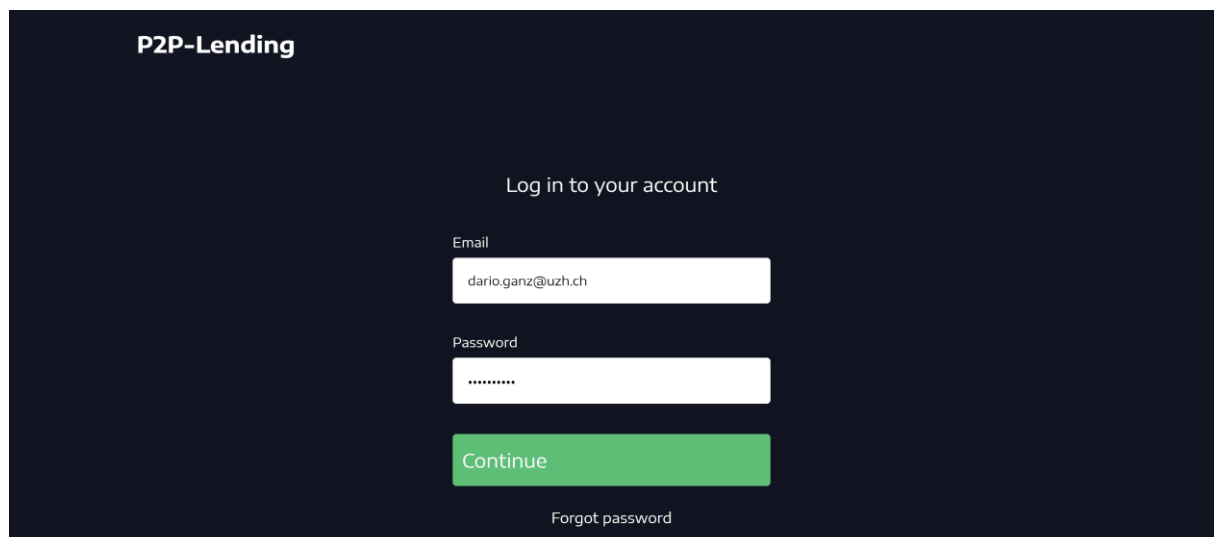
## 1. Landing Page

The P2P Lending Platform's landing page offers two options for accessing the marketplace: either by clicking the «Log In» button or by creating an account through the «Get started» button. Importantly, only registered users are able to access the marketplace and the wallet address of a user is automatically registered in the smart contract.



## 2. Login Page

In this case, I already have a registered account. I simply log in and click «Continue» to arrive at the marketplace.



### 3. Lending Form

As a student or someone in need of a credit, I can fill in the form on the marketplace and submit an enquiry for a loan. To do so, I simply fill out the form with the required information, including the amount, the period until you are able to pay back the loan, your personal wallet address, a precise description of the loan usage, and documents to prove your creditworthiness. In this example, we entered 10 USD because on the test net, we need to receive the equivalent amount of stablecoins from a faucet. On faucets there is a limit to the amount that can be received. Once you have inputted all of this information, click «Add Request» to successfully create a loan request.

**P2P-Lending**

## Lenders Marketplace

Amount (in USD) \*

10

Please enter your desired amount

Period (in days) \*

30

Please enter your desired period until the repayment

Wallet Address \*

0x7591Fd200F9cc71c2c4f0655f15d00B1f4520C91

Please enter your wallet address

Description of loan usage \*

Credit to pay the rent

Please enter a detailed description about the loan usage

UPLOAD FILES

**Selected files:**

- Proof.pdf

ADD REQUEST

## 4. List of Lending Requests

The lending request will be displayed at the bottom of the page in the «Current Lending Requests» List. Once the backend is successfully updated for user data storage, it will be connected to our frontend. Consequently, the user information, such as the name and profile picture, will be directly displayed in the list. If a lender presses the «Grant Credit» button, the MetaMask inject provider pops up and approves the USDC stablecoin contract to transfer the specified amount of tokens to the borrower.

### Current Lending Requests

<b>Amount:</b> \$9	<b>Period:</b> 30days	<b>Description:</b> Credit to pay the rent
<b>Files:</b> <ul style="list-style-type: none"><li>• Proof.pdf</li></ul>		
<button>GRANT CREDIT</button>		

#### About us

Blockchain  
Testimonials  
Investors  
Terms of Services

#### Contact Us

Contact  
Support  
Destinations  
Sponsorship

#### Videos

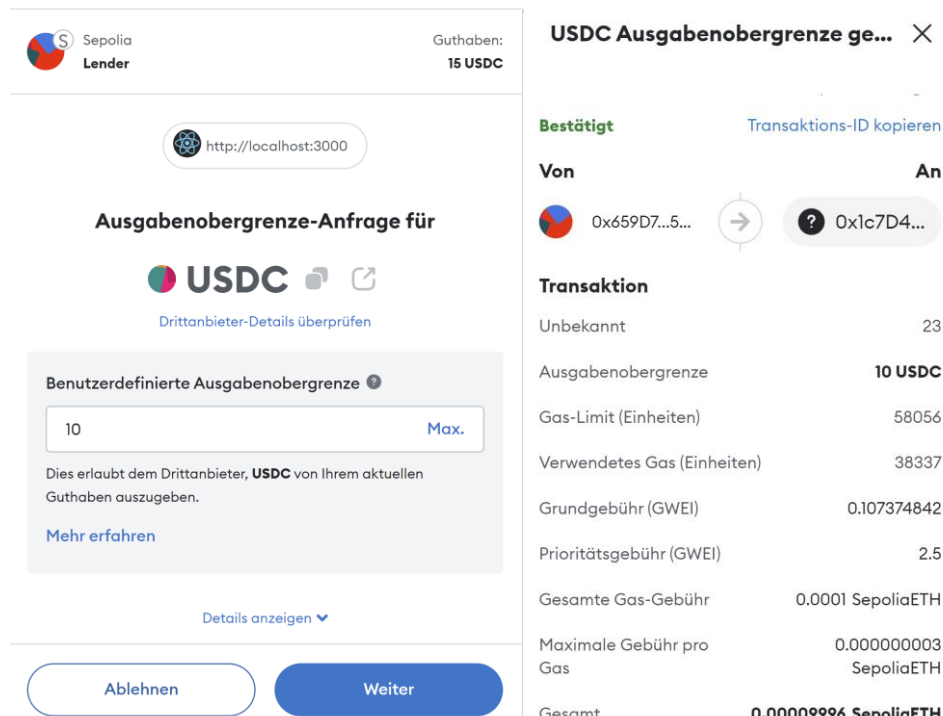
Submit Video  
Ambassadors  
Agency  
Influencer

#### Social Media

Instagram  
Facebook  
Youtube  
Twitter

## 5. Approval for the Token Transfer

When the MetaMask window opens up I can set the spending limit for the stablecoin contract. In the case of our prototype we use the Sepolia test net on Ethereum and the USDC contract (address: [0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238](#)) on that test net. After pressing «Weiter» the spending limit is set. By viewing the confirmation you can see that the spending limit is set for that exact USDC contract address.



**USDC Ausgabenobergrenze ge...** ✕

**Bestätigt** [Transaktions-ID kopieren](#)

**Von** 0x659D7...5... **An** 0x1c7D4...

**Transaktion**

Unbekannt	23
Ausgabenobergrenze	<b>10 USDC</b>
Gas-Limit (Einheiten)	58056
Verwendetes Gas (Einheiten)	38337
Grundgebühr (GWEI)	0.107374842
Prioritätsgebühr (GWEI)	2.5
Gesamte Gas-Gebühr	0.0001 SepoliaETH
Maximale Gebühr pro Gas	0.000000003 SepoliaETH
<b>Gesamt</b>	<b>0.00009996 SepoliaETH</b>

## 6. Approval: Front End

### 6.1 Imports and Initialization

- Web3 is required to interact with the Ethereum blockchain. Web3 is a collection of libraries that allow you to perform actions like interacting with smart contracts or transfer tokens
- The Application Binary Interface (ABI) is a JSON representation that tells the code how to format calls to a smart contract and how to decode data coming back.

```
import Web3 from 'web3';
import SmartContractABI from './ABIs/SmartContractABI.json'
import StableCoinABI from './ABIs/StablecoinABI.json'

const p2pLendingContractAddress = '0xc4382b37B596b7bD2B2C20F34992aF9fbf3B8625';
const stableCoinAddress = '0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238';
```

## 6.2 useEffect Hook for Web3 Initialization

The «useEffect» hook initializes Web3, fetches the user account, and sets up the smart contracts with their ABI's and contract addresses.

```
useEffect(() => {
  const initWeb3 = async () => {
    if (window.ethereum) {
      try {
        // Request account access
        await window.ethereum.request({ method: 'eth_requestAccounts' });
        const web3 = new Web3(window.ethereum);

        // Fetch the list of accounts
        const accounts = await web3.eth.getAccounts();
        if (accounts.length > 0) {
          setUserAccount(accounts[0]); // Sets the first account as the user account
        } else {
          console.log("No accounts found.");
        }

        // Initialize contract instances
        const tempStableCoinContract = new web3.eth.Contract(
          StableCoinABI,
          stableCoinAddress
        );

        const tempP2PLendingContract = new web3.eth.Contract(
          SmartContractABI,
          p2pLendingContractAddress
        );

        setStableCoinContract(tempStableCoinContract);
        setP2PLendingContract(tempP2PLendingContract);
      } catch (error) {
        console.error("Error connecting to MetaMask", error);
      }
    } else {
      console.log('Ethereum provider not found. Install MetaMask.');
```

### 6.3 Function for Approving the Token Transfer

The function «approveCredit» manages the approval of the token transfer by calling the «approve» method of the ERC20 stablecoin contract. According to the ERC20 token standard, a contract cannot move tokens on your behalf until you explicitly approve it. This security feature ensures that your tokens cannot be transferred without your permission.

```
const approveCredit = async (borrowerAddress, amount, period) => {
  if (!stableCoinContract || !userAccount) {
    console.log("In approveCredit - Period:", period);
    return;
  }

  const amountInTokens = amount.toString();
  const amountInWei = (amountInTokens * Math.pow(10, 6)).toString(); // For a token with
  6 decimals
  console.log(amountInWei);

  try {
    console.log(`Attempting to approve ${amount} tokens for the P2P lending
    contract...`);
    await stableCoinContract.methods.approve(p2pLendingContractAddress,
    amountInWei).send({ from: userAccount });

    // Navigate only after successful approval
    console.log(`Navigating with borrowerAddress: ${borrowerAddress}, amount: ${amount},
    period: ${period}`);
    navigate(`/Marketplace/grant-credit/${borrowerAddress}/${amount}/${period}`);
    console.log("Approval successful.");
  } catch (error) {
    console.error("Approval error:", error);
  }
};
```

### 6.4 Integration in the User Interface

The approval function is then passed down to the «LendingRequestList» component.

```
<LendingRequestsList requests={lendingRequests} onApproveCredit={approveCredit}/>
```

When the «Grant Credit» button is pressed in the «LendingRequestList», the approval function is called with the borrower's address, the requested amount and the period as the parameters.

```
<Button
  onClick={() => onApproveCredit(request.borrowerAddress, request.amount, request.period)}
>
  Grant Credit
</Button>
```

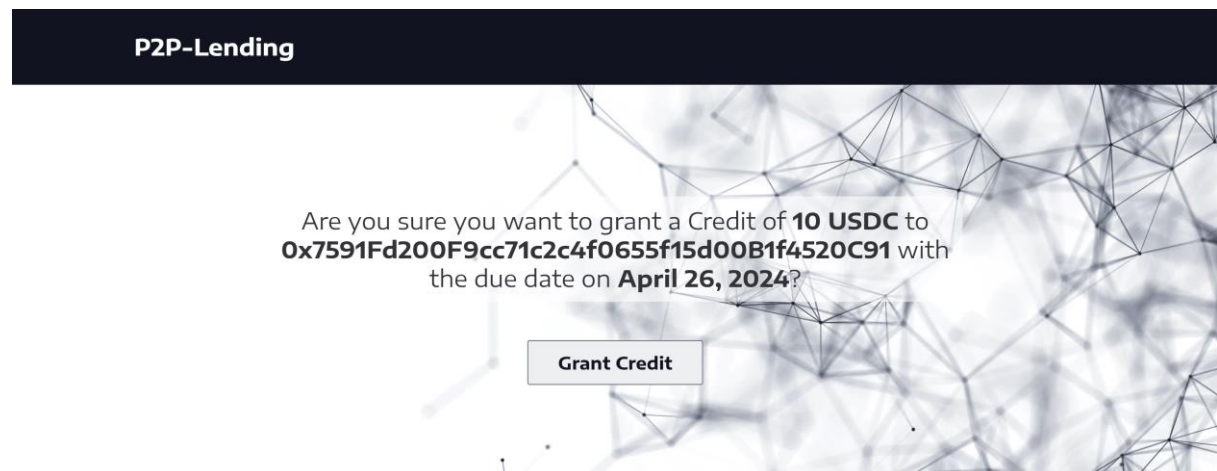
## 7. Approval: Back End

The approve function of the USDC contract enables token holders to authorize another address to spend a specific amount of their tokens. This mechanism is foundational for enabling secure delegated token transactions in decentralized applications.

```
function _approve(  
    address owner,  
    address spender,  
    uint256 value  
) internal override {  
    require(owner != address(0), "ERC20: approve from the zero address");  
    require(spender != address(0), "ERC20: approve to the zero address");  
    allowed[owner][spender] = value;  
    emit Approval(owner, spender, value);  
}
```

## 8. Grant Credit

After a successful approval, the web platform navigates to the confirmation page, which displays the loan details once again. The due date is calculated by adding the number of days to today's date. When the second «Grant Credit» button is pressed, the front end accesses the «P2PLending» smart contract as the back end. The function that transfers the tokens is then triggered.



## 9. Grant Credit: Front End

The imports and initialization as well as the «useEffect» hook work similar on this page of the web platform. The «transferTokens» function is the core of the lending operation. Initially, it converts the loan amount to the appropriate unit based on the token's decimals. The «dueDateTimestamp» is then calculated as the timestamp on the blockchain indicating the latest deadline for the loan repayment. Finally, the function calls the «grantCredit» method of our smart contract to transfer the specific amount of tokens from the lender to the borrower.

```
const transferTokens = async () => {
  if (!p2pLendingContract || !userAccount) {
    console.error("Contract not initialized or user account not found.");
    return;
  }

  const amountInTokens = amount.toString();
  const amountInWei = (amountInTokens * Math.pow(10, 6)).toString(); // USDC has 6
  decimals
  console.log(`Amount in Wei: ${amountInWei}`);

  const numberOfDaysToAdd = parseInt(period, 10);
  if (isNaN(numberOfDaysToAdd)) {
    console.error('Invalid period value:', period);
    return;
  }
  const dueDateTimestamp = Math.floor(Date.now() / 1000) + (period * 24 * 60 * 60);
  try {
    console.log(`Granting credit of ${amount} tokens to ${borrowerAddress}...`);
    await p2pLendingContract.methods.grantCredit(userAccount, borrowerAddress, amountIn
    Wei,
    dueDateTimestamp).send({ from: userAccount });
    console.log("Credit granted successfully.");
    navigate('/');
  } catch (error) {
    console.error("Error granting credit:", error);
  }
};
```



## 10. Grant Credit: Smart Contract (Back End)

### 10.1 Benefits of the Smart Contract

#### 10.1.1 Preventing Reentrancy Attacks

Reentrancy attacks are a common vulnerability in smart contracts. The «noReentrant» modifier prevents these attacks. The USDC contract does not offer this level of security.

#### 10.1.2 Regulated Access

The registration and locking mechanisms ensure that only verified and approved users can participate in the lending process.

#### 10.1.3 Embedded in the Swiss Law

By integrating the smart contract within the framework of Swiss law, this mechanism offers a significant advantage over standard crypto transactions which often exist in a legal grey area.

#### 10.1.4 Structured Lending Agreements

The smart contract explicitly manages loans, including the amount, lender, borrower and due date. This structure provides clarity and enforceability, ensuring that all parties are aware of their obligations and rights.

#### 10.1.5 Automated Enforcement of Terms

The terms of the loan are enforced automatically, including the due date for the repayment of the loan.

#### 10.1.6 Simplified Loan Repayment Process

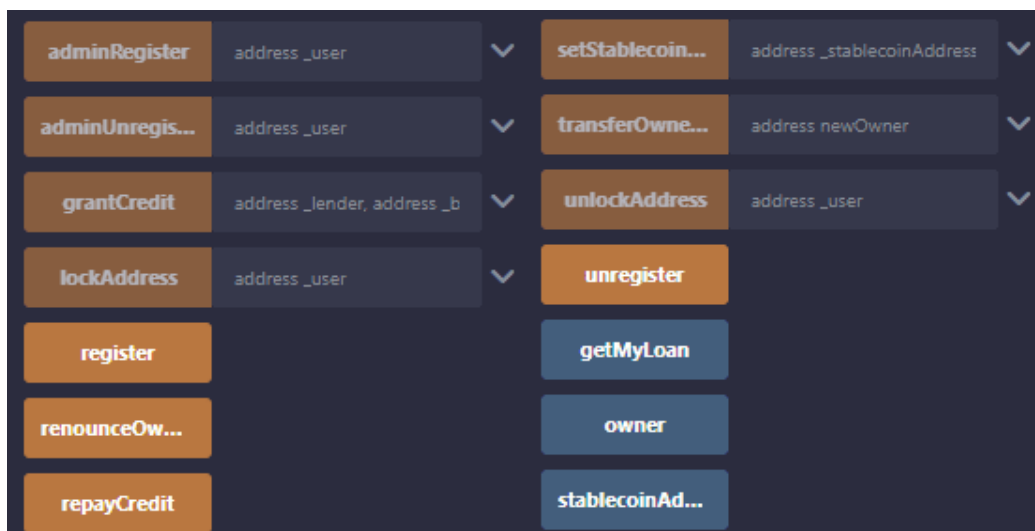
The «repayCredit» function automates the repayment process, directly transferring the owed amount from the borrower to the lender upon execution.

#### 10.1.7 Integrated Event Logging

The use of events offers immutable details of the transactions that can be useful for auditing, tracking and verifying the lending process.

## 10.2 Main functions of the smart contract

- **adminRegister:** Used by the contract owner to manually register a user on the platform, granting them the ability to engage in the lending process.
- **adminUnregister:** Allows the contract owner to manually unregister a user, removing their ability to participate in the lending process on the platform
- **grantCredit:** Transfers an approved amount of stablecoin from a lender to a borrower and records the loan details, including the due date.
- **lockAddress:** Utilized by the contract owner to lock a user's address. Once locked, the address cannot participate in registration-related activities.
- **register:** Allows a user to self-register on the platform to participate in lending activities.
- **renounceOwnership:** Removes the current owner's control over the contract, leaving it without an owner (only callable by the contract owner).
- **repayCredit:** Allows a borrower to repay their loan by transferring the owed stablecoin amount back to the lender and clears the loan record.
- **setStablecoinAddress:** Configures the contract to use a specified ERC-20 stablecoin address for all lending transactions (only callable by the contract owner).
- **transferOwnership:** Changes the ownership of the contract to a new address, granting the new owner administrative privileges (only callable by the contract owner).
- **unlockAddress:** The contract owner can unlock a previously locked user's address, enabling them to participate in registration activities again.
- **unregister:** A registered user can call this function to unregister themselves from the platform.
- **getMyLoan:** Retrieves the details of the active loan for the caller, including the lender's address, loan amount, and due date.
- **owner:** Returns the address of the current owner of the contract.
- **stablecoinAddress:** Displays the address of the ERC-20 stablecoin that the contract is currently using for transactions.



### 10.3 Mechanism of the Token Transfer in the Smart Contract

The smart contract first checks the following modifiers when triggering the «grantCredit» function:

- `noReentrant`: Prevents reentrancy attacks by ensuring that the function cannot be called again before it finishes executing
- `isRegistered`: checks if the borrower's address is registered
- `addressNotLocked`: Ensures that the caller's address is not locked and can perform actions within the contract

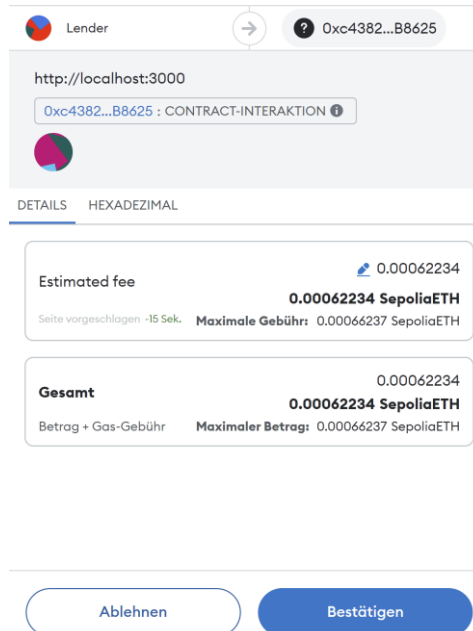
If these conditions are met, the function checks that the borrower does not already have an outstanding loan. Then the function checks if the lender has approved a sufficient allowance to the contract to spend the stablecoin amount intended for the loan. Then the «transferFrom» method is called to securely move the specified amount of stablecoins from the lender's account to the borrower's account. Upon a successful transfer, the function records the loan details (no sensible data) in the private «loans» mapping. Finally, the event «CreditGranted» is triggered to ensure that the loan was granted.

```
function grantCredit(address _lender, address _borrower, uint256 _amount, uint256 _dueDate)
    public noReentrant isRegistered addressNotLocked(msg.sender) {
    require(loans[_borrower].amount == 0, "Active loan exists"); // Prevent loan
    overwriting
    uint256 allowance = IERC20(stablecoinAddress).allowance(_lender, address(this));
    require(allowance >= _amount, "Allowance too low"); // Check allowance

    require(IERC20(stablecoinAddress).transferFrom(_lender, _borrower, _amount),
        "Transfer failed");
    loans[_borrower] = Loan(_lender, _amount, _dueDate);
    emit CreditGranted(_lender, _borrower, _amount, _dueDate);
}
```

## 11. Token Transfer

After pressing the «Grant Credit» button, the creditor can confirm the interaction with our smart contract currently at the address [0xc4382b37B596b7bD2B2C20F34992aF9fbf3B8625](http://localhost:3000/0xc4382b37B596b7bD2B2C20F34992aF9fbf3B8625).



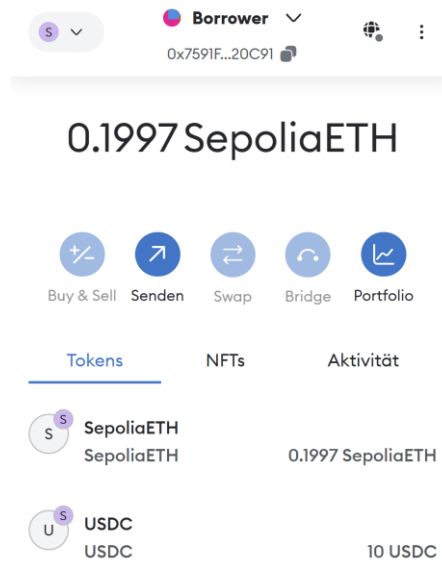
On MetaMask, you can see the confirmation of the spending limit approval, followed by the confirmation of the interaction with our smart contract.



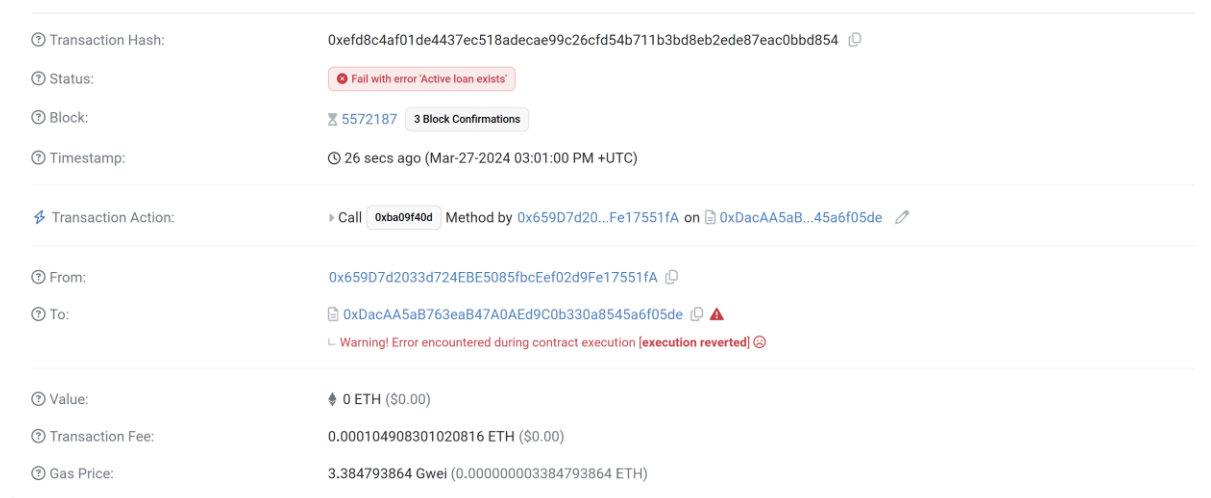
When you navigate to the explorer of Sepolia on Etherscan you can view the transaction details. On the bottom you see that 10 USDC have been transferred from the lender to the borrower of the loan.

Transaction Hash:	0x04cad2c7dbe23a7496311471e4f86813297c4feded87728d71df8c7ef63ec80f
Status:	Success
Block:	5577637 11 Block Confirmations
Timestamp:	2 mins ago (Mar-28-2024 10:20:36 AM +UTC)
Transaction Action:	Call 0xba09f40d Method by 0x659D7d20...Fe17551fA on 0x74769AD8...649bd5e7F
From:	0x659D7d2033d724EBE5085fbcEef02d9Fe17551fA
Interacted With (To):	0x74769AD8DA8C14bD2c2306ABcC3dD0E649bd5e7F
ERC-20 Tokens Transferred:	<div>All Transfers Net Transfers</div> <p>From 0x659D7d20...Fe17551fA To 0x7591Fd20...1f4520C91 For 10 USDC(USDC)</p>

In the MetaMask wallet of the borrower of the loan you can see the received 10 USDC tokens.



In the case that a loan for that user already exists the transaction will fail with the error «Active loan exists». When other requirements are not fulfilled there is a similar error message. Therefore the security measures in the smart contract prevent reentrancy attacks and that a borrower receives multiple loans.



## 12. Next Implementations

- Enhance the user experience (e.g. with loading displays) on the web platform
- Build a page for the pay back mechanism
- Implement the back end of the web platform that securely stores loan requests associated with a specific user