# CSE:586 Blockchain

Coursify
(Market Place to learn New Skills)

Ishant Rana
ishantra@buffalo.edu
50416749

Sanyam Gupta
sanyamgu@buffalo.edu
50359154

Learn Continuously - there's always one more thing to learn. - **Steve Jobs**

# 1. Introduction

**Motivation -**
Learning is a privilege that should be accessible to everyone in the world. But not everyone has the resources to engage in active learning. Either they don't have a skilled teacher, means, or both. Skill development is a decentralized process, only a teacher and student are required. No intermediate platform which acts as a broker should be ideally required. Having an intermediary introduces inefficiencies that can be mitigated. The benefits of having an intermediary can be automated through a process. Additionally, intermediaries generally take a cut from both teacher and student which drive the cost upward. Through smart contracts, the role of intermediary in a learning platform process can be eliminated. Blockchain also makes data openly verifiable which streamlines the verification of learned credentials.

| **Use case: Marketplace for skill development (Coursify)**<br><br>Problem statement: The intermediaries in online learning platforms introduce inefficiencies in the process. | **Issues with existing centralized solutions:**<br><br>1. Platforms are not available in all geographies.<br>2. Intermediaries take percentage cut, driving up overall costs and profits for learners and offereres alike.<br>3. Courses offered/demanded can be for a specific skill which are not offered by the platform.<br>4. The credentials and learning status is not openly verifiable. |
|---|---|
| **Proposed blockchain based solution.**<br><br>1. Course offerers can independently create a course.<br>2. Learners can search for course of interest, and enroll.<br>3. The learners credentials can be verified.<br>4. Seamless payments.<br>5. Course learner for a course can also be course offerer for same or different course. | **Benefits:**<br><br>1. No central authority controls the platform.<br>2. No cuts from intermediaries, hence overall cost is not driven up.<br>3. Automated system for recognizing users and their learning level.<br>4. Course learner can also be course offerer.<br>5. Courses and course content is not restricted by geography. |

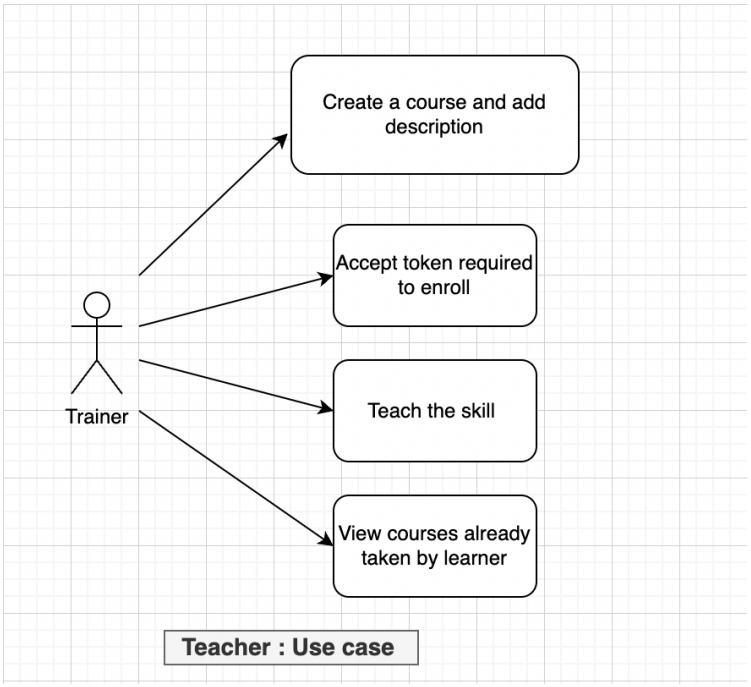
**Quad chart diagram**

**Proposed solution -**

A decentralized marketplace where primarily 2 entities interact with each other.

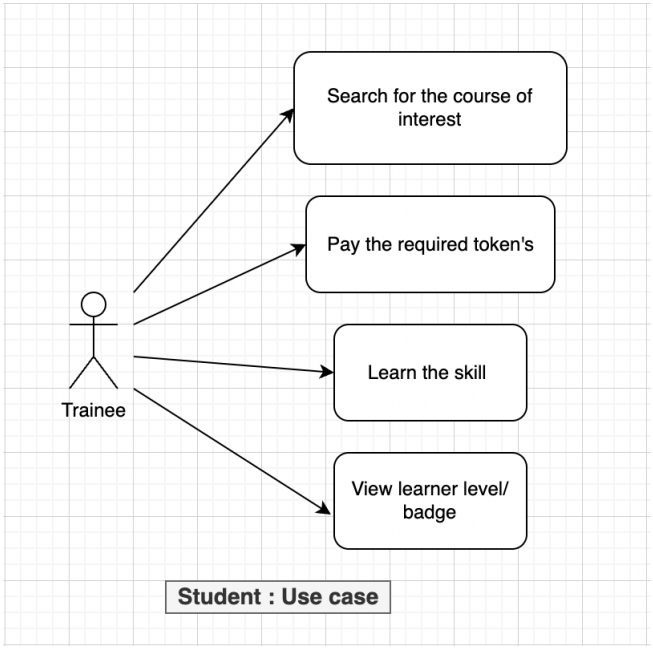
1. Course offerer (Teacher)

    They create a course, set the course structure, cost, logistics etc. They are not selective about the students they enroll in the course. So, they accept the prespecified payments and teach the course.




2. Courses Learner (Student)

They search for the course pertinent to the skill they wish to learn about. After finding the course they must pay the cost required by the course creator. Once the payment has been made, they can begin the learning process. They also get a badge corresponding to their current learning level. This is gauged by the number of courses, they have learned.



## 2. Token Info -

A token is a fungible unit that can represent any asset. It could be ticket, currency, share, etc. It could be virtually anything on Ethereum. Token named “Eth-learn” will be exchanged by the interacting parties for their services. The course offerer will specify the number of tokens required to enroll in the course. The learner will need to have a balance of required tokens, they need to be transferred to the course offerer, to enroll in the course.

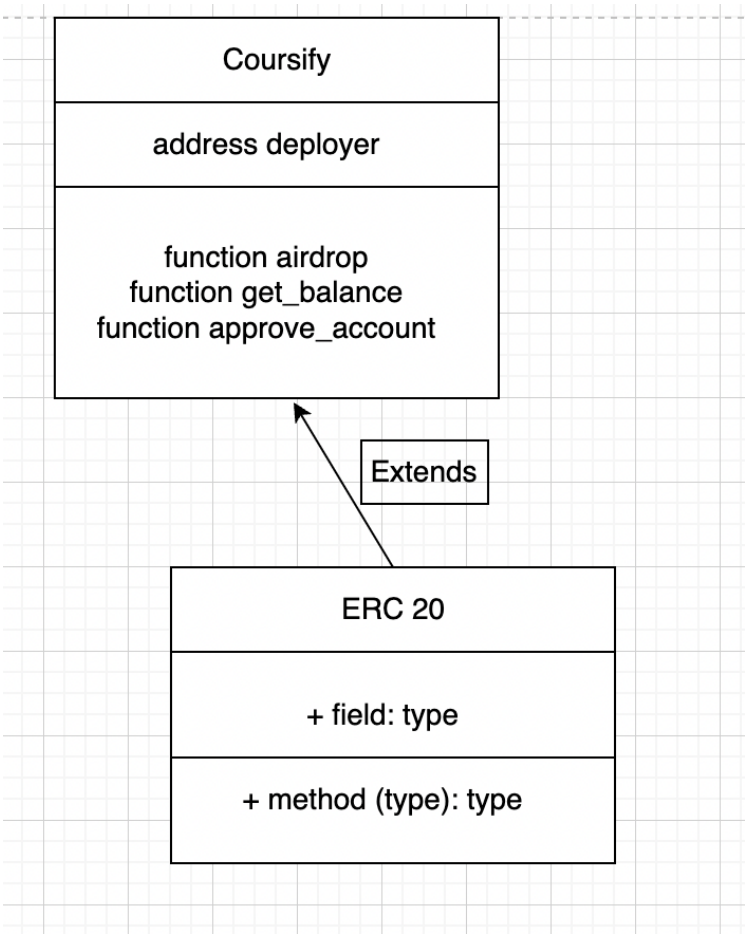
EIP (Ethereum Improvement Proposals) defined an interface for the ERC20 standard. This interface is specified as IERC20 in the openzepplin package. This interface is implemented in the ERC20 package. This package is imported and is being used in the smart contract. Another smart contact Ownable is also being used.

The main functionalities of the standard include the transfer of tokens to an account, querying the balance of an account, querying the total amount of token supply, and approval for token transfer by a third-party account.

This is done by the following functions:-

- totalSupply() - get the total amount of tokens.
- BalanceOf(account) - get token balance for *account.*
- transfer(to, amount) - increment token balance of *to* by *amount.*
- allowance(owner, spender) - return tokens from spender to owner
- approve(spender, amount) - allow spender to get a specified number of tokens
- transferFrom(from, to, amount) - execute the above-approved function.

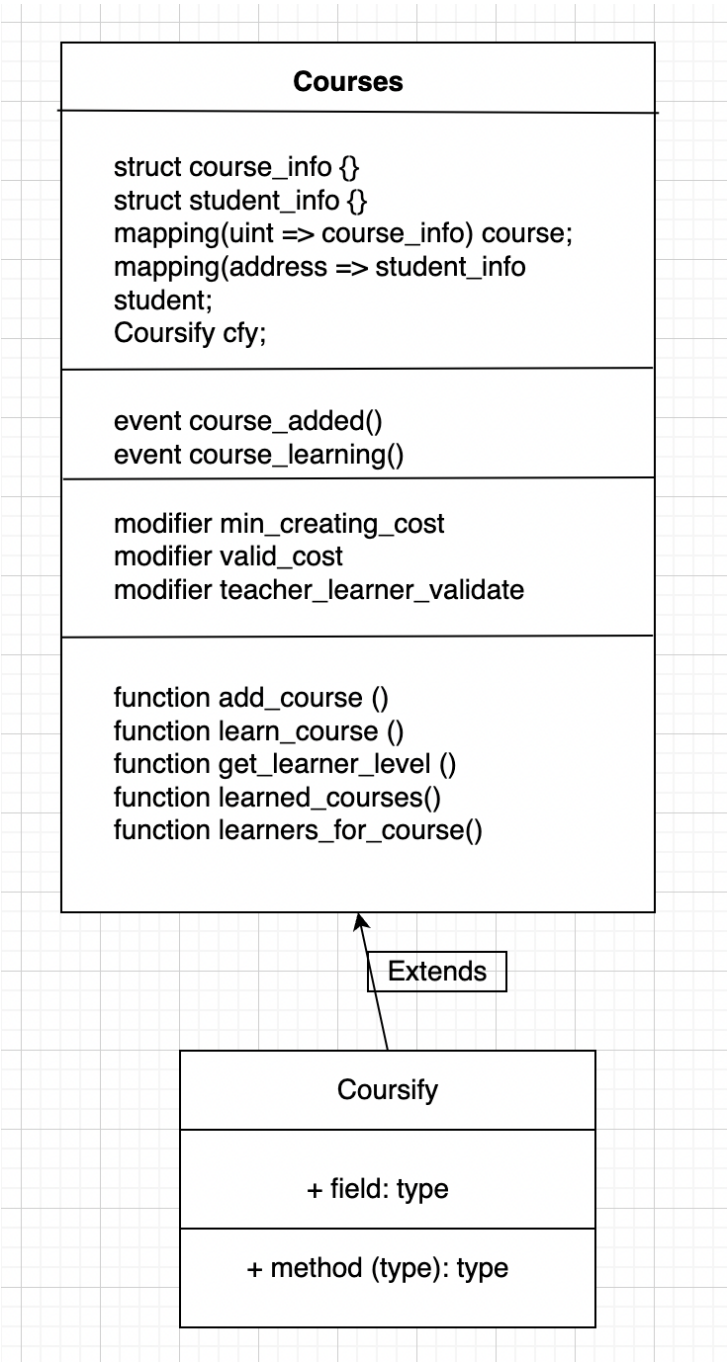
The openzepplin module is imported and inherited in the implementation of the smart contract module.




**Class Diagram for Coursify (token)**

The Courses smart contract is initialized with the address of deployed Token. An object of token is used for subsequent actions.

**Addresses on ropsten:**
Smart contract address (courses) : 0x4F16dF3616995285C3Cb05B595f28ec7DB505e20
Token address (coursify) : 0x687885B020A5d7169A178b7C398F8E1028AF977B
Deployed by address : 0x6089a732e305EEd1FFdA95A31ef3F0C256Eaf50c




**Class Diagram for Courses**

Name: Eth-learn

Logo -

Rationale - The C programming language was how we were introduced to the computing world. The knowledge changed our life. Since each new skill has the potential to change a person's life, this is a tribute to the skill that changed ours.

## 3. Instructions to deploy

You can also check the README.md file in the code.

In the project directory, you can run:

Deployment Already done on Ropsten using Remix by following Steps:

1. ERC20 Token Contract at
2. Marketplace Contract at

The abi generated is integrated in the dApp.
Start the dApp following these commands:

1. Start database server
# Run npm install -g json-server
# json-server --watch db.json --port 3001

2. Navigate to app directory
# cd app

3. Install the node modules
# npm install

4. Start the website
# cd ../app
# npm start

# 4. Code walkthrough

There are mainly 3 functions :

**1. GET Courses**
We are fetching the courses offered from our database.
Each courses have 5 data points id, name, description,cost and creator hash

```
getCourses(){
    this.setState({show_loading:true})
    let instance = this;
    fetch("http://localhost:3001/courses",{
        mode: 'cors',
        headers: {
            'Access-Control-Allow-Origin':'*'
        }
    })
    .then(res => res.json())
    .then(result =>{
        console.log(result);
        instance.setState({courses_list:result,show_loading:false});
    })
    .catch(console.log);
}
```

Coursify

Create Course

My Balance: 9611

AirDrop to Address | 0 | Drop | My Drops

Approve Amount | Approve

Levels: Bronze Silver Gold Platinum

My Level: bronze / Courses: 0

Courses Offered

1. Java Creator

Learn Java

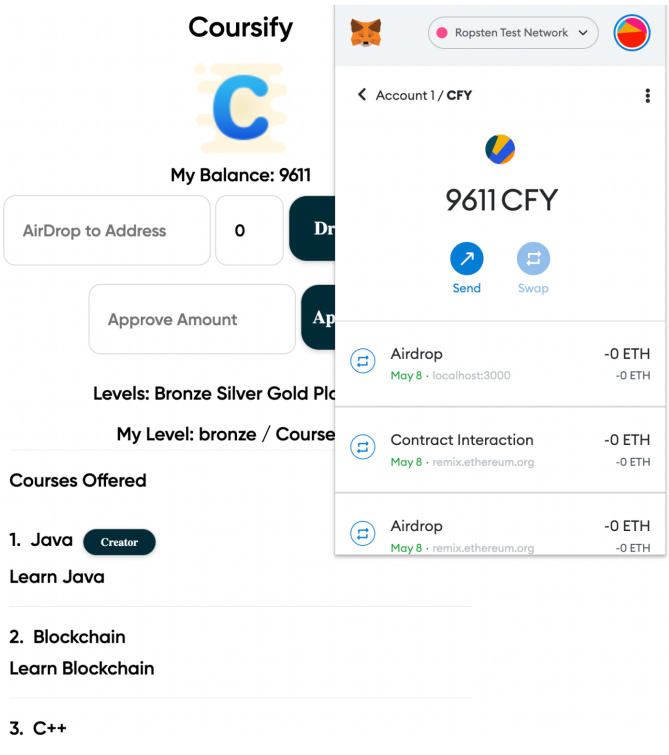
2. Blockchain

Learn Blockchain

3. C++

Learn C++

## User Screen with information

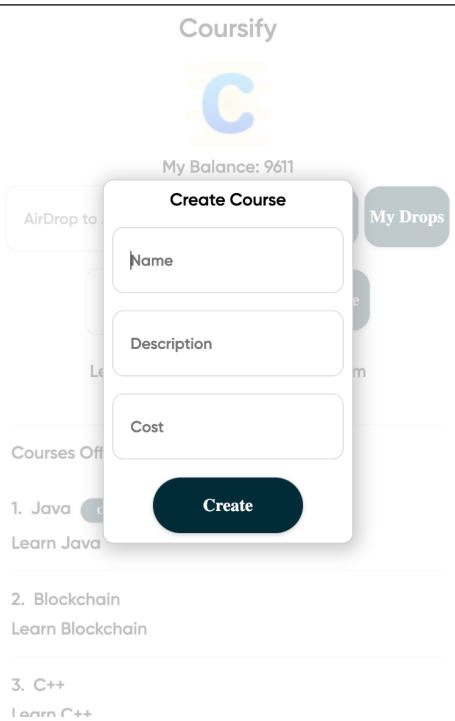


**2. Create Course**

We are creating courses with input parameter as Name, Description and Cost
And also making a call to our smart contract and creating a course on the chain.

```
async createProduct(id,cost) {
    this.setState({show_create_course:false,show_loading:true});
    this.state.courses.methods
      .add_course(id, cost)
      .send({ from: this.state.account })
      .once('receipt', receipt => {
        console.log(receipt);
        this.setState({show_loading:false});
      });
    // .then(this.setState({ loading: false }));
}
```

```
createCourse(name,description,cost){
    let data = {name,description,cost};
    let instance = this;
    fetch("http://localhost:3001/courses", {
    method: "POST",
    mode: 'cors',
        headers: {
            'Access-Control-Allow-Origin':'*'
        },
    headers: {
    "Content-Type": "application/json"
    },
    body: JSON.stringify(data)
    }).then(res => res.json())
    .then(res => {
        // instance.getCourses();
        var courses_list = instance.state.courses_list;
        courses_list.push(res);
        instance.setState({courses_list})
        instance.createProduct(res.id,res.cost);
    }).catch(console.log);
}
```

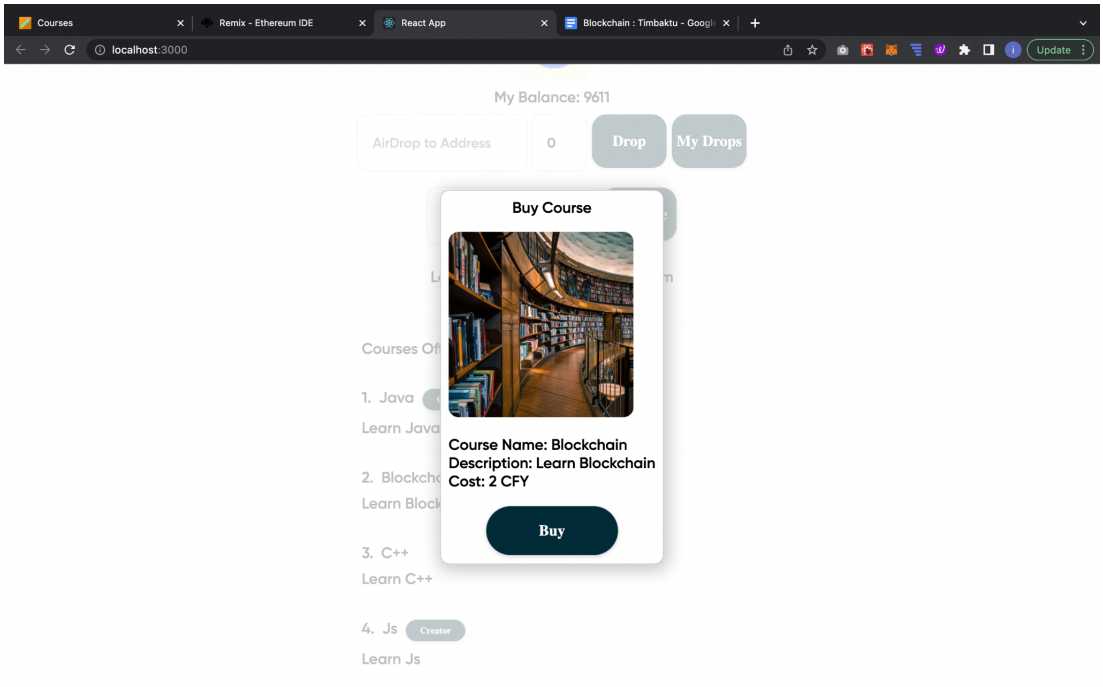



**Create course page**

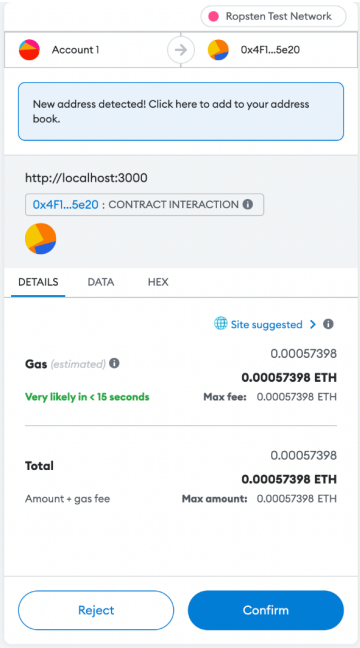
### 3. Buy Course

In this the user can buy a course by interacting with smart contracts.
In the smart contract we are updating the courses enrolled by the user and the current level. To buy the course we are passing the course id and the account of the user.

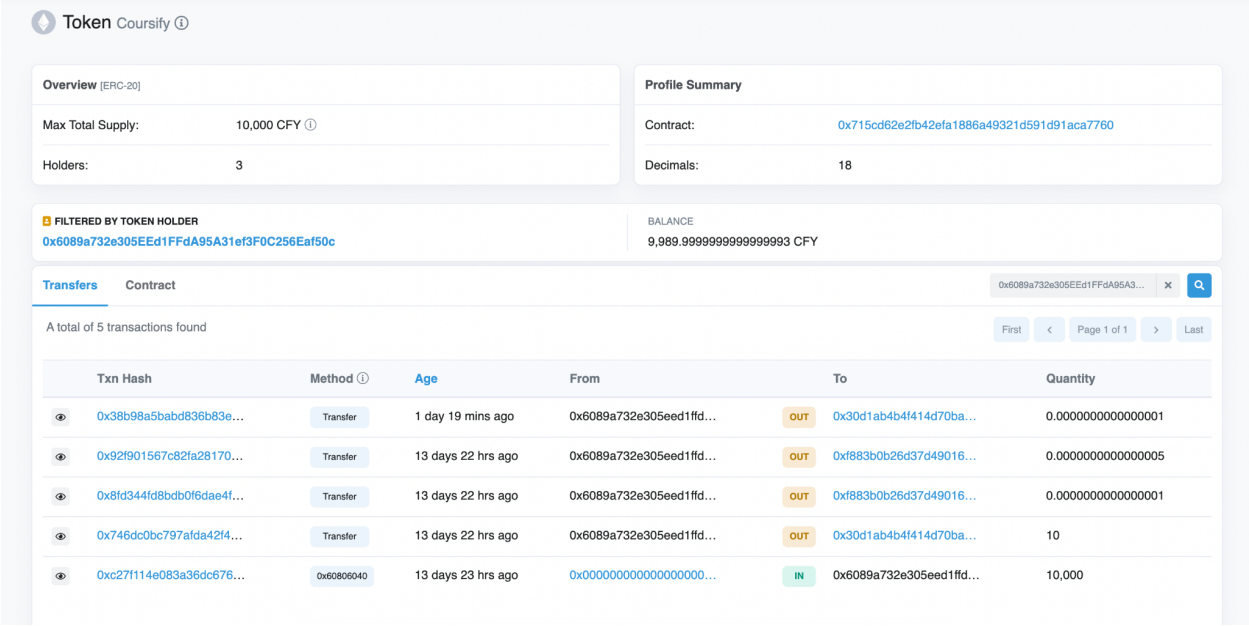
```
async purchaseProduct(id, cost) {
    await this.setState({ show_loading: true });
    await this.state.courses.methods
      .learn_course(id)
      .send({ from: this.state.account, value: cost })
      .once('receipt', receipt => {
        this.setState({show_loading: false});
      });
  }
```



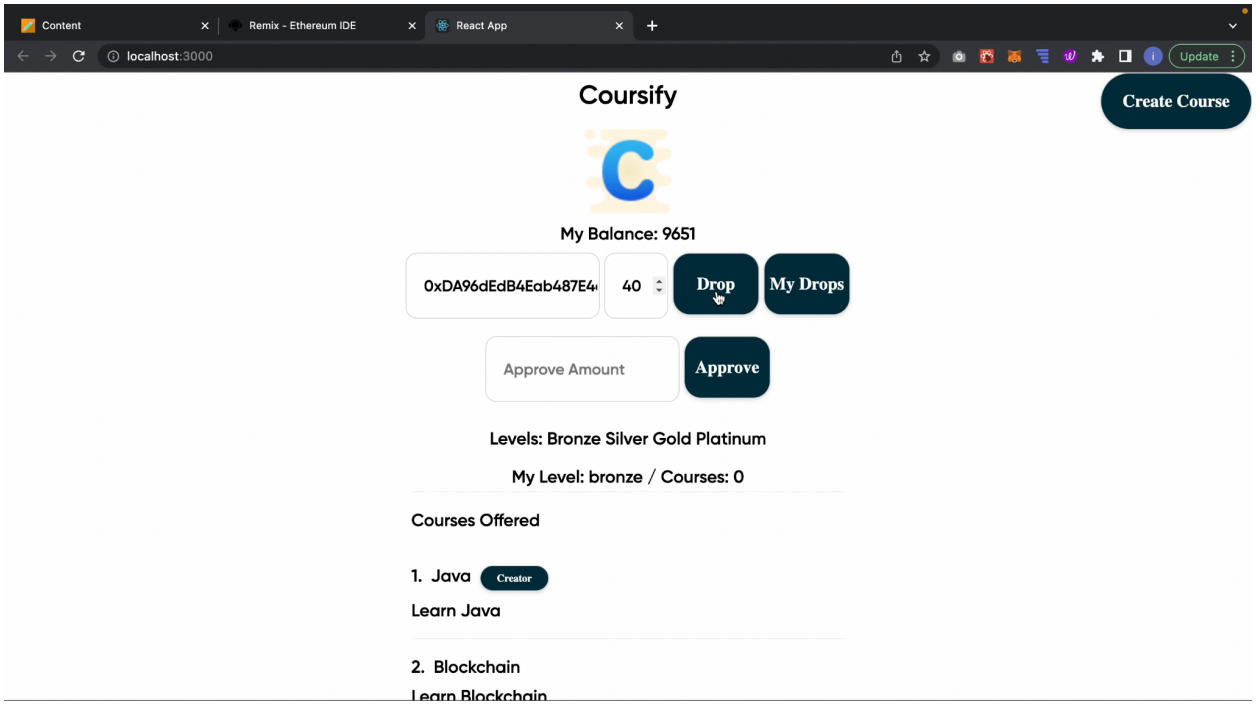

## Create home page

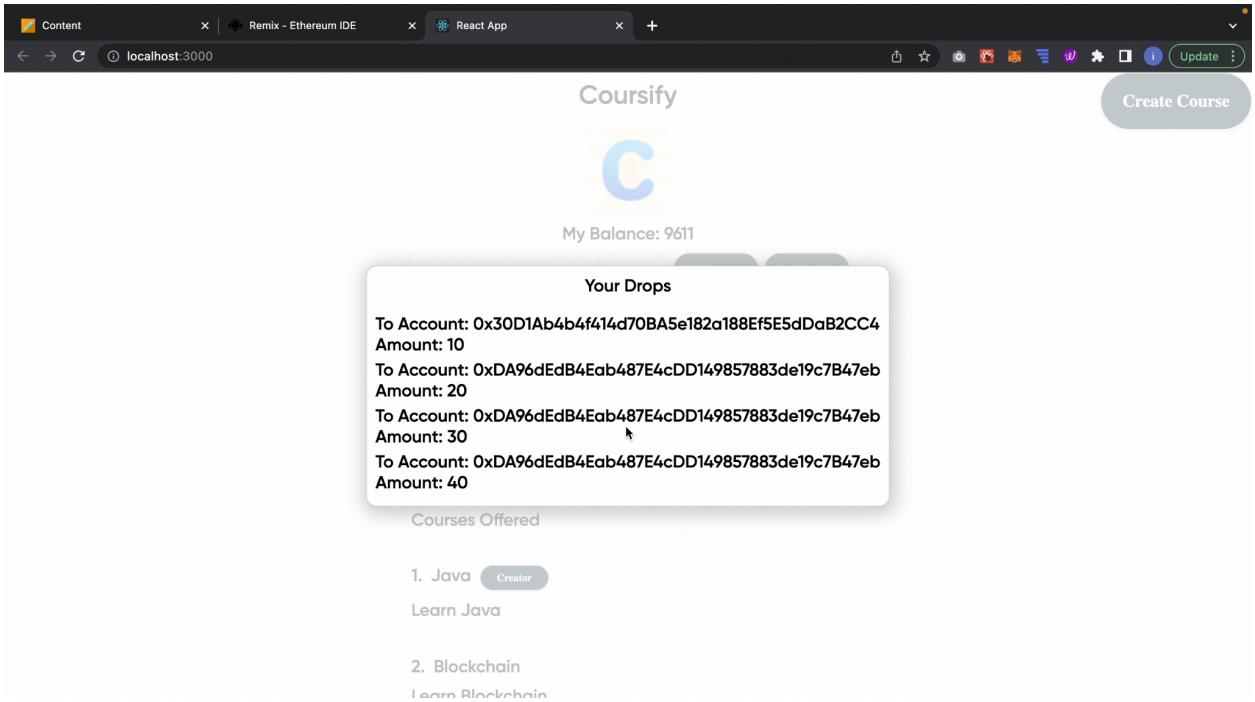


## 4. Coursify Token on Ropsten created by owner account




## 5. Airdrop






**Check your drops stored in offchain database**

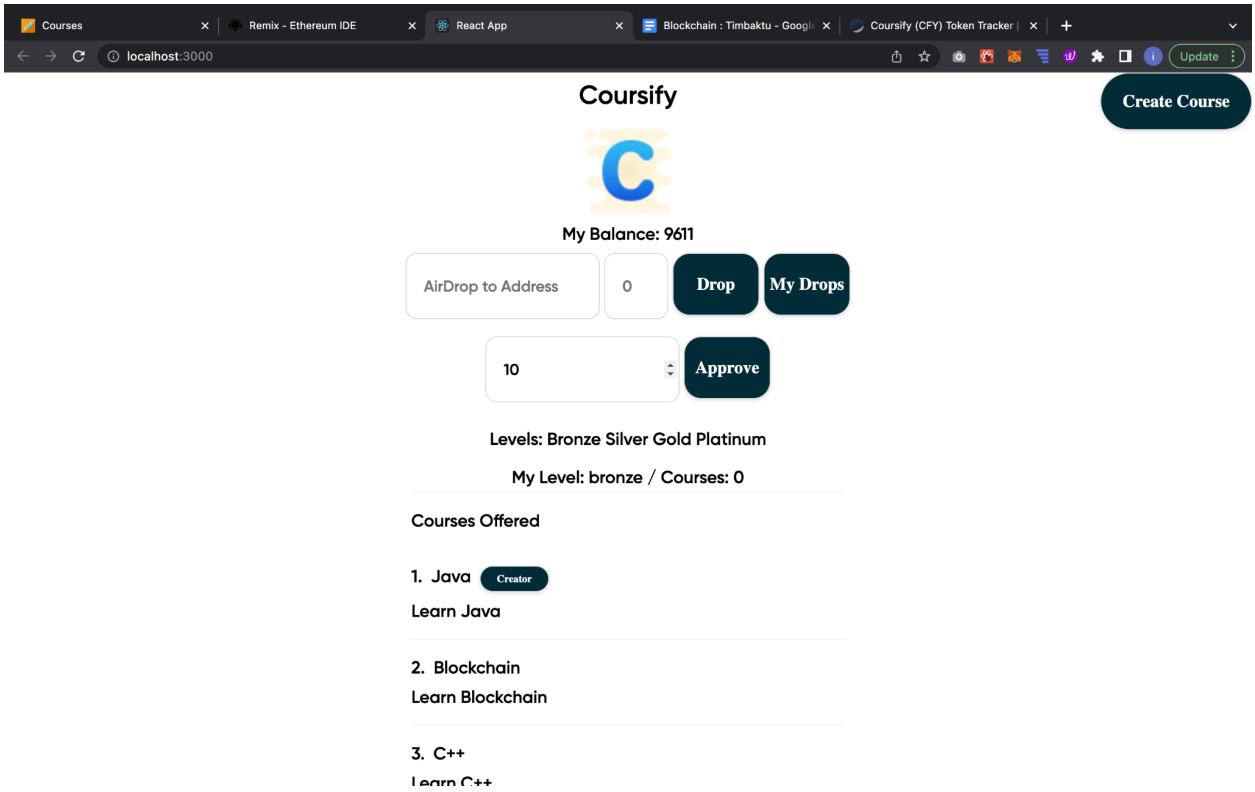
```
async executeAirdrop(){
    if(this.state.airdrop_address.length == 0 || this.state.airdrop_amount <= 0){
        return;
    }
    this.setState({ show_loading: true });
    await this.state.token.methods
      .airdrop(this.state.airdrop_address,parseFloat(this.state.airdrop_amount))
      .send({ from: this.state.account})
      .once('receipt', receipt => {
        // var courses_enrolled = this.state.courses_enrolled;
        // courses_enrolled.push(id);
        console.log(receipt);
        // this.airdropLocal(this.state.account,this.state.airdrop_address,parseFloat(this.state.airdrop_amount));
        this.airdropLocal(this.state.account,this.state.airdrop_address,parseFloat(this.state.airdrop_amount));
        this.setState({show_loading: false,airdrop_address:'',airdrop_amount:0});
        this.getAccountBalance()
      });
}
```

```
airdropLocal(from,to,amount){
    let data = {from,to,amount};
    fetch("http://localhost:3001/airdrops/", {
    method: "POST",
    mode: 'cors',
        headers: {
            'Access-Control-Allow-Origin':'*'
        },
    headers: {
    "Content-Type": "application/json"
    },
    body: JSON.stringify(data)
    }).then(res => res.json())
    .then(res => {
        console.log(res);
    }).catch(console.log);
}
```

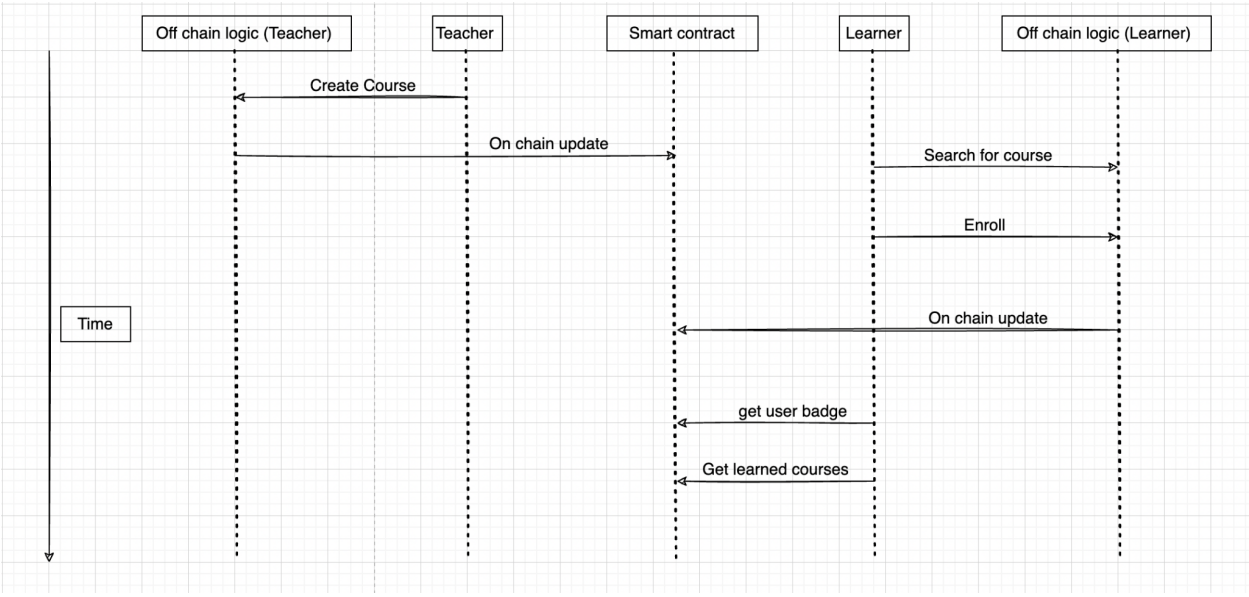
**Store drop to offchain db**

6. Approve Amount

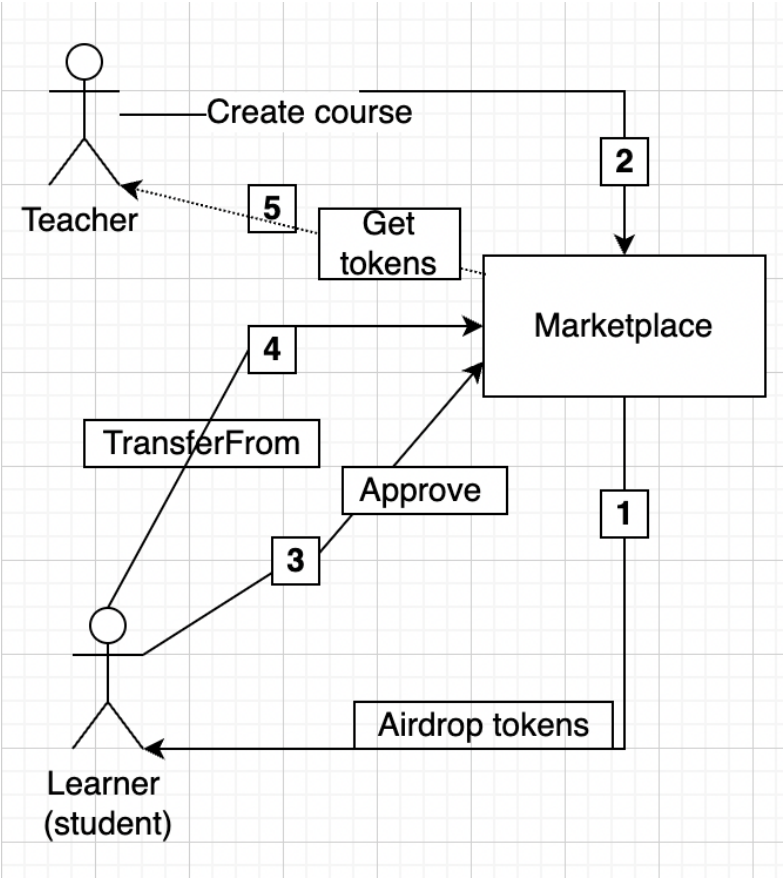
```
async approveAmount(amount){
    this.s any te({show_loading:true});
    await this.state.token.methods
      .approve_account(this.state.courseAccount,parseFloat(amount))
      .send({from: this.state.account})
      .once('receipt', receipt => {
        // var courses_enrolled = this.state.courses_enrolled;
        // courses_enrolled.push(id);
        console.log(receipt);
        this.setState({show_loading: false});
      });
}
```


Courses
Remix - Ethereum IDE
React App
Blockchain : Timbaktu - Googl
Coursify (CFY) Token Tracker
localhost:3000
Update
Coursify
Create Course
My Balance: 9611
AirDrop to Address
0
Drop
My Drops
10
Approve
Levels: Bronze Silver Gold Platinum
My Level: bronze / Courses: 0
Courses Offered
1. Java
Creator
Learn Java
2. Blockchain
Learn Blockchain
3. C++
Learn C++

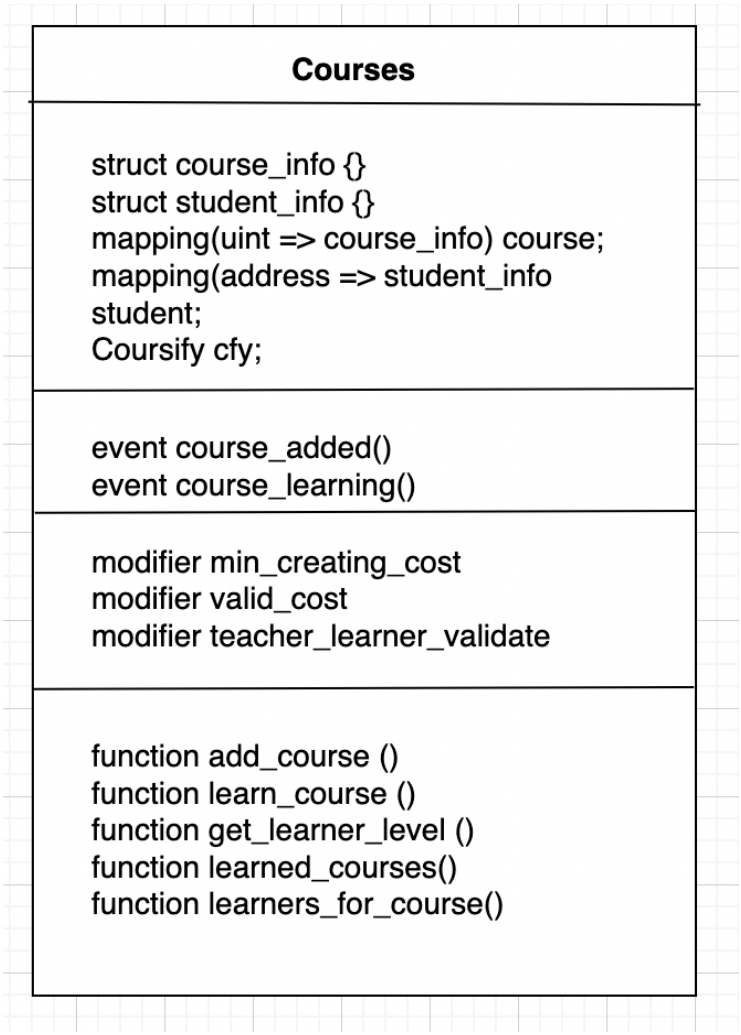
# 5. Smart Contract




**Sequence Diagram**




**Application Flow**




**Contract Diagram**

## Code Explanations -

1. Data Structures

1. Course_info - Structure to track the course information. It stores the address of the instructor and the credits required for the course.
2. Student_info - Structure to track the courses a learner has paid for. The number of courses also gets the user a badge, which can be openly verified by anyone.
3. Course - Mapping to link a unique id that is mapped via an off-chain logic for uniquely identifying a course with the course_info struct.
4. Student - Mapping to link learner's address to student_info struct.
5. Cfy - This is a object of token. Which is used for subsequent actions.

2. Events

1. Course_added - emitted when a new course is added.
2. Course_learning - emitted upon successful enrolment in a course.

3. Modifiers -
    1. min_creation_cost - the cost of a course has to be greater than 0.
    2. valid_cost - It checks if the cost paid by the enroller is greater than the cost specified by the course creator.
    3. teacher_learner_validate - learner of a course can be the teacher of the course.
4. Functions -
    1. add_course - create a new course by taking a unique id(off-chain logic) and the cost specified by the course creator.
    2. learn_course - the learner of the course enrolls in the course by paying the cost for the course.
    3. get_learner_level - Get the level(bronze, silver, gold,  platinum) a learner is on.
    4. learners_for_course - get the learners of a course.
    5. learned_courses - get courses a learner has enrolled in.

## Code implementations

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

// import "hardhat/console.sol";
import "./token.sol";

contract Courses {
   struct course_info{
       address payable instructor_id;
       uint credits_required;
       address[] teaching_to;
       address[] taught_to;
   }

   struct student_info{
       uint[] learned;
       uint learnedCourses;
   }

   mapping(uint => course_info) course;
   mapping(address => student_info) student;
   Coursify cfy;

   modifier min_creation_cost(uint course_cost){
```

```
        require(course_cost > 0);
        _;
    }

    modifier valid_cost(uint minCost, uint currCost){
        require(currCost*10**18 >= minCost);
        _;
    }

    modifier teacher_learner_validate(address teacher, address learner){
        require(teacher != learner);
        _;
    }

    event course_added(
        address payable instructor_id,
        uint credits_required
    );

    event course_learn
    vaing(
        address learner_id,
        uint course_id,
        uint credits_payed
    );

    constructor(address tokenAddress){
        cfy = Coursify(tokenAddress);
    }
    // constructor() ERC20("Coursify", "CFY") {
    //     // _mint(msg.sender, 10000 * 10);
    //     _mint(msg.sender, 10000 * 10 ** decimals());
    // }

    // function mint(address to, uint256 amount) public onlyOwner {
    //     _mint(to, amount);
    // }

    function add_course(uint course_id, uint cost) public min_creation_cost(cost){ //
teacher, invoker
        address payable inst_id = payable(msg.sender);
        uint tot_cost = cost * 10 ** 18;
```

```
        course[course_id].instructor_id = inst_id;
        course[course_id].credits_required = tot_cost;

        emit course_added({
            instructor_id: inst_id,
            credits_required : tot_cost
        });
    }

    // function learn_course(uint course_id, uint amount) public payable
valid_cost(course[course_id].credits_required, msg.value)
teacher_learner_validate(course[course_id].instructor_id, msg.sender){
    function learn_course(uint course_id, uint amount) public
valid_cost(course[course_id].credits_required, amount)
teacher_learner_validate(course[course_id].instructor_id, msg.sender){
        // address payable instructor_addr = payable(course[course_id].instructor_id);
        address payable instructor_addr = course[course_id].instructor_id;
        uint tot_amount = amount * 10 ** 18;


        // require(prep.allowance(msg.sender, address(this)) >= _course.price);
        // cfy.approve(address(this), tot_amount);
        uint allowance_balance = cfy.allowance(msg.sender, address(this));
        require(allowance_balance >= course[course_id].credits_required);

        // console.log("id", course[course_id].instructor_id, instructor_addr);

        cfy.transferFrom(payable(msg.sender), instructor_addr, tot_amount);

        course[course_id].teaching_to.push(msg.sender);
        student[msg.sender].learned.push(course_id);
        student[msg.sender].learnedCourses += 1;

        emit course_learning({
            learner_id: msg.sender,
            course_id: course_id,
            credits_payed: tot_amount
        });
    }

    function get_learner_level() public view returns(string memory badge){
```

```
        address studentAddr = msg.sender;
        uint numCourses = student[studentAddr].learnedCourses;

        if (numCourses <= 1){
            badge = "bronze";
        }
        else if (numCourses == 2){
            badge = "silver";
        }
        else if (numCourses == 3){
            badge = "gold";
        }
        else{
            badge = "platinum";
        }
    }

    function learners_for_course(uint course_id) public view returns(address[] memory
learners){
        learners = course[course_id].teaching_to;
    }

    function learned_courses() public view returns(uint[] memory courses){
        courses = student[msg.sender].learned;
    }
}
```

**Courses.sol**

```
                            // SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract Coursify is ERC20 {

    address payable deployer;
    constructor() ERC20("Coursify", "CFY") {
        _mint(msg.sender, 10000 * 10 ** decimals());
        deployer = payable(msg.sender);
```

```
    }

    function airdrop(address payable to, uint amount) public {
        uint tot_amount = amount * 10 ** decimals();
        _transfer(deployer, to, tot_amount);
    }

    function get_balance(address of_account) public view returns(uint
truncated_balance){
        uint amount = balanceOf(of_account);
        truncated_balance = amount/(10**decimals());
    }

    function approve_account(address spender, uint amount) public{
        approve(spender, amount*10**decimals());
    }
}
```

**token.sol**

## 6. References -


OpenZeppelin Documentation (https://docs.openzeppelin.com/contracts/4.x/api/token/erc20)
EIP-20: Token Standard (https://eips.ethereum.org/EIPS/eip-20)