**DApp2 – Lab 9**

For each function of the final project: give the name, the kind of function, the modifiers enforcing the constraints related to its call, the parameters, and a description of its purpose.

**Token.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| mint | public |  | onlyOwner | address \_to  uint256 \_value |
| Action – Notes:  Add \_value to totalSupply  Add \_value to balances[\_to]  Emit MintEvent | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| destroy | public |  | onlyOwner | address \_from  uint256 \_value |
| Action – Notes:  Subtract \_value from totalSupply  Subtract \_value from balances[\_from]  Emit DestroyEvent | | | | |

**DonationWallet.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| donate | public |  | onlyOwner | uint \_amount  string \_projectName |
| Action – Notes:  Charitable project address set from a catalog of charitable projects.  address projectAddress = projectCatalog.getProjectAddress(\_projectName);  token.approve(projectAddress, \_amount);  Approve token donation to a project  Project(projectAddress).donateFromWallet(\_amount);  Donate from donor wallet to project | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| donate | public |  | onlyOwner | uint \_amount  string \_projectName |
| Action – Notes:  Get the charitable project address from the catalog  address projectAddress = projectCatalog.getProjectAddress(\_projectName);  The project cannot be address(0)  require(projectAddress != address(0));  ERC20 token = Project(projectAddress).getToken(); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| refund | public |  | onlyOwner | ERC20 \_token  uint \_amount |
| Action – Notes:  Refund tokens  \_token.transfer(owner, \_amount); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| balance | public | view |  | Returns uint256 |
| Action – Notes:  Return token balance  return \_token.balanceOf(this); | | | | |

**ProjectWithBonds.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| investFromWallet | public |  |  | uint \_amount |
| Action – Notes:  *require*(getToken().transferFrom(*msg.sender*, beneficiaryAddress, \_amount));  uint256 couponCount = \_amount.div(couponNominalPrice);  Create a coupon so investor can retrieve investment  coupon.mint(*msg.sender*, couponCount);  liability = liability.add(getPriceWithInterests(\_amount));  *emit* CouponIssuedEvent(*msg.sender*, couponCount); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| validateOutcome | public |  |  | bytes32 \_claimId  uint \_value |
| Action – Notes:  Project validator must be the contract owner  *require* (*msg.sender* == validatorAddress);  Make sure validation amount is less than total  *require* (\_value <= total);  Projects are only paid out from the Escrow account after validation.  Subtract the validated amount from the total of investor capital.          uint256 unvalidatedLiability = liability.sub(validatedLiability);  *if* (\_value > unvalidatedLiability) {            uint256 surplus = \_value.sub(unvalidatedLiability);            getToken().transfer(beneficiaryAddress, surplus);            validatedLiability = validatedLiability.add(unvalidatedLiability);          } *else* {            validatedLiability = validatedLiability.add(\_value);          }          total = total.sub(\_value);          ImpactRegistry(IMPACT\_REGISTRY\_ADDRESS).registerOutcome(\_claimId, \_value);          emit OutcomeEvent(\_claimId, \_value); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| redeemCoupons | public |  |  | uint256 \_amount |
| Action – Notes:  Investor who has provided seed capital to charity, redeems a coupon to get their initial investment back plus interest.  uint256 redeemedValue = getPriceWithInterests(\_amount.mul(couponNominalPrice));  *require*(validatedLiability >= redeemedValue);          coupon.burn(*msg.sender*, \_amount);          getToken().transfer(*msg.sender*, redeemedValue);          liability = liability.sub(redeemedValue);          validatedLiability = validatedLiability.sub(redeemedValue);  *emit* CouponRedeemEvent(*msg.sender*, \_amount); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getPriceWithInterests | public | view |  | uint256 \_value  Returns uint256 |
| Action – Notes:  Investor receives 1% interest on their investment.  *return* \_value.add(\_value.mul(couponInterestRate).div(10000)); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getCoupon | public | view |  | *Returns* Coupon |
| Action – Notes: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getLiability | public | view |  | *return* liability |
| Action – Notes: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getValidatedLiability | public | view |  | *return* validatedLiability |
| Action – Notes: | | | | |

**ProjectCatalog.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| addProject | public |  | onlyOwner | string \_name  address \_projectAddress |
| Action – Notes:  Add a new charitable project to the catalog.  bytes32 nameAsBytes = \_name.stringToBytes32();  require(projects[nameAsBytes] == address(0));  projects[nameAsBytes] = \_projectAddress;  emit AddedProject(nameAsBytes, \_projectAddress); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getProjectAddress | public | view |  | string \_name  return projects[nameAsBytes] |
| Action – Notes:  bytes32 nameAsBytes = \_name.stringToBytes32(); | | | | |

**ImpactRegistry.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| registerDonation | public |  | onlyMaster | address \_from  uint \_value |
| Action – Notes:  Register the donation from a new donor.  if (accountBalances[\_from] == 0) {  accountIndex.push(\_from);  }  accountBalances[\_from] = accountBalances[\_from].add(\_value);  } | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| setMasterContract | public |  | onlyOwner | address \_contractAddress |
| Action – Notes:  masterContract = \_contractAddress; | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| setLinker | public |  | onlyOwner | ImpactLinker \_linker |
| Action – Notes:  linker = \_linker; | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| linkImpact | external |  | onlyOwner | bytes32 \_claimId |
| Action – Notes:  linker.linkImpact(\_claimId); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| payBack | public |  | onlyMaster | address \_account |
| Action – Notes:  accountBalances[\_account] = 0; | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| registerImpact | external |  | onlyLinker | bytes32 \_claimId  uint \_accountIndex  uint \_linkedValue |
| Action – Notes:  Impact storage impact = impacts[\_claimId];  address account = this.getAccount(\_accountIndex);  if (impact.values[account] == 0) {  impact.addresses[impact.count++] = account;  }  require(impact.value.sub(impact.linked) >= \_linkedValue);  updateBalance(\_accountIndex, \_linkedValue);  impact.values[account] = impact.values[account].add(\_linkedValue);  impact.linked = impact.linked.add(\_linkedValue);  } | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| updateBalance | internal |  |  | uint \_index  uint \_linkedValue |
| Action – Notes:  uint oldBalance = accountBalances[accountIndex[\_index]];  uint newBalance = oldBalance.sub(\_linkedValue);  accountBalances[accountIndex[\_index]] = newBalance;  if (newBalance == 0) {  accountIndex[\_index] = accountIndex[accountIndex.length-1];  accountIndex.length = accountIndex.length - 1;  } | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getAccountsCount | public | view |  | return accountIndex.length |
| Action – Notes: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getAccount | public | view |  | uint \_index  return accountIndex[\_index] |
| Action – Notes: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getBalance | public | view |  | address \_donorAddress  return accountBalances[\_donorAddress] |
| Action – Notes: | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getImpactCount | public | view |  | bytes32 \_claimId  return impacts[\_claimId].count |
| Action – Notes: | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Function Name | Function Visibility | | Function Type (pure/view/payable) | | Modifiers | | | Paramaters |
| getImpactLinked | public | | view | |  | | | bytes32 \_claimId  return impacts[\_claimId].linked |
| Action – Notes: | | | | | | | | |
| Function Name | | Function Visibility | | Function Type (pure/view/payable) | | Modifiers | Paramaters | |
| getImpactTotalValue | | public | | view | |  | bytes32 \_claimId  return impacts[\_claimId].value | |
| Action – Notes: | | | | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getImpactUnmatchedValue | public | view |  | bytes32 \_claimId |
| Action – Notes:  return impacts[\_claimId].value.sub(impacts[\_claimId].linked); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getImpactDonor | public | view |  | bytes32 \_claimId  uint index |
| Action – Notes:  return impacts[\_claimId].addresses[index] | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| getImpactValue | Public | view |  | bytes32 \_claimId  address addr |
| Action – Notes:  return impacts[\_claimId].values[addr]; | | | | |

**FlexibleImpactLinker.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| updateUnit | public |  | onlyOwner | uint \_value |
| Action – Notes:  unit = \_value; | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| linkImpact | external |  | onlyRegistry | bytes32 \_claimId |
| Action – Notes:  uint value = registry.getImpactTotalValue(\_claimId);  uint linked = registry.getImpactLinked(\_claimId);  uint left = value.sub(linked);  if (left > 0) {  uint i = linkingCursors[\_claimId];  address account = registry.getAccount(i);  uint balance = registry.getBalance(account);  if (balance >= 0) {  //Calculate impact  uint impactVal = balance;  if (impactVal > left) {  impactVal = left;  }  if (impactVal > unit) {  impactVal = unit;  }  registry.registerImpact(\_claimId, i, impactVal);  //Update index  if (balance == impactVal) {  i--;  }  uint accountsCount = registry.getAccountsCount();  if (accountsCount > 0) {  linkingCursors[\_claimId] = (i + 1) % accountsCount;  } else {  linkingCursors[\_claimId] = 0;  } | | | | |

**InvestmentWallet.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| invest | public |  | onlyOwner | uint \_amount  string \_projectName |
| Action – Notes:  Investor can invest funds in a charitable project basically providing startup capital.  address projectAddress = projectCatalog.getProjectAddress(\_projectName);  require(projectAddress != address(0));  ERC20 token = ProjectWithBonds(projectAddress).getToken();  token.approve(projectAddress, \_amount);  ProjectWithBonds(projectAddress).investFromWallet(\_amount); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| redeemCoupons | public |  | onlyOwner | uint \_amount  string \_projectName |
| Action – Notes:  Investor can redeem coupon for verified project completion, and retrieve initial investment plus interest.  address projectAddress = projectCatalog.getProjectAddress(\_projectName);  require(projectAddress != address(0));  ProjectWithBonds project = ProjectWithBonds(projectAddress);  Coupon coupon = project.getCoupon();  require(coupon.balanceOf(this) >= \_amount);  project.redeemCoupons(\_amount); | | | | |

**Coupon.sol**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| mint | public |  | onlyOwner | address \_to  uint256 \_value |
| Action – Notes:  Create a coupon that an investor can redeem in the future after a project has been validated.  totalSupply\_ = totalSupply\_.add(\_value);  balances[\_to] = balances[\_to].add(\_value);  emit MintEvent(\_to, \_value); | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function Name | Function Visibility | Function Type (pure/view/payable) | Modifiers | Paramaters |
| burn | public |  | onlyOwner | address \_from  uint256 \_value |
| Action – Notes:  Burn the coupon after an investor has redeemed a coupon to retrieve the investment plus interest.  totalSupply\_ = totalSupply\_.sub(\_value);  balances[\_from] = balances[\_from].sub(\_value);  emit BurnEvent(\_from, \_value); | | | | |