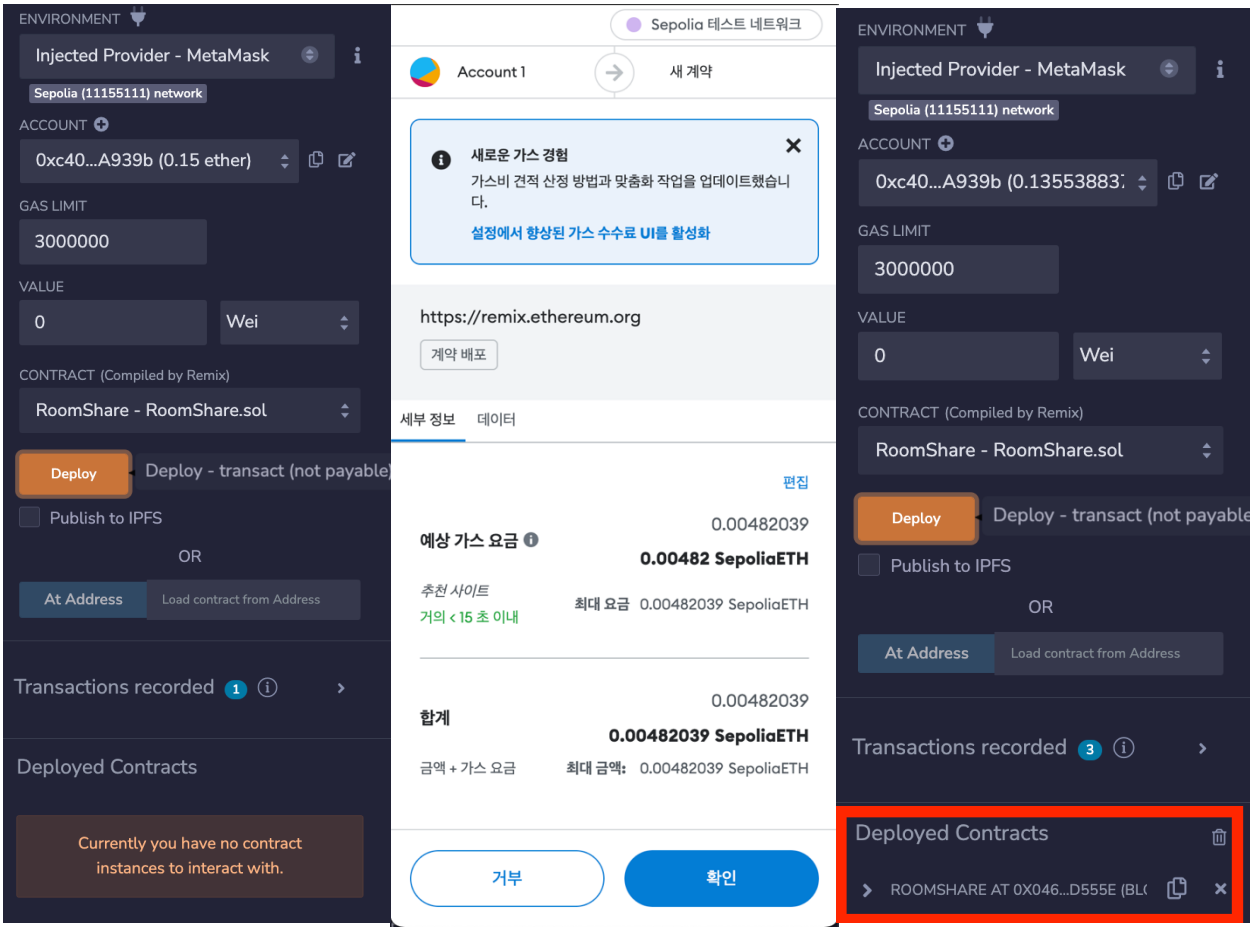
DAPP(RoomShare) Project

2018008304 컴퓨터소프트웨어학부 박경하

1. 실행과정

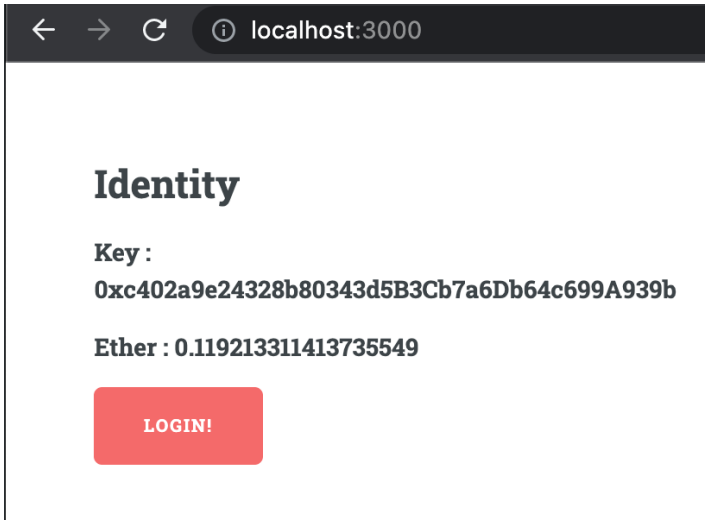
1) Deploy 후, web3interface.js파일의 smart_address 값 갱신



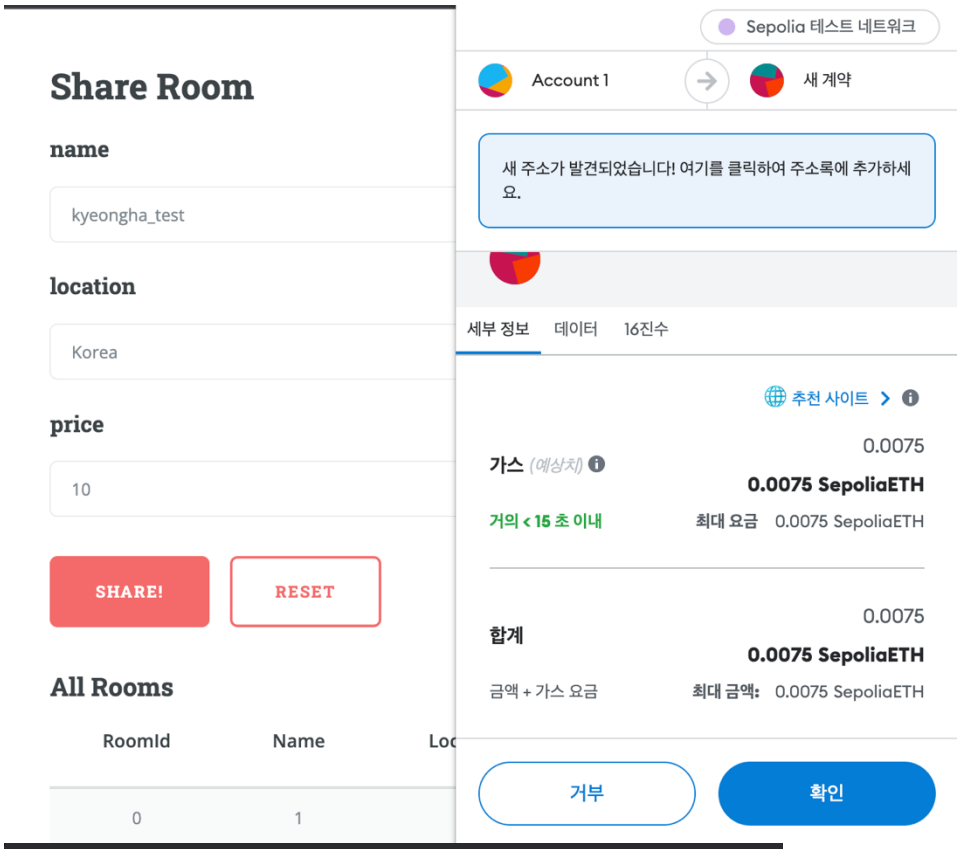
2) 작업폴더에서 npm을 이용하여 서버실행

```
~/desktop/HY/HYU_2022/D/RoomShare_Project git master !1 ?4 ❯ npm update

changed 1 package, and audited 606 packages in 24s

85 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
~/desktop/HY/HYU_2022/D/RoomShare_Project git master !1 ?4 ❯ npm start

> start
> node ./bin/www
```

3) localhost:3000 홈페이지에서 login



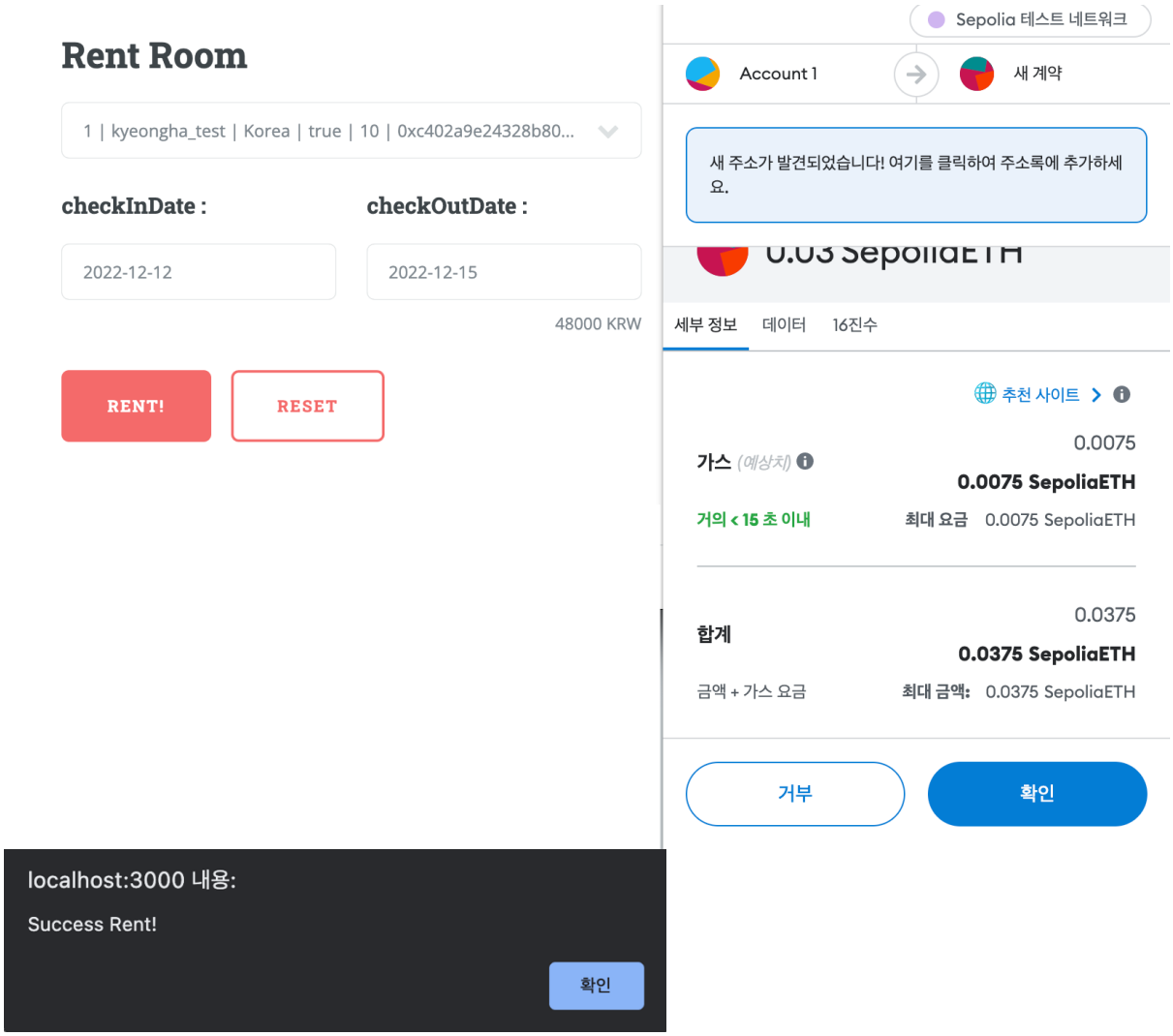
4) Share room 기능 수행



All Rooms

RoomId	Name	Location	IsActive?	Price	Owner
0	1	1	true	1	0xc402a...
1	kyeongha_test	Korea	true	10	0xc402a...

5) Rent Room 기능 수행



My Rents

RentNumber	RoomId	CheckInDate	CheckOutDate
0	1	Mon Dec 12 2022	Thu Dec 15 2022

Room History

RentNumber	CheckInDate	CheckOutDate	Renter
0	Mon Dec 12 2022	Thu Dec 15 2022	0xc402a9e243...

2. 코드 설명: sol파일

1) 구현을 위해 아래의 두 함수를 추가하였습니다.

```
function getRoomId() external view returns(uint256);
function getroomId2room(uint _roomId) external view returns(Room memory);
```

각각의 변수가 JS에 바로 리턴되지 않아 return해주는 함수를 추가하였습니다.

```
function getRoomId() public view override returns(uint256){
  return roomId;
}
```

getRoomId() 함수는 sender의 roomID을 리턴해주는 함수이고,

```
function getroomId2room(uint _roomId) public view override returns(Room memory){
  return roomId2room[_roomId];
}
```

getroomId2room는 인자의 roomId에 해당하는 room 배열을 리턴해주는 함수입니다.

따라서 IRoomShare.sol 파일도 함께 첨부하였습니다..

2) get 함수

```
function getMyRents() public view override returns(Rent[] memory) {
  /* 함수를 호출한 유저의 대여 목록을 가져온다. */
  return renter2rent[msg.sender];
}

function getRoomRentHistory(uint _roomId) public view override returns(Rent[] memory) {
  /* 특정 방의 대여 히스토리를 보여준다. */
  return roomId2rent[_roomId];
}
```

get함수들은 필요한 조건에 따라 즉각적으로 return 해주도록 하였습니다.

3) shareRoom 함수

```
function shareRoom(string calldata name, string calldata location, uint price) public override {
  /* 1. isActive 초기값은 true로 활성화, 함수를 호출한 유저가 방의 소유자이며,
        365 크기의 boolean 배열을 생성하여 방 객체를 만든다. */
  bool[] memory isRented = new bool[](365);
  for(uint i = 0; i < 365; i++)
    isRented[i] = false;

  Room memory targetRoom = Room(roomId, name, location, true, price, msg.sender, isRented);

  /* 2. 방의 id와 방 객체를 매핑한다. */
  roomId2room[roomId] = targetRoom;
  //roomId++;
  emit NewRoom(roomId++);
}
```

inRented에 365사이즈의 배열을 선언한 뒤, 모두 False값을 취해주었습니다. 이 후, 인자와 함께 room 객체를 하나 생성해주어 roomId2room 객체에 roomId와 매핑하여 push해주었습니다.

4) rentRoom 함수

```
function rentRoom(uint _roomId, uint checkInDate, uint checkOutDate) payable external override{
  /**
   * 1. roomId에 해당하는 방을 조회하여 아래와 같은 조건을 만족하는지 체크한다.
   *    a. 현재 활성화(isActive) 되어 있는지
   *    b. 체크인날짜와 체크아웃날짜 사이에 예약된 날이 있는지
   *    c. 함수를 호출한 유저가 보낸 이더리움 값이 대여한 날에 맞게 지불되었는지(단위는 1 Finney, 10^15 Wei)
   * 2. 방의 소유자에게 값을 지불하고 (msg.value 사용) createRent를 호출한다.
   * *** 체크아웃 날짜에는 퇴실하여야하며, 해당일까지 숙박을 이용하려면 체크아웃날짜는 그 다음날로 변경하여야한다. ***
   */
  Room memory targetRoom = roomId2room[_roomId];

  uint8 decimals = 15;
  uint dayPrice = targetRoom.price * (10 ** uint(decimals));

  //a. 현재 활성화 상태가 아니면, 에러메세지 발생
  require(targetRoom.isActive, "NotActive");

  //b. 해당 날짜가 렌트되어 있는 상태면, 렌트 불가능한 상태로 변경
  for(uint i = checkInDate; i < checkOutDate; i++){
    if(targetRoom.isRented[i-1] == true){
      require(false, "NotCheckDate");
    }
  }

  //c. 이더리움 값 지불이 올바르지 않으면, 렌트 불가능한 상태로 변경
  uint totalPrice = dayPrice * (checkOutDate - checkInDate);
  require((totalPrice == msg.value), "NotMatchPrice");

  //2. 조건 만족시, 방의 소유자에게 값을 지불하고 createRent 호출
  _sendFunds(targetRoom.owner, msg.value);
  _createRent(_roomId, checkInDate, checkOutDate);
}
```

인자로 받은 roomId을 통해 매핑되는 room 객체를 찾습니다.
매핑된 room의 정보에서 위의 조건과 일치하는지, 현재 대여가능한 상태인지 파악합니다.
대여가능한 상태가 아닐 경우, require을 통해 함수를 중단합니다.

4) _createRent 함수

```
function _createRent(uint256 _roomId, uint256 checkInDate, uint256 checkOutDate) public override{
  /* 1. 함수를 호출한 사용자 계정으로 대여 객체를 만들고,
        변수 저장 공간에 유의하며 체크인날짜부터 체크아웃날짜에 해당하는 배열 인덱스를 체크한다.
        (초기값은 false이다.). */
  Rent memory targetRent = Rent(rentId, _roomId, checkInDate, checkOutDate, msg.sender);
  for(uint i = checkInDate; i < checkOutDate; i++){
   roomId2room[_roomId].isRented[i-1] = true;
  }
  /* 2. 계정과 대여 객체들을 매핑한다. (대여 목록) */
  renter2rent[msg.sender].push(targetRent);
  /* 3. 방 id와 대여 객체들을 매핑한다. (대여 히스토리) */
  roomId2rent[_roomId].push(targetRent);

  emit NewRent(_roomId, rentId++);
}
```

방이 대여가능한 상태일시, 해당되는 date의 배열값을 false로 바꿉니다.
이후 계정과 방 id을 대여 개체와 매핑하여 리스트를 저장해둡니다.

4) recommendDate 함수

```
function recommendDate(uint _roomId, uint checkInDate, uint checkOutDate) public view override re
  /**
   * 대여가 이미 진행되어 해당 날짜에 대여가 불가능 할 경우,
   * 기존에 예약된 날짜가 언제부터 언제까지인지 반환한다.
   * checkInDate(체크인하려는 날짜) <= 대여된 체크인 날짜 , 대여된 체크아웃 날짜 < checkOutDate(체크아웃하려는 날짜)
   */

  Room memory targetRoom = roomId2room[_roomId];
  uint[2] memory checkDate = [checkInDate, checkOutDate];

  // 해당 날짜가 렌트되어 있는 상태면, 대여정보를 찾는다.
  for(uint i = checkInDate; i < checkOutDate; i++){

    if(targetRoom.isRented[i-1] == true) {
      Rent[] memory targetRent = roomId2rent[_roomId];
      // 대여하고 있는 방 정보 중에서 해당 날짜를 포함하고 있는 방을 찾는다.
      // 해당하는 방의 체크인 날짜와 체크아웃 날짜를 배열에 저장한다.
      for(uint j = 0; j < targetRent.length; j++){
        if(targetRent[j].checkInDate <= i << targetRent[j].checkOutDate){
          checkDate[0] = targetRent[j].checkInDate;
          checkDate[1] = targetRent[j].checkOutDate;
        }
      }
    }
  }
  return checkDate;
}
```

대여가 불가능할 경우, JS 파일에서 해당 함수를 호출하게 됩니다.

이후, check 함수에 매핑하여 언제 대여가 가능한지 조건에 맞게 리턴해주게 됩니다.

2. 코드 설명: js파일

1) _shareRoom 함수

```
const _shareRoom = async (name, location, price) => {
  // 에러 발생시 call 또는 send 함수의 파라미터에 from, gas 필드 값을 제대로 넣었는지 확인한다. (e.g. {from: ..., gas: 3000000, ...})

  /* RoomShareContract의 shareRoom 함수를 호출한다.
     방 이름, 위치, 하루당 대여 요금을 입력하고 컨트랙트에 등록한다. */
  const result = await contract.methods.shareRoom(name, location, price).send({from:user, gas:3000000}).catch(e=> {
    console.log(e);
  });
  /*  트랜잭션이 올바르게 발생하면 알림 팝업을 띄운다. (e.g. alert("등록")) */
  if(result)
    alert("등록");

  /*  화면을 업데이트 한다. */
}
```

sol 파일의 shareRoom함수를 불러올 때, user와 gas값을 입력하여 send 합니다. (트랜잭션을 생성하므로)
이후, 정상적으로 등록되면 result가 true가 되므로 등록 안내창을 띄웁니다.

2) _getMyRents 함수

```
const _getMyRents = async () => {
  // 내가 대여한 방 리스트를 불러온다.
  let myRents = [];
  const contractMyRent = await contract.methods.getMyRents().call({from:user}).catch(e=> {
    console.log('getMyRentsError: ', e);
    return myRents;
  });

  for (let i = 0; i < contractMyRent.length; i++){
    let myRentObject = {};

    myRentObject.id = contractMyRent[i].id;
    myRentObject.rId = contractMyRent[i].rId;
    myRentObject.checkInDate = contractMyRent[i].checkInDate;
    myRentObject.checkOutDate = contractMyRent[i].checkOutDate;
    myRentObject.renter = contractMyRent[i].renter;

    myRents.push(myRentObject);
  }
  return myRents;
}
```

getMyRents() 한 뒤, 각각의 room들을 myRent 배열에 push하여 리턴해줍니다.

3) _getAllRooms 함수

```
const _getAllRooms = async () => {
  // Room ID 를 기준으로 컨트랙트에 등록된 모든 방 객체의 데이터를 불러온다.
  const roomId = await contract.methods.getRoomId().call();
  let rooms = [];

  for(let i = 0; i < roomId; i++){
    let roomObject = {};
    const roomContract = await contract.methods.getroomId2room(i).call();

    roomObject.id = roomContract.id;
    roomObject.name = roomContract.name;
    roomObject.location = roomContract.location;
    roomObject.isActive = roomContract.isActive;
    roomObject.price = roomContract.price;
    roomObject.owner = roomContract.owner;
    roomObject.isRented = roomContract.isRented;

    rooms.push(roomObject);
  }

  console.log(rooms);
  return rooms;
}
```

roomId을 통해 contract함수에서 호출하여, 해당 room들을 불러와 roomObject에 저장한 뒤, rooms라는 배열에 push합니다.

4) _rentRoom 함수

```
const _rentRoom = async (roomId, checkInDate, checkOutDate, price) => {
  // 체크인 날짜와 체크아웃 날짜의 차이, 하루당 대여 요금을 곱하여 컨트랙트로 송금한다.
  // 단위는 finney = milli Eth (10^15)
  const totalPrice = price * Math.pow(10, 15);
  const result = await contract.methods.rentRoom(roomId, checkInDate, checkOutDate).send({from:user, gas:3000000, val
    // 이더의 양이 맞지 않아서 트랜잭션이 종료되었을 경우에는 다른 팝업을 띄운다. (Solidity의 require과 관련됨)
    if(e == "NotCheckDate"){
      alert("Please check Price.");
    }
    // Room ID에 해당하는 방이 체크인하려는 날짜에 대여되어서 대여되지 않는다면 _recommendDate 함수를 호출한다.
    if(e == "NotMatchPrice"){
      alert("NotCheckDate");
      _recommendDate(roomId, checkInDate, checkOutDate);
    }
  });
  // 대여가 성공하고 트랜잭션이 올바르면 알림 팝업을 띄운다.
  if(result)
    alert("Success Rent!");

  // 화면을 업데이트 한다.
  await _updateUserBalance(user);
  _updateRents();
  _updateRooms();
}
```

총 price을 계산하여 eth로 바꿔준 뒤, contract의 함수를 send합니다.
require을 통해 에러가 출력되면 아래의 경우에 따라 핸들링하게 됩니다.
- 금액이 맞지 않은 경우에는 팝업창을 띄웁니다.
- 이미 대여중인 날짜인 경우에는 recommendDate 함수를 호출하여 핸들링합니다.

5) recommendDate 함수

```
const _recommendDate = async (roomId, checkInDate, checkOutDate) => {
  // Room ID에 해당하는 방이 체크인하려는 날짜에 대여되었다면,
  // 기존에 대여된 날짜가 언제부터 언제까지인지 알림 팝업으로 표시한다.
  // checkInDate <= 대여된 체크인 날짜 , 대여된 체크아웃 날짜 < checkOutDate
  // 체크아웃 날짜에는 퇴실하여야하며, 해당일까지 숙박을 이용하려면 체크아웃날짜는 그 다음날로 변경하여야한다.
  // 주어진 헬퍼 함수 dateFromDay 를 이용한다.
  const recommendDate = recommendDate(roomId, checkInDate, checkOutDate).call();
  alert("대여된 체크인 날짜: " + dateFromDay(currentYear, recommendDate[0]).toDateString() +
        "\n대여된 체크아웃 날짜: " + dateFromDay(currentYear, recommendDate[1]).toDateString());
}
```

이미 대여중이여서 해당 함수를 호출한 경우, contract함수를 통해 추천날짜를 팝업창으로 띄워줍니다.

6) getRoomRentHistory 함수

```
const getRoomRentHistory = async () => {
  // 빈 배열을 만들고 주어진 헬퍼 함수 returnOptionsJSON 를 사용하여 선택된 방의 ID 값을 이용해 컨트랙트를 호출한다.
  const jsonobj = returnOptionsJSON();
  const roomId = jsonobj.id;
  let history = [];
  const contractHistory = await contract.methods.getRoomRentHistory(roomId).call();

  // 선택된 방에 대해 그동안 대여했던 사람들의 목록(히스토리)을 불러온다.
  for (let i = 0; i < contractHistory.length; i++){
    let historyObject = {};

    historyObject.id = contractHistory[i].id;
    historyObject.rId = contractHistory[i].rId;
    // 헬퍼 함수 dateFromDay 를 이용한다.
    historyObject.checkInDate = dateFromDay(currentYear, contractHistory[i].checkInDate).toDateString();
    historyObject.checkOutDate = dateFromDay(currentYear, contractHistory[i].checkOutDate).toDateString();
    historyObject.renter = contractHistory[i].renter;

    history.push(historyObject);
  }
  //console.log(history);
  return history;
}
```

contract 호출을 통해 rent 방을 object에 임시로 저장해둔 뒤, 각각의 object을 history 배열에 push하여 총 리스트를 만들어줍니다.