

**National College of Ireland**

Higher Diploma in Computing (Blockchain Stream)

2024/25

John Moloney

Student ID: x19215959

RentSure – Smart Contract Escrow System for Property Rentals

Final Technical Report

**Table of Contents**

[Executive Summary 3](#_Toc351559315)

[1 Introduction 3](#_Toc351559316)

[1.1 Background 3](#_Toc351559317)

[1.2 Aims 3](#_Toc351559318)

[1.3 Technologies 3](#_Toc351559319)

[1.4 Structure 3](#_Toc351559320)

[2 System 3](#_Toc351559321)

[2.1 Requirements 3](#_Toc351559322)

[2.1.1 Functional requirements 3](#_Toc351559323)

[2.1.2 Data requirements 3](#_Toc351559324)

[2.1.3 User requirements 3](#_Toc351559325)

[2.1.4 Environmental requirements 3](#_Toc351559326)

[2.1.5 Usability requirements 3](#_Toc351559327)

[2.2 Design and Architecture 3](#_Toc351559328)

[2.3 Implementation 3](#_Toc351559329)

[2.4 Testing 3](#_Toc351559330)

[2.5 Graphical User Interface (GUI) Layout 3](#_Toc351559331)

[2.6 Customer testing 3](#_Toc351559332)

[2.7 Evaluation 3](#_Toc351559333)

[3 Conclusions 3](#_Toc351559334)

[4 Further development or research 3](#_Toc351559335)

[5 References 3](#_Toc351559336)

[6 Appendix 3](#_Toc351559337)

[6.1 Project Proposal 3](#_Toc351559338)

[6.2 Project Plan 3](#_Toc351559339)

[6.3 Requirement Specification 3](#_Toc351559340)

[6.4 Monthly Journal 3](#_Toc351559341)

[6.5 Other Material Used 3](#_Toc351559342)

RentSure – Smart Contract Escrow System for Property Rentals

# Executive Summary

RentSure is a decentralized application (DApp) that uses smart contracts to make property rentals simpler and more secure. The traditional rental market often runs into problems like late payments, disputes over deposits, and trust issues between landlords and tenants. This project aims to tackle those issues by building an Ethereum-based smart contract system that manages rental payments, holds deposits in escrow, and automatically enforces penalties if rent is late.  
  
I built RentSure using Solidity for the smart contracts, Hardhat for local development and testing, and React with MetaMask to let landlords and tenants easily interact with the blockchain. Everything is deployed on the Sepolia Ethereum test network, so transactions work just like the real thing but without actual financial risk. I also set up tests using Chai and Mocha to make sure the contracts do what they’re supposed to.  
  
Overall, RentSure shows how blockchain can cut out middlemen, reduce disputes, and bring more transparency to renting. If developed further, this could include things like on-chain identity checks or connections to off-chain data for credit scoring. The project was managed through an iterative approach, with regular testing and updates to improve the solution step by step.

# 1. Introduction

## 1.1 Background

Renting property comes with a lot of hassle for both landlords and tenants. Landlords worry about getting paid on time and whether they’ll get their property back in good condition, while tenants often struggle to get their deposits back or have to deal with complicated manual agreements. These problems are common because the whole process relies heavily on trust, manual contracts, and banks, which aren’t always reliable or transparent.  
  
Blockchain has the potential to change this. With smart contracts, rental agreements can be automated and made tamper-proof, reducing disputes and removing the need for costly intermediaries.

## 1.2 Aims

The main goal of this project is to build a smart contract escrow system that:  
  
- Automates rental agreements on the blockchain.  
- Holds deposits securely and handles scheduled rent payments.  
- Enforces penalties if tenants miss payments.  
- Gives landlords and tenants a simple, secure way to view and manage their rental agreements.

## 1.3 Technologies

Here’s a quick overview of what I used to build this:  
  
- Solidity: To write the smart contracts that handle rental agreements, deposits, and penalties.  
- Ethereum (Sepolia testnet): To deploy and test the contracts in a real blockchain environment without spending actual money.  
- Hardhat: For developing, testing, and deploying the contracts locally before pushing them to the testnet.  
- React & JavaScript: To build the front end where users can connect their MetaMask wallets and interact with the contracts.  
- MetaMask: So users can sign transactions and manage their crypto wallets directly from their browsers.  
- Chai & Mocha: To write tests that make sure the smart contracts work as expected.

## 1.4 Structure

This report is split up like this:  
  
- Section 2 goes into the system design, requirements, architecture, implementation, testing, and how the user interface looks and works.  
- Section 3 covers what I achieved, what went well, and what could be better.  
- Section 4 talks about possible future improvements and ideas.  
- Section 5 lists all the references I used.  
- Section 6 (Appendix) has the original proposal, project plan, requirements, and anything else that supports the project.

# 2. System

## 2.1 Requirements

### 2.1.1 Functional requirements

- The system must let a landlord register a new rental agreement with a tenant by calling a smart contract function. This includes storing the deposit amount, monthly rent, lease duration, and penalty rules.  
- It must securely accept the initial deposit and hold it in escrow on the blockchain.  
- It should automatically trigger monthly rent payments from the tenant’s wallet to the landlord based on a timestamp.  
- If the tenant misses a payment by a set deadline, the system must apply a penalty by reducing the deposit or transferring a penalty amount to the landlord.  
- When the lease ends, the contract should return any remaining deposit to the tenant, minus penalties.  
- The front end must let landlords and tenants view contract status, upcoming payments, penalties, and transaction history through MetaMask.

### 2.1.2 Data requirements

- Store on-chain data like:  
 - Wallet addresses of landlord and tenant.  
 - Deposit amount.  
 - Monthly rent and lease duration.  
 - Timestamps for next payments.  
 - Total penalties deducted.  
- Minimal off-chain data; most info is handled directly by smart contracts. The front end just reads data from the blockchain.

### 2.1.3 User requirements

- Tenants and landlords must have an Ethereum wallet (MetaMask) set up in their browser.  
- Users must be comfortable approving transactions via MetaMask pop-ups.  
- They should have internet access and know how to navigate a simple web interface.

### 2.1.4 Environmental requirements

- The app must be accessible through a modern web browser (Chrome, Firefox) with the MetaMask extension installed.  
- It should be deployed on the Ethereum Sepolia testnet for this prototype stage, with potential to move to Ethereum mainnet or another chain later.  
- The local dev environment uses Hardhat for testing and deployment.

### 2.1.5 Usability requirements

- The UI should be clean and easy to use, with clear labels showing what each contract action does.  
- MetaMask transactions should have helpful messages so users understand what they’re approving.  
- The site should be responsive, working on laptops, tablets, and phones.  
- Error messages should be user-friendly, explaining if a transaction failed due to insufficient funds, wrong network, or user rejection.

## 2.2 Design and Architecture

The architecture for RentSure is pretty straightforward. It uses a React front end that talks directly to smart contracts on the Ethereum blockchain. Users interact with the app through MetaMask, which handles authentication and signing transactions.  
  
Front end: Built with React and JavaScript. It connects to the blockchain using the ethers.js library and MetaMask.  
Blockchain: The core logic runs in Solidity smart contracts deployed to the Sepolia test network. These contracts manage rental agreements, deposits, payments, and penalties.  
Local development & testing: Done using Hardhat, which lets me spin up a local blockchain for testing before deploying to Sepolia.  
  
I’ve also designed the contracts to be modular, so functions like registering a lease, paying rent, applying penalties, and ending the lease are all separate. This makes it easier to maintain and upgrade later.

## 2.3 Implementation

The smart contract is written in Solidity. It has functions like:  
  
- registerLease(): Sets up the agreement with all the terms.  
- payRent(): Lets the tenant pay their monthly rent, checks timestamps, and updates the next due date.  
- applyPenalty(): Automatically called if the tenant misses a payment deadline.  
- endLease(): Finalizes the contract and returns any leftover deposit.  
  
On the front end, I used React to build a simple dashboard for landlords and tenants. It shows details about the lease, next payment dates, and penalties. It also has buttons that call the smart contract functions, triggering MetaMask to sign and send the transactions.  
  
Hardhat scripts handle deployment to the Sepolia testnet. For testing, I used Chai and Mocha to make sure each contract function behaves the way it should.

## 2.4 Testing

I set up unit tests in Hardhat with Chai and Mocha to:  
  
- Check that leases register properly with the right parameters.  
- Make sure deposits are held correctly in escrow.  
- Verify that payRent() only works when called on or after the due date.  
- Confirm that penalties reduce the deposit or transfer funds when payments are late.  
- Ensure endLease() sends back the correct remaining deposit to the tenant.  
  
On top of that, I manually tested the UI by connecting different MetaMask accounts as landlord and tenant, running through the full process from registering a lease to ending it.

## 2.5 Graphical User Interface (GUI) Layout

The GUI is built with React and styled using plain CSS. It’s intentionally simple:  
  
- Home page: Lets you connect your MetaMask wallet. Shows your wallet address and ETH balance.  
- Dashboard: Displays contract details — landlord and tenant addresses, deposit held, monthly rent, next payment due, and penalties applied.  
- Actions: Buttons like “Pay Rent,” “End Lease,” or “Apply Penalty” (for landlord). When clicked, these trigger MetaMask to approve the transaction.  
  
I also tested the layout on mobile and tablet to make sure it stays readable and clickable.

## 2.6 Customer testing

Since this is a prototype, I ran informal “customer tests” with a few friends acting as landlords and tenants. I gave them instructions and watched how they used the app.  
  
Feedback was generally positive — they found it easy to understand what each button did, though a couple of people suggested clearer status messages after a transaction goes through. I took that on board and updated the UI to show success or failure messages after each action.

## 2.7 Evaluation

RentSure meets its main objectives. The smart contract correctly manages the escrow process, automates rent payments, and applies penalties without needing a middleman. The front end works well with MetaMask, so landlords and tenants can clearly see the state of their lease.  
  
Some improvements would be needed before using this in the real world, like adding oracles to pull in off-chain data (for example, real-world time confirmations) and doing full security audits on the contracts.  
  
Overall, this project shows how blockchain can simplify rental agreements by making them automatic, transparent, and hard to tamper with.

# 3. Conclusions

The goal of this project was to build a decentralized application that helps solve common issues in the rental market — mainly delayed payments, deposit disputes, and trust breakdowns between landlords and tenants. RentSure shows that a blockchain-based solution can handle these problems in a smarter, more automated way.  
  
By using Ethereum smart contracts, the app makes sure that deposits are held securely, rent payments are scheduled, and penalties are applied without needing a middleman. This not only helps reduce disputes but also brings more transparency to the process for both parties.  
  
I was able to design and implement a fully working prototype using Solidity, React, MetaMask, and Hardhat. The system works as planned: landlords can register leases, tenants can pay rent, and the contract takes care of everything in between, including penalties and returning deposits. I also got helpful feedback during testing, which led to better UI feedback messages and clearer status indicators.  
  
There are still things that could be improved or added, like better timekeeping accuracy using oracles, a proper on-chain dispute resolution feature, and support for different types of rental setups. But overall, I’m happy with how it turned out and I think it shows real potential to be useful if developed further.

# 4. Further Development or Research

While the core functionality of RentSure works well, there are several features I’d consider adding if I had more time or resources to continue development.  
  
First, one of the main areas for improvement would be to integrate oracles. At the moment, the smart contract uses block.timestamp to track rent due dates, but that’s not ideal for anything that needs precision or proof of real-world time. Using a trusted oracle service like Chainlink could help make the rent schedule more accurate and reliable, especially for automated payments and penalties.  
  
Another useful addition would be a dispute resolution mechanism. Right now, everything is automated — which is great for reducing manual errors and conflicts — but there’s no way to handle edge cases or disagreements. A possible solution would be to include a third-party arbitration option, or build in a system for both sides to submit claims or evidence if there’s a problem.  
  
I’d also like to explore identity verification, especially for a real-world version of the app. This could involve verifying users with a decentralized identity system (DID) or linking to government ID checks off-chain, just to make the system safer and reduce fraud.  
  
Lastly, if I was planning to release this as a real product, I’d work on multi-chain compatibility or even layer 2 integration (like Polygon or Arbitrum) to reduce gas costs and improve performance.  
  
Overall, this project showed me the value of smart contracts for automating real-life agreements. There's plenty of potential to keep building on top of this foundation and make it suitable for production use.

# 5. References

* Ethereum Foundation (2024) Ethereum developer documentation. Available at: https://ethereum.org/en/developers/docs/
* OpenZeppelin (2025) OpenZeppelin Contracts Wizard. Available at: https://wizard.openzeppelin.com/
* MetaMask (2025) MetaMask documentation. Available at: https://docs.metamask.io/
* Hardhat (2025) Ethereum development environment for professionals. Available at: https://hardhat.org/
* Chainlink (2025) Chainlink Oracles. Available at: https://chain.link/
* Mocha (2025) Mocha - JavaScript test framework. Available at: https://mochajs.org/
* Chai (2025) Chai assertion library. Available at: https://www.chaijs.com/
* React (2025) React – A JavaScript library for building user interfaces. Available at: https://reactjs.org/
* Ethers.js (2025) Ethers.js - Ethereum wallet and DApp interaction library. Available at: https://docs.ethers.org/
* Remix IDE (2025) Remix - Solidity IDE. Available at: https://remix.ethereum.org/

# 6. Appendix

This section includes supporting documents and additional materials that were part of the RentSure project. These help give a fuller picture of the planning, development, and testing process.

## 6.1 Project Proposal

The original proposal outlines the motivation behind RentSure, the goals, and the plan to use blockchain technology to automate rental payments and manage escrow in a secure, transparent way. It covers the background problem in the rental market, my objectives, the tools I planned to use, and the ethical considerations of using simulated data and testnet deployments.

## 6.2 Project Plan

The project was managed using a Gantt chart broken into phases:  
- Research and planning  
- Smart contract development  
- Frontend integration  
- Testing (unit and UI)  
- Report writing and presentation  
  
The timeline was built to keep work steady and avoid last-minute issues. Most tasks were completed on time, with some extra time added to testing and bug-fixing.

## 6.3 Requirement Specification

This document lists out:  
- Functional requirements like registering leases, automating payments, handling penalties, and releasing deposits.  
- Non-functional requirements like responsiveness, usability, and compatibility with MetaMask and testnet environments.  
- It also includes a breakdown of user roles and expected system behaviour.

## 6.4 Monthly Journal

Throughout the course of the project, I kept a monthly journal noting progress, challenges, and adjustments:  
- Logged key decisions (e.g. switching to Hardhat for better testing support).  
- Reflected on testing feedback and updates to the UI.  
- Recorded any delays or issues (e.g. MetaMask RPC issues on Sepolia).

## 6.5 Other Material Used

This includes:  
- Screenshots of the smart contract deployments in Remix and transaction hashes on Sepolia testnet.  
- Testing output logs from Hardhat.  
- Code snippets and screenshots of the React UI.  
- Feedback notes from informal testers (landlord/tenant roles).  
- GitHub repository with version history (if applicable).

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.