Table of contents

[**Part A** 2](#_Toc201168600)

[Q1, How does decentralization in the HopeChain system help ensure transparency and prevent tampering of donation records? 2](#_Toc201168601)

[Q2, Describe how either Proof of Work or Proof of Stake could be used to validate donations on a blockchain network. 4](#_Toc201168602)

[Q3, Why I Chose Proof of Stake Over Proof of Work for HopeChain 6](#_Toc201168603)

[Q4, Transforming HopeChain, From Single Admin to Shared DecisionMaking 8](#_Toc201168604)

[**Part B** 11](#_Toc201168605)

[Q1, Why Ethereum is the Perfect Match for HopeChain 11](#_Toc201168606)

[Q2, Getting HopeChain Live, My Journey Through Remix IDE 15](#_Toc201168607)

[Q3, Two Big Security Holes in HopeChain (And How We Fix Them) 31](#_Toc201168608)

[Q4, How HopeChain Could Actually Fix Healthcare and Education 34](#_Toc201168609)

[**Part C** 38](#_Toc201168610)

[Q1, Three Major Threats That Could Destroy My HopeChain Donation App 38](#_Toc201168611)

[Q2, How to Fix the Donor Privacy Problem in My HopeChain App 41](#_Toc201168612)

[Q3, Legal and Regulatory Challenges for HopeChain in Jordan 44](#_Toc201168613)

[Q4, Learning from The DAO Hack, Making HopeChain Bulletproof 47](#_Toc201168614)

[**Part D** 50](#_Toc201168615)

[Q1, How HopeChain Keeps Donation Records Safe from Tampering 50](#_Toc201168616)

[**Q2, HopeChain Access Control, AdminOnly Withdrawals** 52](#_Toc201168617)

[Q3, Two Major Improvements I Needs to Make After the testing 54](#_Toc201168618)

[References 56](#_Toc201168619)

# Part A

# Q1, How does decentralization in the HopeChain system help ensure transparency and prevent tampering of donation records?

So before I built the HopeChain ont the blockchain, the donation tracking was honestly a big mess. I was dealing with spreadsheets and a very basic databases that only I could access it, which meant that the donors had to just trust me on it when I said that their money went to the right place. So there was zero public audit trail, so people literally had to take my word for it. Plus it was very super easy for me or anyone with the admin access to manipulate, delete, or "accidentally lose" the records. So if my server crashed or got hacked, everything and I mean it, it could disappear. The trust problem was real the donors basically had to have blind faith in me and that I wasn't pocketing their money or lying about where it went, which isn't exactly the foundation i want for a charity platform.

When I re designed this as an at the Ethereum DApp, I fundamentally changed about how the whole system would works. Instead of me or anyone controlling everything from my laptop or theirs , the system now runs on a thousands of Ethereum nodes worldwide, so I literally can't manipulate any of the data even if I wanted to. Everything gets recorded on the public Ethereum blockchain where anyone can verify the data without asking my permission or logging into a some admin panel. All the donation processing happens through the Solidity code that executes automatically with no human intervention and no "oops I forgot to record that donation." Once a donation hits the blockchain, it's there forever and I can't edit it, delete it, or pretend it never happened.

The real benefits I get from this are pretty significant. My getAllDonations() function lets anyone see every donation that's ever been made with no hiding, andno cherrypicking data, no "trust me bro" just raw, verifiable facts. Once I record a donation with donations[donationCount] = Donation(msg.sender, msg.value, block.timestamp), that data is locked in forever. Even I can't change it later. Donors don't need to trust me personally anymore because they just need to trust the Ethereum network's consensus mechanism, which has billions of dollars validating it every day. Way more reliable than trusting some random charity admin.

Since it runs on Ethereum, anyone anywhere can donate and verify records. There's no central server that goes down, no geographic restrictions, no "sorry, our payment processor doesn't work in your country." I can even prevent donation spam without any centralized moderation using require(block.timestamp lastDonationTime[msg.sender] > 5 seconds, "Wait 5s before donating again") the blockchain handles rate limiting automatically.

This actually matters for everyone involved. Donors can verify their donation actually happened and see exactly where their money went without having to trust me or any organization. Who it should reach get proof that funds are real and available, plus a transparent audit trail for their own accountability. Regulators have complete transaction history available 24/7 for auditing without needing to request records from anyone. For me, I don't have to be the "trusted middleman" anymore because the system's trustworthiness comes from the technology, not from people believing I'm a good person.

The technical reality is pretty straightforward. The traditional way meant data lived on my server where I had complete control, creating high risk of manipulation, loss, or downtime. The blockchain way means data lives on thousands of nodes where no one has complete control, creating extremely low risk of manipulation or loss. Instead of querying my private database, the frontend calls var donation = await contract.donations(i) to pull data directly from the blockchain, not from any system I control.

# Q2, Describe how either Proof of Work or Proof of Stake could be used to validate donations on a blockchain network.

so for me to making sure every donation is legit and can't get faked it is obviously super super important for any blockchainbased charity platform, but it's especially more important for the HopeChain since we're dealing with people's actual money that is going to help the others. and the way this works is through something is called a consensus mechanism, and in technical words that where thousands of distributed nodes all agree on what's actually happening on the blockchain at any given moment.

so here HopeChain runs on the Ethereum, because it made a huge change back in the 2022 when it switched from Proof of Work to Proof of Stake with the Ethereum 2.0 upgrade. Instead of using miners competing to solve the complex math problems like in the old system, so basicly ethereum now uses the validators who are chosen to verify the transactions based on how much ETH they've staked or locked up in the network. It's basically like putting up a security deposit and the more you stake, the more likely you are the one to be chosen to validate the transactions, but if you try to cheat in anyway, then you lose your staked money.

When someone makes a donation through my HopeChain frontend, here's what actually happens behind the scenes. First, the user clicks donate and my DApp creates a transaction with await “contract.donate({ value, ethers.parseEther(ethAmount) })”, which gets signed through MetaMask and sent to the Ethereum network. The transaction then gets broadcast to the Ethereum mempool, which is basically a waiting room for unconfirmed transactions. And next, the PoS system randomly selects a validator to include this donation transaction in the next block, but the selection is weighted by how much ETH are each validator has staked there. If the validator does their job correctly, they get rewarded with transaction fees plus some newly minted ETH.

And once the validator verifies that and adds the donation to the block, the smart contract permanently records it and that’s by using the “donations[donationCount] = Donation(block.timestamp, msg.sender, msg.value)” . and at this point, the transaction becomes irreversible and final. Also the best thing about PoS its that at its finall happens way faster than the old Proof of Work system and we're talking about confirmation that happens in just a few seconds rather than minutes, and it uses way less energy too.

The benefits of using the PoS for the HopeChain's donation system are honestly pretty impressive. Donors get confirmation of their donation in about 12 seconds, which makes the whole experience feel responsive and trustworthy. The energy efficiency is huge too since PoS uses something like 99% less power than Proof of Work, which aligns perfectly with the values of a charity platform that cares about environmental sustainability. So from a security perspective, the validators are economically very rewarded to play by the rules because if they try anything malicious or wrong, they would lose their staked ETH through a process called the "slashing."

And te system also handles the high transaction volumes really well, which is very crucial during the emergencies or the disaster at the relief campaigns when tons of people might be trying to donate at the same time. The validators work together to bundle the transactions into the new blocks, and the threat of losing their stake it keeps everyone honest. This creates a network that's both secure and a trustworthy, which is the exactly what I need for a global donation platform where people are trusting me with their money to help others. The whole PoS mechanism basically ensures that every donation in HopeChain is properly validated without relying on any single authority or institution.

# Q3, Why I Chose Proof of Stake Over Proof of Work for HopeChain

Any blockchain system needs some way to decide how new the blocks get added and how the transactions gets verified without a having a central authority calling the shots. and these are called the consensus mechanisms, and they're basically the foundation that keeps the whole network running and trustworthy. When I was building HopeChain, I had to think about which consensus mechanism would work best for a donation platform.

Ethereum, which is what HopeChain runs on, used to use Proof of Work like Bitcoin, but they switched to Proof of Stake in 2022 with the Ethereum 2.0 upgrade. i think many are glad they made that change because the PoS is way better suited for what I'm trying to do with the HopeChain. Instead of the miners competing to solve energyhungry puzzles like in the Proof of Work, here PoS determines who gets to validate the transactions based on how much ETH they've "staked" or locked up as a security deposit. It's basically putting your money where your mouth is if you want to validate transactions, you need to stake some ETH, and if you try to cheat, you lose that money through a process called "slashing."

The way PoS actually works is pretty straightforward. Users stake their ETH to become the validators, and when it's time to add a new of the blocks, one of them gets randomly selected to do the job. The more ETH you've staked, the higher your chances of being chosen to be it, but it's still random enough to keep things very fair. If the validators try to pull any wrong stuff, part of their staked ETH gets taken away, which creates a strong financial rewards to play by the rules. This whole system means that the Ethereum can confirm blocks every 12 seconds instead of the 10+ minutes you see with the Bitcoin's Proof of Work.

For HopeChain specifically, PoS is obviously the better choice over PoW for several reasons. First, the speed difference is huge when someone donates money to help disaster victims, they want to see that transaction go through immediately, not wait around for 10 minutes wondering if it worked. With PoS, I can give donors confirmation in just seconds, which makes the whole experience feel responsive and trustworthy. The energy efficiency is also a big deal because HopeChain is supposed to be helping people and making the world better, not burning through massive amounts of electricity like Bitcoin mining does.

From a practical view, PoS just makes more sense for a charity platform. that's because It can handle way more transactions ata a once, which is crucial during the emergencies when alots of people might be trying to donate at the same time. The gas fees tend to stay lower even when the network gets very busy, which means more of each donation actually goes to the intended cause instead of getting eaten up by a transaction costs. The security model is also really elegant validators lose their own money if they try to cheat, so there's this builtin accountability mechanism that doesn't require me or any admin to police the system.

The values alignment is probably the most important factor though. HopeChain is all about social and environmental responsibility, so using a consensus mechanism that wastes enormous amounts of energy would be completely hypocritical. PoS uses about 99.9% less energy than Proof of Work, which aligns perfectly with what a charity platform should stand for. It's more modern, more efficient, and more sustainable basically everything you'd want in a system designed to help make the world a better place.

# Q4, Transforming HopeChain, From Single Admin to Shared DecisionMaking

Right now, HopeChain has a pretty serious trust problem that I honestly didn't think about enough when I first built it. Only I, as the single admin, can check funds and make withdrawals, which means every donor has to put complete faith in my judgment and security practices. This is basically like giving one person the only key to a charity's entire bank account, and even if that person is completely honest, there are so many things that could go wrong. What happens if my wallet gets hacked? What if I lose my private key? What if I make a terrible decision about fund allocation? All the donated money is at risk, and there's no backup plan or oversight.

From a donor's perspective, this setup requires blind trust, which honestly defeats one of the main benefits of using blockchain for charitable giving. People can't verify that emergency withdrawals are actually for emergencies, or that large fund releases are going to legitimate causes. There's no transparency in the decisionmaking process, and donors are basically back to the old model of "just trust the charity to do the right thing." That's not what blockchain should be about.

The solution I'm thinking about is replacing my single admin wallet with the multisignature setup where the multiple trusted the stakeholders share control of the funds. And instead of me controlling everything alone, the donation funds would be managed by something like a 3of5 or 4of7 multisig wallet from all. And let's say we have a five trusted people maybe two senior staff members, two board members, and one of the community representative. So for any of the withdrawal to happen, at least three of these five people would need to digitally sign off on it. The blockchain automatically enforces this rule, so even if someone tries to hack the system, they can't access the funds without getting any of the majority approval.

This immediately solves the single point of the failure problem because if one person's wallet gets compromised, the attacker still can't steal anything since they need multiple of the signatures. If one signer is unavailable, the others can still authorize legitimate withdrawals. The system becomes way more resilient and secure because the attacks would need to simultaneously target a multiple independent people, which is exponentially more difficult and expensive.

But multisig wallets only handle the execution of decisions we also need a transparent way to make those decisions in the first place. This is where on chain voting becomes a really powerful. And the stakeholders could create the proposals directly on the blockchain for things like "Withdraw 15 ETH for hurricane relief supplies" or "Allocate 25 ETH for the school construction project.", and these proposals would also include a clear descriptions of what the money would be used for, who would receive it, and what are the outcomes we would expect.

Then designated stakeholders vote on these proposals during a set voting period. The voting could be structured differently for different types of decisions maybe simple majority for routine expenses under $5,000, but requiring supermajority approval for larger withdrawals. The brilliant part is that all voting happens transparently on the blockchain, so anyone can look up a proposal, see who voted for or against it, and verify that approved actions actually took place. It creates a permanent, unchangeable record of every decision.

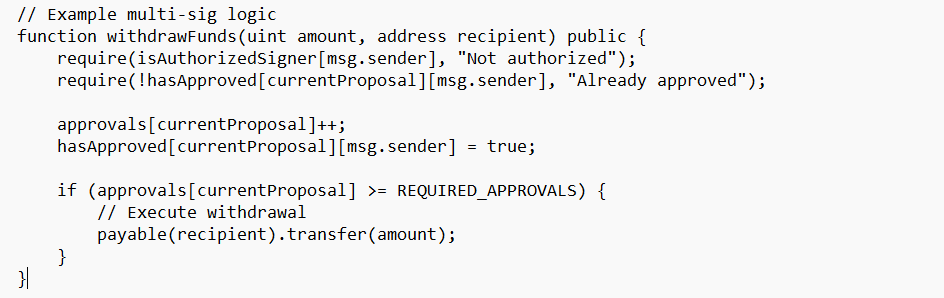
This dramatically improves the aHopeChain in a many ways. Instead of having all the security depend on the one person's wallet and judgment, risk gets distributed through across of the multiple stakeholders. And also the donors can finally see exactly how their money is being used instead of wondering whether the contributions are making a difference there in the world, and they can look up specific proposals, and see how the stakeholders has voted, and track the outcomes of the funded projects. This level of transparency builds the trust with the current donors and attracts new supporters who want accountability.

When multiple people have to agree on every major for aspending decision, it creates a natural checks and the balances. If someone proposes a questionable spending, the other stakeholders can ask tough questions, and demand more information, or vote it down. The system becomes self policing because everyone would knows their decisions are being watched and evaluated by the peers. Every vote, proposal, and withdrawal gets recorded permanently on the blockchain, so years from now, anyone can look back and see exactly how decisions were made and funds were used.

To make this more practical, a different types of decisions could require the different levels of the oversight. Like routine expenses under $1,000 might only need 2of5 signatures, while the medium withdrawals between $1,000$10,000 could require 3of5 signatures plus a public proposal period. And for the large withdrawals over $10,000 might need a 4of5 signatures plus a community voting period. And for the genuine emergencies needing an immediate funding, there could be a fasttrack process that requiring 3of5 signatures within a 24 hours, but with the mandatory post emergency community review and handle.

So after the core stakeholders, the broader community could participate through an advisable votes on the major fo the decisions, and the public comment periods before the final votes, and having a realtime transparency dashboards that’s showing the pending proposals. This shift from centralized to decentralized governance isn't just about making the HopeChain more secure it's about fundamentally changing how the charitable giving works. Instead of asking the donors to trust the individual organizations, here we're creating the systems where trust is built into the technology itself.

So when the donors can see exactly how much of their money is being used, participate in the allocation decisions, and verify that the funds is reach who it should reach who it should reach, then it creates a completely different relationship between the givers and the charitable causes. And this kind of radical transparency and the democratic governance that could rebuild the public trust in the charitable giving and also unlock much much higher levels of the social impact. The shared decisionmaking model also makes the HopeChain more legitimate and credible to the regulators, and major donors, and partner organizations and that’s because when the financial decisions are made transparently by the groups rather than the individuals, it carries a lot much more weight and reduces the concerns about the mismanagement or fraud. Ultimately, this transforms HopeChain from a "trust me" system into a "verify everything" system, which is exactly what blockchain technology should and would enable for the charitable be giving.



A computer screen shot of text

AI-generated content may be incorrect.

# Part B

# Q1, Why Ethereum is the Perfect Match for HopeChain

for the first is the smart contracts that it have, that's why its the biggest reason Ethereum works so well for the HopeChain is that it was specifically built to run the smart contracts. While Bitcoin is great for the simple transactions, Ethereum lets us write actual programs that live on the blockchain and getting execute automatically without anyone needing to manually oversee them.

so for the HopeChain, this is absolutely very crucial. and here thesmart contracts can automatically handle the donation processing, meaning keep the transparent records of every of the transaction, and manage the withdrawals according to pre determined rules. and when someone donates ETH to the contract, it will automatically updates the balances, and emits events that anyone can track, and also ensures the funds are properly allocated all without human interfear.

and the fact that the Ethereum uses the solidity as its programming language is another a huge advantage. the solidity has been around for good years at now, and it's mature and well documented, and there's a very massive community of the developers who know how to use it effectively. and this means when writing the HopeChain contract, means need to find solutions to the problems, and get help when im stuck, and build on proven patterns that other developers have already tested it.

The events system in the contract here like the Donated and the Withdrawn events, it creates a transparent audit trail that anyone can follow it. and this level of the automatic transparency and accountability would be really difficult to achieve on any other of the blockchain platforms.

Developer Tools That Actually Make Sense

a good thing was there in the ethereum was how a mature and userfriendly the development ecosystem is and was. When you're building it, it needs a reliable tools that helps write, test, and deploy the code very efficiently.

Remix IDE is probably the most beginnerfriendly blockchain development environment I've ever used. I can write Solidity code, compile it, test it, and deploy it all from my web browser. For someone just getting started with the blockchain development, this removes so many potential barriers and any of the technical headaches.

Ganache lets me run a complete Ethereum blockchain simulation on my local computer. This means I can test the HopeChain thoroughly without spending any real money on transaction fees or waiting for the network confirmations. I can simulate different scenarios, test edge cases, and also make sure that's everything works perfectly before the deploying to the real network.

the MetaMask merge makes it unbelievably easy for the users to interact with the HopeChain. Instead of having to set up a complex wallet software or to understand the technical details, the donors can just install a browser extension and start using the DApp immediately. The fact that the MetaMask is so widely adopted means our users probably already have it installed and makes it resier to us.

The JavaScript libraries like Web3.js and the Ethers.js create a seamless bridge between our frontend interface and the smart contracts. This means I can build a normallooking website that can talks to the blockchain behind the scenes, this means giving the users a familiar experience while leveraging all the benefits of the decentralization.

The Ethereum developer community is absolutely massive, and more importantly, it's incredibly helpful and very active. When I run into problems building the HopeChain, I can usually find solutions on the Stack Overflow, and GitHub, or Discord within minutes rather than hours or days.

There are thousands of the opensource projects that can learn from, and code libraries I can use to avoid reinventing the wheel, and also detailed tutorials that is covering almost every aspect of the DApp development. This community is driven by knowledge sharing and that makes the development so much faster and more reliable.

The security aspect is especially important for a donation platform like the HopeChain. The Ethereum community has years of the experience of identifying and fixing the smart contract vulnerabilities. and the security best practices would be well documented, and auditing tools are readily available, and there's a strong culture of the peer review that helps catch potential problems before they become real issues.

One of the things I really appreciate about choosing Ethereum is that it gives HopeChain room to evolve and add new features over time. The platform supports various token standards like ERC20 and ERC721 that could enable really interesting future functionality.

For example, we could eventually issue the NFT certificates to the donors as thankyou tokens, or create the governance tokens that let major contributors vote on how the funds should be allocated. We might want to integrate with DeFi protocols to earn yield on donated funds while they're waiti ng to be distributed, or connect with oracle services to automatically trigger donations based on realworld events like natural disasters.

All of these possibilities exist because the ethereum has standardized ways of handling the different types of digital assets and the interactions there. Building on Ethereum means we're not painting ourselves into a corner we're building on a platform that can grow with our vision.

I was honestly surprised by how easy it is to get a smart contract running on Ethereum. Using Remix with MetaMask's "Injected Web3" provider, I can deploy the HopeChain to the blockchain in just a few clicks. The process is unthinking of, this gives clear feedback about what's happening, and provides the transaction hashes I can use to verify everything worked correctly.

The ability to test on networks like Goerli or Sepolia before going live is invaluable. I can make sure that the HopeChain works perfectly in a real blockchain environment without risking real money or dealing with the mainnet transaction fees during the development.

The gas estimation features also help me optimize the contract code to minimize the transaction costs for users. This is particularly important for a donation platform where the high fees could discourage a smaller contributions.

Why This All Matters for HopeChain's

When building a platform designed to handle the people's charitable donations, it need technology that's proven, reliable, and a transparent. so here ethereum provides exactly that foundation.

The public ledger means that every donation and withdrawal is permanently recorded and verifiable. and the smart contract system ensures that the funds are handled according to the pre determined rules that can't be changed arbitrarily. also the mature development ecosystem means that it can build robust, secure code without the having to solve every problem from the scratch.

But maybe most importantly here that ethereum's widely spreading the adoption means that HopeChain can reach the broadest possible audience. and the donors don't need to learn about some mysterious blockchain or set up an unfamiliar wallet software, they can use the tools and the processes they already know.

and the fact that Ethereum is continues to evolve with the solutions like Layer 2 scaling means that HopeChain won't get left behind as the technology advances. We're building on a platform that's actively working to solve the challenges of main stream adoption while maintaining the core benefits of the decentralization.

Ethereum gives HopeChain the perfect combination of the technical capability, community support, and the futureproofing that a donation platform needs to succeed. It's not just about building something that works today it's about building something that can grow and adapt to serve the charitable causes for years to come.

# Q2, Getting HopeChain Live, My Journey Through Remix IDE

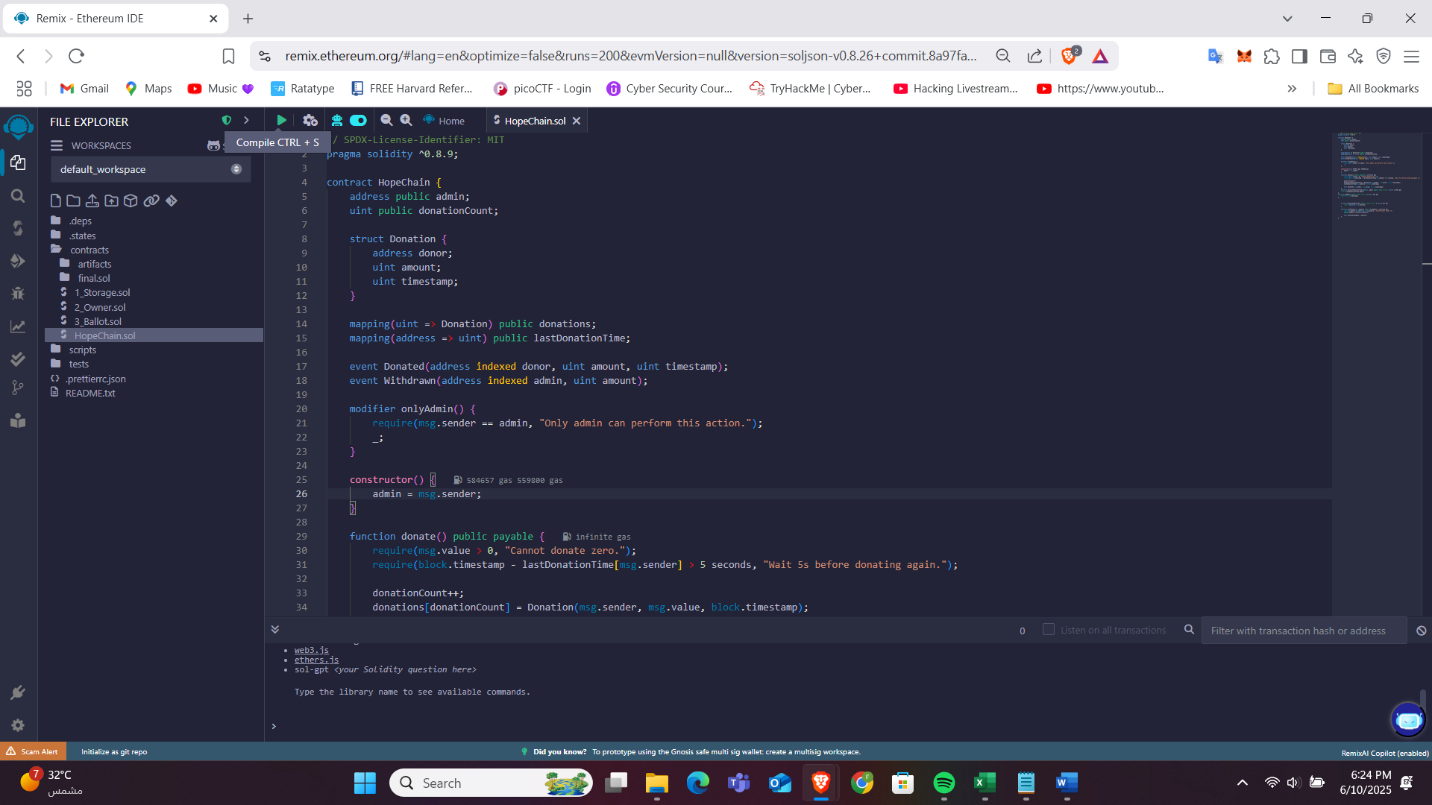
so deploying this was actually pretty straightforward

how I got the HopeChain from code to a smart contract running on the blockchain.



**Step 1, Setting Up the Remix**

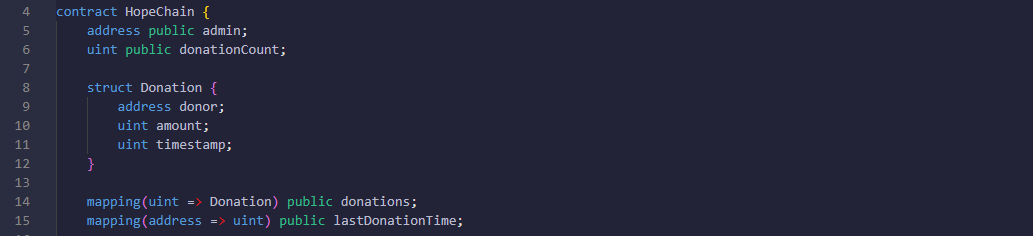
First things first I turned up Remix IDE, then I created a new file called HopeChain.sol in the workspace.



**Step 2, Writing the Contract**

This is where everything happens. I built out the HopeChain contract with all the important pieces

And this is the core Structure,



**The Key Functions in the code was, (will get to them in details later)**

donate() Lets people send money to the charity

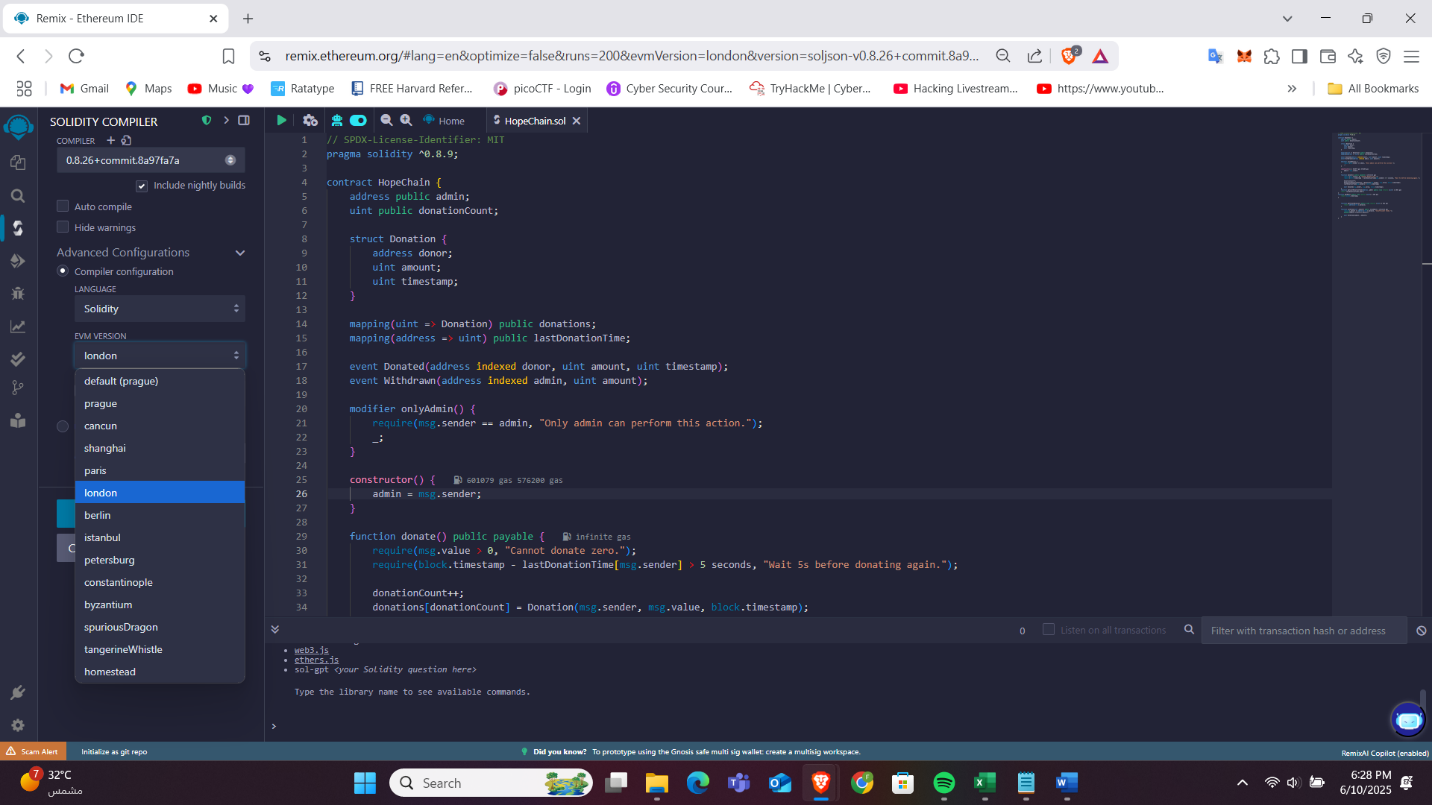
withdraw() Lets the admin take the money out (only the admin though)

getTotalBalance() Shows how much money is in the account

getLastDonationTime() Prevents people from spamming a donations

The greatest part was adding the onlyAdmin modifier it's like a bodyguard that checks if you're allowed to do admin stuff before letting you through in .

**Step 3, Compiling**

****

**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

Here. I switched to the Solidity Compiler tab and made sure I had the right version selected (0.8.9 need to match what I coded for).

clicked that Compile button and it worked after tries, Remix helped because is actually pretty good at telling exactly what's wrong if it messed up.

**Step 4, Deployment**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screenshot of a computer

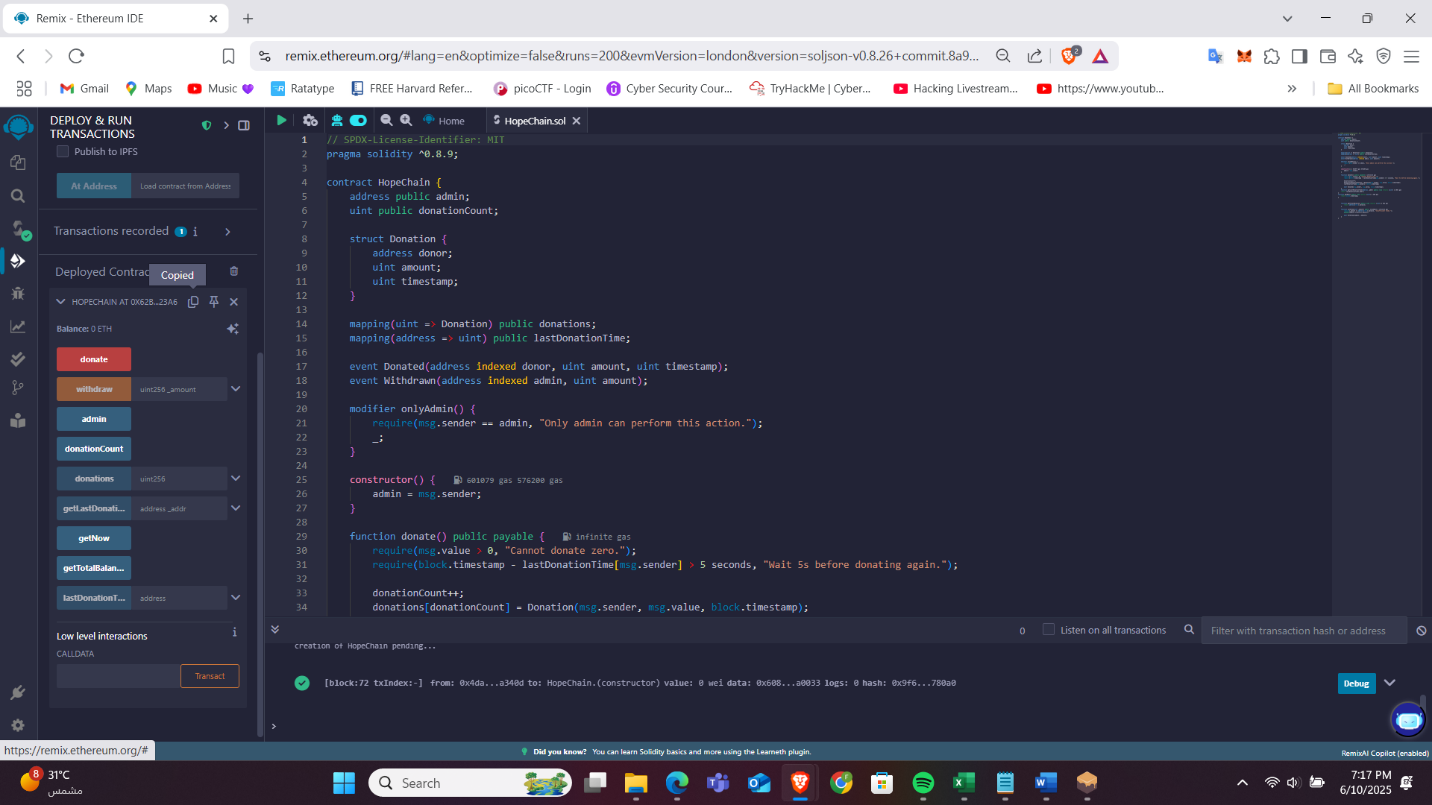
AI-generated content may be incorrect.**

**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

****

This is where it gets more good. I went to switched over to the Deploy & Run Transactions tab, I chose Injected Web3 which basically means "talk to my MetaMask wallet." This is what connects the Remix to the actual blockchain (or the testnet in my case), Selected the HopeChain from the contract dropdown, and Deploy, Here comes MetaMask asking me to confirm the transaction. Had to pay some gas fee, clicked confirm, and waited for the blockchain to finish.

A few seconds later, Remix showed me the contract address. This is like the permanent home address for my smart contract on the blockchain. And copied that for my html.

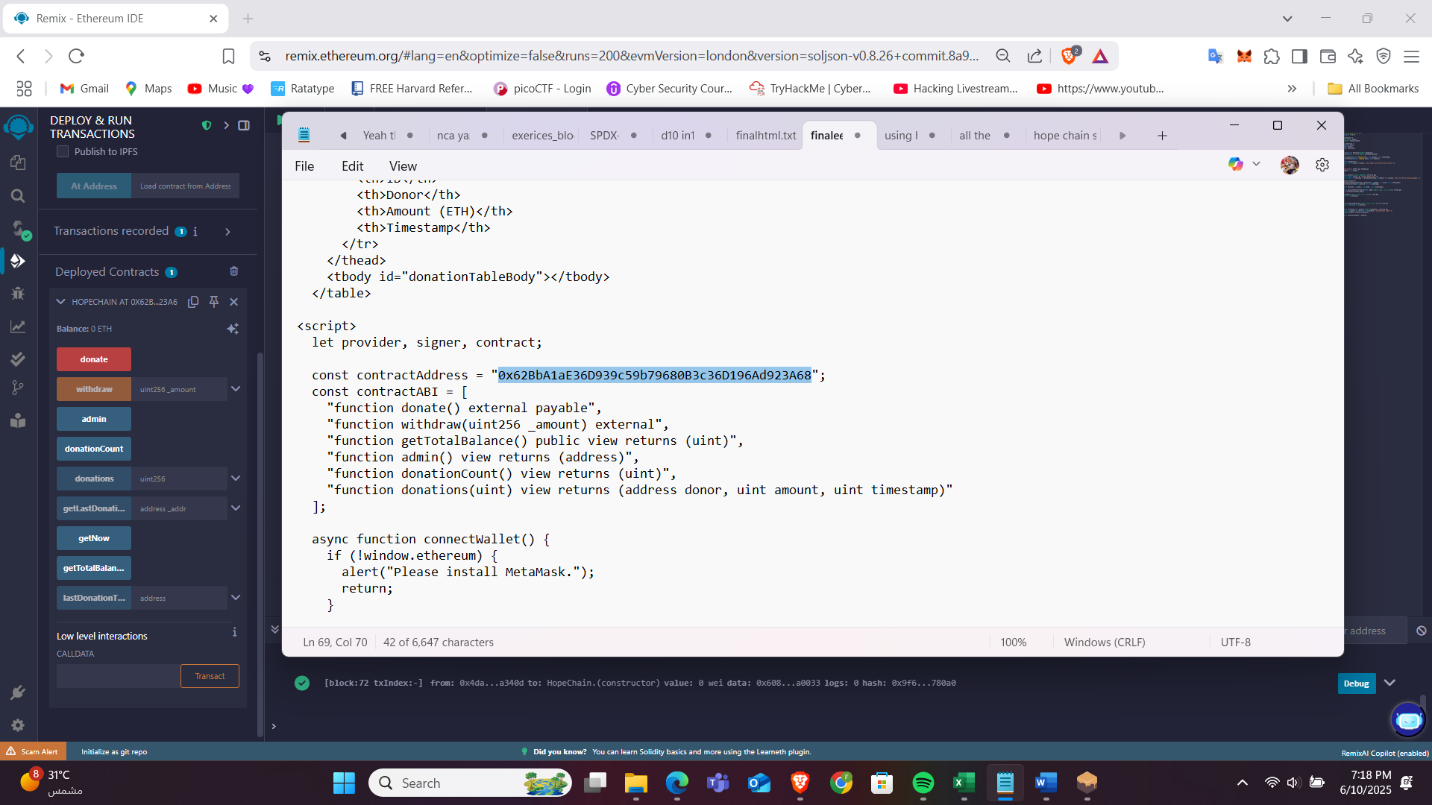
**Step 5, Testing Everything Actually Works**

**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

****

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

Now for the fun part need to be making sure this actually does what it's supposed to do.

I connected my frontend interface (HTML + JavaScript + ethers.js) to the deployed contract and tried sending some test donations. Watched my MetaMask balance go down and the contract balance go up. That was super satisfying it worked. And then tried to withdraw funds. It only worked when I was logged in as the admin address exactly as it intended to work. Also tested what happens when nonadmin users try to withdraw and it, and it fails with "Only admin can perform this action, as I needed

Then I got the web interface to show,

Current contract balance / List of all donations with timestamps / Last donation time for addresses / Admi only withdrawal interface

**Step 6, The Web Interface Magic**

**A screenshot of a computer

AI-generated content may be incorrect.**

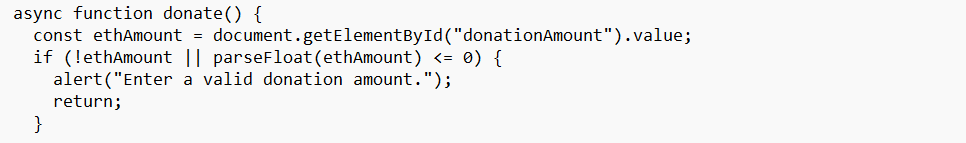
**A screenshot of a computer

AI-generated content may be incorrect.**

This is where it all comes together. Using ethers.js, I connected the smart contract to a clean web interface,





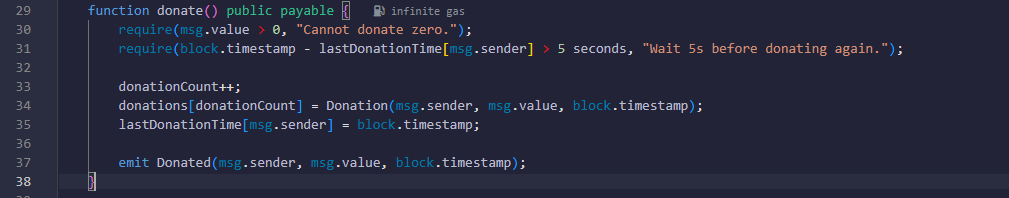




The best part was setting up the admin check the withdrawal button only shows up if you're the admin. Everyone else just sees the donation interface.

And for the Functions,

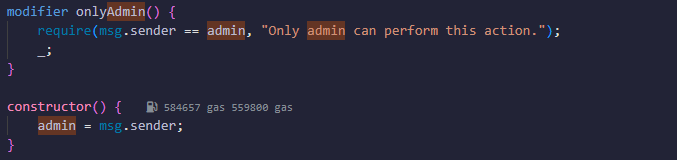
So when I was writing my HopeChain contract, I basically had to create a bunch of functions to handle all the donation stuff. Each function does something specific and I'll explain how I wrote them and what they actually do.

1- Donation Functionality  
The system should allow any of the users with a digital wallet to make a donation using a secure and userfriendly interface (the HTML) that is connected to the blockchain remix. and when a user donates, then the donation amount should be securely received and added to the total amount that is collected by the organization.  
So the main thing I needed was a function that the people that could actually use to send the money to the charity. I made the donate() function public and payable so that anyone with a wallet like the MetaMask can call it and send an ETH at the same time, 

The way this works is very straight forward and when someone clicks the donate button on my website, it triggers the MetaMask to pop up and ask them to confirm the transaction they made. and the msg.value is whatever of the amount of ETH they have decided to send, and msg.sender is their wallet address. and once they click confirm it, the ETH gets automatically transferred to the contract and stored securely on the blockchain.  
and the function is designed to work seam lessly with the web interface because I wanted it to be a userfriendly. Users don't need to understand smart contracts or anything complicated they just click donate, MetaMask handles the wallet stuff, they confirm the transaction, and boom their donation is processed. The ETH gets added to the contract's total balance automatically and safely.  
I made sure the interface is secure by using the builtin blockchain security features. Since everything happens onchain, the donations can't be intercepted or stolen during the transfer. And because the contract address never changes, people know exactly where their money is going.

2- AdminOnly Access to Funds

the only one have an authorized administrator representing the HopeChain just should be allowed to access and withdraw the funds gathered in the system. and this role must be clearly defined and securely managed.

I set up a secure admin system where only that the one specific person (whoever deployed the contract) can withdraw the donated funds. Here's how I made sure only the authorized admin can access the money in there, 

A computer screen shot of text

AI-generated content may be incorrect.

so when the contract gets deployed, the constructor() automatically sets the admin to whoever that deployed it (that's me representing the HopeChain here). and the admin address gets stored permanently and that can't be changed later, which is good because you don't want people fighting over who gets to control the funds.

The onlyAdmin modifier is like a security guard that checks every of the withdrawal request. It looks at the msg.sender (who's trying to make the withdrawal) and compares it to the stored admin address in there. and if they don't match exactly, the transaction will fails immediately with an error message. and as I said only the exact admin address can be successfully withdraw the funds.

The withdrawal function also has a safety checks built in. It verifies that there's an enough balance before trying to transfer anything using the require(\_amount <= address(this).balance). This prevents any errors and makes sure the admin can't accidentally try to withdraw more money than the contract actually has.

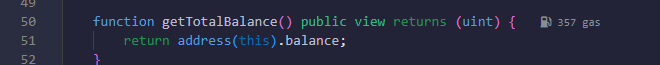
When a withdrawal happens, the function emits a Withdrawn event that gets recorded permanently on the blockchain. This provides transparency because everyone can see when money was withdrawn and with how much was taken out, and who took it and all.

The admin role is clearly defined and securely managed because the blockchain addresses are unique and can't be spoofed. When I deployed the contract, my wallet address got permanently recorded as the admin there, and only the transactions signed by that exact wallet can pass the security check. Its like having a digital signature that can't be forged.

3-Transparent Public , for Making Everything Visible

The platform must provide a transparent view of all the donation transactions, and allowing anyone to see a summary of who have donated, when, and how much. This fosters the transparency and builds the public trust in the system.

I made everything completely transparent and public so that anyone can see what's going on with the donations. and the main way I did this was by making the donation mapping and the other important variables public,



Since the donations mapping is a public, anyone can call it to see who donated, when they donated, and how much they have donated. They can just go donations(1) and see all the details for the first donation, or the donations(50) for the 50th donation, and so on for all.

The getTotalBalance() function lets anyone see how much total of the ETH is currently stored in the contract. This is super important for the transparency because donors want to know that their money is actually there and hasn't been stolen or misused. Since its a view function, it doesn't cost any gas to call it so people can check it as much as they want.

My web interface also has a "Show All Donations" button that loops through all the donation records and displays them in a nice table format. and this makes it easy for anyone to browse there through the complete donation history without they having to call te individual functions or know how to interact with the smart contracts directly.

The best thing is that since everything is on the blockchain, its completely transparent and verifiable by the design. Even if someone didn't trust the website or thought I was lying about something, they could check everything directly on the blockchain explorers like Etherscan. They could see every single of the transaction, every donation, every withdrawal everything is public and permanent.

This transparency is what builds trust in the system because people can verify for themselves that the money is being handled properly. and they don't have to just take my word for it they can check the blockchain and see exactly what's happening with their donations.

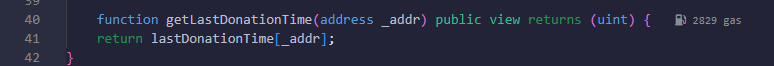
The system must include the basic protections to prevent the abuse, such as disallowing zer ovalue or an excessively frequent for the donations. Such the measures that help protect against the malicious use of the platform.

I built in two main types of protection to prevent people from the abusing of the donation system or trying to spam it with fake transactions,





4- The getLastDonationTime() Function AntiSpam Helper

I added this function mainly for the debugging and the testing for the spam prevention feature,

The first protection stops the zero value donations using the require(msg.value > 0). so this makes sure that the people can't send transactions with a zero ETH, which would be pointless and could clog up the system. Like why would someone even try to donate a zero money? It doesn't help the charity and it just wastes gas fees and the network resources.

The second protection is time based to prevent the rapidfire spam attacks. and i use a mapping called the lastDonationTime that tracks when each wallet address made their last donation. and before accepting any new donation, the function checks if at least 5 seconds have passed since their last one. If someone tries to donate too quickly less than that, the transaction fails with an error message will appear.

I chose 5 seconds as the bestdown period because its long enough to discourage any of the automated bots and the spam attacks, but short enough that the real users won't be annoyed if they want to make a multiple donations. Like if a someone realizes they didn't donate enough at the first time, they can just wait 5 seconds and then donate again without any major inconvenience.

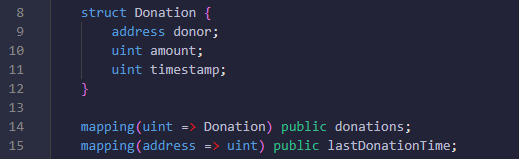
so both of these protections happen automatically at the smart contract level, which is really very important. Even if someone tried to bypass the web interface and call the contract directly through any other tools, they still couldn't spam the system because the blockchain itself will enforces these rules.

The spam prevention helps protect the platform from the malicious actors who might try to flood it with a fake transactions, and waste network resources, or disrupt the donation process for the legitimate users. and it keeps the donation system clean and focused on the real contributions to the HopeChain's mission instead of dealing with a garbage transactions.

so these measures also help maintain the integrity of the donations records because every recorded donation represents a real contribution rather than spam or test transactions that don't mean anything.

5- Automatic Donation Records that Keeping Track of Everything

so each donation must be automatically recorded on the blockchain. and this record should include the unique identifier (such as the sender's wallet address) and the donation amount. and the record should be safely stored and accessible for the verification.

Every single donation gets automatically recorded and stored forever on the blockchain. so i created a struct called Donation that captures all the important information there, 

So whenever someone donates, the donate() function will automatically creates a permanent record with a three pieces of info, first who donated (their wallet address), how much they have donated, and exactly when they donated it. The wallet address is the unique identifier there because every wallet has a different address that can't be faked.

and here the mapping donations acts like a database that is stores all the donation records with a unique ID numbers. like Donation 1 gets stored at the donations[1], donation 2 at the donations[2], and so on. and this data gets written directly to the blockchain during the transaction, so it can never be deleted, modified, or lost later because its saved.

and as for the donationCount variable, it keeps a track of how many is the total of the donations that we have been received, and it also serves as the ID for the next donation. So lets say if we have 100 donations already, then the next one will be the donation 101.

Since I made the mapping public, so the solidity automatically creates the getter functions that anyone can call for free. and people can check donations(1) to see the first donation details, donations(2) for the second one, and so on. and this makes the verification super easy because anyone can look up any donation by its ID number and verify all the details.

so the records are safely stored because they're on the blockchain, and which means they're distributed across all thousands of the computers around the world. and even if the website disappeared or something happened to the admin, all the donation records would still be there and can be reached forever.

# Q3, Two Big Security Holes in HopeChain (And How We Fix Them)

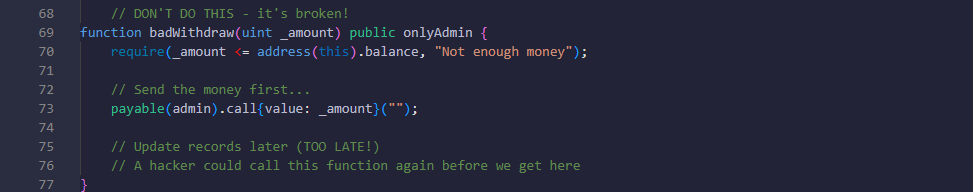
So I've been diving deep into HopeChain's security, and honestly? There are two ways this thing could get absolutely wrecked if we're not careful. Let me break down what could go wrong and how we make sure it doesn't.

**First problem is The Sneaky Reentrancy Attack**

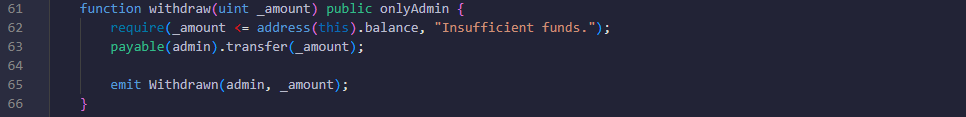
So imagine you're at an ATM in the bank, and you ask for a $100. Here the ATM will starts counting out the money, but before it finishes it, but somehow managed to press the withdrawal button again. And again. And again. And tricked it, and each time it starts giving another $100 because it hasn't finished updating the account balance yet.

That's basically what a reentrancy attack does to smart contracts. It's like the hacker found a way to spam the "withdraw" button before the contract realizes what's happening.

Here's what vulnerable code looks like,



An attacker could drain the entire contract by calling withdraw over and over before the contract realizes what's happening. This exact attack stole $60 million from "The DAO" back in 2016. Yep, sixty million dollars gone because of this bug.



The magic word here is transfer(). It only gives the receiving contract a tiny amount of gas (2300 units) not nearly enough to pull off any sneaky reentrancy tricks. If they try, their attack just fails.

**The second problem is The "All Your Eggs in One Basket" Problem**

The Single Admin, Right now, HopeChain has one admin. Just one. If that person,

If they Loses their private key → All donation money is locked forever

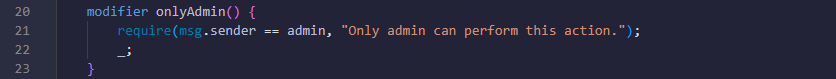
If they Gets their key stolen → Hacker drains everything

If they Goes rogue → They can steal all the funds

If they Gets hit by a bus → Funds are permanently inaccessible

This is what we call a "single point of failure," and it's terrifying when you're dealing with other people's charitable donations.

**The Current Protection (It's Something, But Not Great)**



This does prevent random people from withdrawing funds, which is good. But it doesn't solve the "what if the admin's key gets compromised" problem.

**The Better Way, Multiple Keys Required**

A screen shot of a computer

AI-generated content may be incorrect.

**How This Protects Us**

Multisignature requirements, will need 2 out of 3 from the authorized people to approve of any of the withdrawal. If one key gets stolen, the hacker still can't do anything alone with it.

Time delays for big withdrawals, Want to withdraw more than 10 ETH? You have to wait 24 hours. This gives everyone time to notice if something fishy is happening.

Recovery mechanisms, If the main admin loses their key, the backup admin can step in and fix things.

Emergency powers, If everything goes wrong, there's an emergency admin who can pause the contract or recover access.

These Fixes Matter because, it means hackers can't drain the contract through sneaky recursive calls. The attack that stole $60 million from The DAO simply won't work. and no single person can lose or steal all the donated funds. Even if one key gets compromised, the funds stay safe. Also Time delays give the community time to spot and respond to suspicious activity before large amounts of money disappear.

Recovery mechanisms mean the contract won't become permanently unusable if keys are lost.

So, smart contract security is hard. Really hard. Every few months, some new DeFi protocol gets hacked for millions because they missed some edge case or used a pattern that seemed safe but wasn't.

# Q4, How HopeChain Could Actually Fix Healthcare and Education

So we built this donation DApp for charity, right? But here's the thing the same tech that makes HopeChain work could completely revolutionize healthcare and education. Let me show you how the transparency and trust features we built could solve some massive realworld problems.

**Healthcare, Where Trust Goes to Die**

Healthcare is basically broken when it comes to trust. Patients don't trust doctors, doctors don't trust insurance companies, nobody trusts pharmaceutical companies, and everyone's keeping secrets from everyone else. Sound familiar? That's exactly the problem HopeChain solves for donations.

**Medical Research, Show Me the Money**

You know how our donation system tracks every penny? Imagine applying that to medical research funding,

**The Problem,** Big Pharma says they spent $2.8 billion developing on aa new drug. Did they really? Or did the most of that money go to executive bonuses and the marketing?

Instead of "trust us, this drug cost billions to develop," you get "here's the blockchain proof of every dollar spent on research, and here's exactly what results each dollar produced."

**Patient Data, You Own Your Medical Records**

Remember how our donation system prevents the admin from tampering with records? Same concept, but for your medical data,

The Problem, Your medical records are scattered across different hospitals, insurance companies, and clinics. You can't see them, you can't control them, and they get "lost" all the time.

Want to share your cancer treatment data with researchers? You get paid directly, and you can revoke access anytime. No more mysterious "data brokers" selling your health information without your knowledge.

**Drug Supply Chain, No More Fake Medicine**

Our adminonly withdrawal system could prevent counterfeit drugs,

Instead of one admin controlling withdrawals, require multiple healthcare authorities to approve each drug shipment. Can't release medications without verified temperature logs, handling records, and authenticity checks.

**Education, it has the Fake Degree Factory Problem**

Education has the same trust issues as healthcare, just different flavors.

Here the degrees would Actually Mean Something. , the Problem, that there's a $1 billion industry in fake degrees and certifications. Anyone can buy a "Harvard MBA" online for $200.

The HopeChain Solution, that is the universities issue degrees on blockchain, just like how we record donations,

Employers can instantly verify any degree, certificate, or skill claim. No more "my transcript got lost in the mail" or mysteriously impressive resumes.

**Crowdfunded Education**

As I sad that the HopeChain lets anyone donate to charity? Same thing would happen here, but for education,

Direct Student Funding, It can skip the banks and the loan companies. Community members can directly fund the students they believe in.

PerformanceBased Scholarships, so here smart contracts automatically release the funding when the students hit verified milestones. No bureaucracy, no paperwork, just "prove you're learning and get paid."

Teacher rewards, the communities can directly rewards the teachers whose students succeed. That means by pass the broken school beocken bureaucracy.

**Why This Could Actually Work**

The Trust Problem is Universal we would always have it no matter what, Every broken system we've talked about has the same core issue, information asymmetry. One party has information the other party needs, but they can't or won't share it honestly.

1. Healthcare, Doctors know more than patients, insurance companies hide coverage details, pharma companies obscure research costs
2. Education, Schools control transcripts, employers can't verify skills, students can't prove learning outcomes

Blockchain fixes this by making relevant information transparent and verifiable by default. And the beautiful thing is that most of the hard technical work is done. Here in HopeChain its already has a Transparent record keeping, Tamperproof data storage, Access control systems, Timestamp verification, and Multiparty approval processes

We just need to adapt these features for different use cases.

**The Real Challenges (Because It's Not That Simple)**

**Regulation Hell**

Healthcare, HIPAA says patient data must be private. Blockchain says data should be transparent. These two things don't play nice together.

Education, FERPA has similar privacy rules for student records. Plus, every country has different standards for what counts as a "real" degree.

**Legacy System Nightmare**

Hospitals and schools have spent billions on existing computer systems. They're not going to throw all that away to adopt blockchain tomorrow.

Most healthcare runs on systems that were built in the 1990s and held together with digital duct tape. Getting them to talk to blockchain is like getting a rotary phone to send text messages.

And as we know people hate changes big times, like At the Healthcare the Doctors don't want more transparency because it makes them accountable. Insurance companies profit from confusion and information asymmetry. And at the Education the Universities make money from their exclusive control over credentialing. They're not excited about systems that let anyone verify educational achievements. And for the rest, Most people barely understand email, let alone blockchain wallets and smart contracts.

**How to Actually Make This Happen**

For the first Pick one willing hospital or school and run a small pilot program. Show concrete benefits like reduced fraud, lower costs, or better outcomes.

Second, Get a few more institutions on board. Build network effects where the system becomes more valuable as more people use it.

Third, Regulatory agencies notice the benefits and start updating laws to accommodate blockchain systems.

make It Invisible to End Users, so the biggest mistake crypto people make is expecting everyone to become a blockchain expert. Normal people don't care about gas fees or private keys they just want systems that work better.

Build user interfaces that feel like normal websites and apps. Handle all the blockchain complexity behind the scenes.

So to Solve Real Problems, don’t build blockchain solutions looking for problems. Start with genuine pain points that existing systems can't solve, medical billing fraud costs $100+ billion annually, Credential fraud costs employers billions in bad hires, Research funding is often wasted on projects with no real results show how blockchain specifically solves these specific problems better than alternatives.

Part C

# Q1, Three Major Threats That Could Destroy My HopeChain Donation App

after investigation there are some serious security threats that could be mess up the whole system. These aren't just theoretical problems they're real vulnerabilities that attackers could actually exploit unfortunntley.

Threat 1, Sybil Attack via Multiple Wallet Addresses

The Problem, My spam prevention is basically useless against anyone who knows what they're doing.

Looking at my antispam code, 



I thought this was pretty clever when I wrote it, but it only protects against the donations from the same wallet address. The thing is, creating a new wallet addresses is super easy and free. Anyone can generate hundreds or thousands of a different addresses in like 5 minutes.

So a bad actor could just create 100 different wallets and donate from each one simultaneously, completely bypassing my 5second bestdown. Since each address has its own entry in the lastDonationTime mapping, there's nothing stopping rapidfire donations from multiple addresses controlled by the same person.

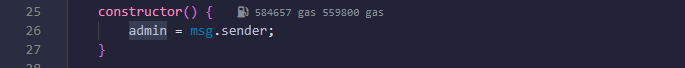
This could be used for all kinds of malicious stuff. Like someone could use it for money laundering taking dirty money and making it look like legitimate charitable donations by splitting it across hundreds of fake donation transactions. Or they could inflate the donation statistics to make HopeChain look way more popular than it actually is, which would be fraudulent.

Even worse, they could overwhelm the system with thousands of tiny donations that consume excessive gas fees and make the donation history completely unusable. Imagine trying to look through the donation list and seeing 10,000 donations of 0.001 ETH from obviously fake addresses. It would make the whole transparency feature pointless.

My current implementation provides zero protection against this attack vector because I was only thinking about preventing one person from spamming from one wallet, not about someone using multiple wallets. And the bad thing is that this vulnerability is visible to anyone who reads the contract code on the blockchain.

Threat 2, Permanent Admin Lockout Threat

The Problem here that if I lost the access to my wallet, everyone's donations are stuck there forever with no way to get them out.

My admin system has a important flaw, 



The admin address will gets set once when I just deploy the contract and there's no way to change it or transfer the admin privileges to a someone else. This will creates a single point of failure that could be big problem for everyone who have donated.

also what happens if I lose my private key? or forget my wallet password? Or my computer crashes and I don't have my seed phrase backed up properly? Or what if something happens to me personally like if I get sick or die? Then all the donated funds will become permanently locked in the contract forever.

Looking at the withdraw function, it has the onlyAdmin modifier, which means only the original deployer (and in this case its me) can ever access the funds in there. There's no backup admin, no multisignature requirement, and no emergency recovery mechanism, and no way to transfer ownership to someone else.

This is absolutely terrifying from a donor's perspective. People are trusting me with their money, but if I make one mistake with my wallet security or if something bad happens to me, their donations just disappear into the blockchain forever. No charity, no refunds, no nothing.

This violates the basic trust that donors place in charitable organizations. People donate because they want to help causes, not because they want their money to potentially get lost forever due to one person's technical mistakes. so the real charities have multiple people who can access the funds, proper governance structures, and the backup plans for these kinds of situations.

The fact that there's are no admin transfer functionality like transferOwnership() or a emergency recovery mechanisms means the contract is missing standard safety features that the production donation platforms always have.

Threat 3, Donor Identity Exposure in a Public Ledger

The Problem, that everyone's donation history is permanently visible to anyone who wants to look, which creates serious privacy risks.

My donation storage system records everything publicly, 



This means that anyone can look at the blockchain and see exactly who donated, when they donated, and how much they gave. The wallet addresses are supposed to be pseudonymous, but in practice they're often linked to real identities.

If someone's wallet address ever gets connected to their realworld identity like if they post it on social media, use it with an ENS name, or it gets leaked in a data breach then their entire donation history becomes publicly visible forever. People can track their giving behavior, see how much money they have, and figure out what causes they support.

This is way different from traditional charity donations where your information is kept private. With my syste m, if you donate to HopeChain and your identity gets revealed, everyone can see that you gave money and exactly how much. That's a huge privacy violation that most donors probably don't realize they're signing up for.

The privacy risks are really very serious. Wealthy donors could get targeted by the scammers who see they have the money and are willing to give it away. and people might face some social pressure or some criticism for the supporting of certain causes. In some countries, supporting certain charities that could even put them the people in some danger.

This threat is specific to the donation systems because the privacy is an extra important when it comes to the charitable giving. In a normal DApp, having a transparent transactions can be fine or even a good. But for the donations, the lack of the privacy could actually put off the people from giving or put them at risk.

for the problem is that the blockchain transparency, which is usually a good feature, becomes a bug when it comes to the donor privacy. My system needs privacypreserving techniques like zkSNARKs or stealth addresses, but I didn't implement any of that stuff because I didn't think about the privacy implications when I was building it.

These three threats show how my focus on basic functionality left some really serious security and privacy gaps that could undermine the whole purpose of the donation platform.

# Q2, How to Fix the Donor Privacy Problem in My HopeChain App

How to Fix the Donor Privacy Problem in My HopeChain App

so I have realized that my donation system exposes everyone's history to the public, I've been researching the ways to protect the donor privacy while still keeping the transparency that makes blockchain donations a trustworthy. There are basically three main approaches that could work with my HopeChain platform, each with different tradeoffs between privacy, complexity, and practicality.

The simplest solution would be using pseudonymous addresses, where donors would use fresh wallet addresses for each donation instead of reusing the same address over and over. Right now, my contract just records whatever address calls the donate() function, so instead of using their main wallet that might be linked to their identity, donors could generate new addresses from an HD wallet for each donation. So instead of seeing that one specific address donated 5 times, people would see 5 different addresses that each donated once. and the most modern wallets like MetaMask already support the HD wallets that can create a unlimited addresses from a one seed phrase, so I could add instructions on my website telling the donors to generate a new address before donating, or better yet, integrate with the wallet APIs that are automatically suggest using a fresh address for each one of the transactions.

The problem is that the blockchain analysis tools are getting a very really developed, so even if someone uses a different addresses,and patterns in timing, amounts, or transaction flows it can sometimes link them back to the same person. Like if someone always donates exactly 0.1 ETH every month, that's a really pretty obvious pattern even with a different addresses. Plus, if any of those addresses ever get linked to their real identity maybe they used one on an exchange that requires KYC, or posted it on social media then investigators might be able to connect the dots and figure out their other donation addresses too. But it's still way better than what I have now, and it's easy to implement, so it would be a good first step toward better privacy.

The hightech solution would be implementing a zeroknowledge proofs, which is where the things get really really interesting. and instead of storing the actual donor address and the amount, I could just use zero based knowledge proofs to let the people prove that they made a valid donations without revealing who they are or exactly how much they gave. For example, I could re design the donation system so that the donors prove statements like "I donated at least $50" without revealing if it was $50 or $5000, or "I'm eligible for the VIP donor status" without showing the donation history, or "This is a legitimate donation from a real person" without revealing identity.

I'd need to completely restructure the smart contract to work with the cryptographic proofs instead of the direct address storage. The donation process would look like, donor generates a cryptographic proof that they're making a good valid donation, then they submit the proof and the ETH to the contract, and the contract verifies the proof mathematically and accepts the donation, and the blockchain only records that "a valid donation of X amount occurred" without any of the identity info. The benefits are a very huge complete privacy for the donors while still maintaining the transparency about the total amounts, and the ability to implement sophisticated donor programs like the "gold member" status without exposing any of the people's giving history, and protection against both blockchain analysis and data breaches, and the futureproofing against quantum computing attacks with zkSTARKs.

The downside is that this stuff is really complex to implement it very correctly. it would need to learn advanced cryptography, and use specialized libraries, and thoroughly test everything to make sure that there are no other vulnerabilities. and the gas costs might also be higher for the generating and verifying the proofs, but the privacy benefits could be worth it for a the serious donation platform.

The intermediate solution would be using mixing services or privacy coins. Instead of having donors send ETH directly to my contract, they could first send it through a mixing service like the Tornado Cash to break the transaction trail. and the process would be, a donor deposits ETH into a mixing pool with a specific amount like 1 ETH, and then they wait for some time for other people to also use the mixer, they withdraw the same amount to a fresh address thats using a cryptographic proof, and then they donates to HopeChain from the fresh address. This way, even if s someone analyzes the blockchain, they can't trace the donation back to the original donor because the transaction path gets scrambled with everyone else who used the mixer.

I could also modify HopeChain to accept donations in privacy coins like Monero or Zcash instead of ETH. These coins have privacy built into their protocols Monero uses ring signatures to hide the sender, stealth addresses to hide the receiver, and RingCT to hide amounts, while Zcash offers shielded transactions where the sender, receiver, and amount are all cryptographically hidden. The main problem with mixing services is that some of them like Tornado Cash have faced legal issues because they can be used for money laundering, so I'd need to be really careful about compliance and make sure donors understand the legal implications. For privacy coins, I'd need to add support for multiple cryptocurrencies to my smart contract, which makes everything more complex, and I'd also need to handle the conversion between privacy coins and ETH if the charity needs funds in a specific currency.

One middleground approach would be to implementing a ring signatures directly in the smart contract. and i could create a "donor pools" where a multiple people can contribute to a ring signature, and then one anonymous member of the pool makes the actual donation. and the outsider observers would know that someone from the group has donated, but they couldn't tell who did it. This would provide a good privacy while keeping everything on Ethereum instead of the requiring external mixing services or any other cryptocurrencies. so each approach has its own complexity and the privacy tradeoffs, but any of them would be a huge improvement over the current system where every donation is completely public and traceable.

# Q3, Legal and Regulatory Challenges for HopeChain in Jordan

I thought the laws in jordan would be more straight tforward. But I was completely wrong. The regulatory landscape here is so so complex, and especially when dealing with both charitable donations and the cryptocurrency.

**The Charity Regulation Maze**

Charitable giving in Jordan isn't something you can just jump into. The government takes this stuff really seriously, which I actually appreciate from a donor protection standpoint, but it creates real headaches for innovative platforms like HopeChain.

The main law governing all this is Law 51 of 2008, updated in 2009, which requires any organization collecting charitable donations to register as a "Society" with the Ministry of Social Development. You have to submit your application at least a month before starting any fundraising activities, then wait up to three weeks for approval while they consult with other government ministries.

But here's where it gets complicated if you're dealing with international donations, which HopeChain almost certainly would be, you need separate approval from the Ministry of Planning and International Cooperation. They've required this since 2019, adding another whole layer of bureaucracy. If the money you're collecting will be spent outside Jordan, you have to set up a temporary committee specifically authorized by the Ministry just to handle that collection.

For HopeChain, this means we can't just launch and start accepting global donations. We'd either need to register as some kind of charitable technology platform or partner with an existing licensed NGO. Either way, we'd have to provide detailed records of where every donation comes from, and after collecting donations, deposit everything in registered bank accounts with full Ministry notification. So even though blockchain provides amazing transparency, the government still wants traditional financial reports in Arabic.

**Smart Contracts in Legal Limbo**

Here's where things get really interesting Jordan doesn't have any laws that explicitly recognize the smart contracts as legally enforceable agreements. my entire platform is built around the smart contracts that automatically enforce rules about donations and withdrawals, but if something goes wrong and we end up in court, a Jordanian judge might look at our smart contract and say "This is just computer code, not a legal contract."

That's scary when you're dealing with people's charitable donations. From a technical standpoint, our smart contracts work perfectly, but the legal system doesn't understand or acknowledge them. The solution seems to be creating hybrid legal structures with traditional written contracts that align with what the smart contracts do. We'd essentially maintain two parallel systems the automated blockchain system and traditional legal contracts that courts would recognize.

**Privacy and Transparency Conflicts**

So jordan has a very strict data privacy laws, and that’s including the Cybercrime Law from 2015 and the new ones, which requires consent before collecting any personal data and makes the organizations legally responsible for breaches. But the blockchain is inherently transparent and permanent. When someone donates to the HopeChain, their wallet address, and amount, and timestamp, and transaction details become part of the public blockchain record on forever.

Sure, wallet addresses are pseudonymous, but the regulators might not see that as a sufficient privacy protection. Jordan has also stepped up their antimoney laundering requirements they're no longer on international watch lists, but now they have stricter compliance requirements. Every financial transaction needs enhanced due diligence, suspicious activity reporting, and Know Your Customer procedures for all donors.

So we'd need robust transaction monitoring and donor verification systems while trying to maintain the decentralized, pseudonymous nature that makes blockchain appealing in the first place.

**The Cryptocurrency Problem**

This is probably the biggest challenge HopeChain would face in Jordan. The Central Bank has essentially banned all financial institutions from having anything to do with cryptocurrencies. Banks can't process crypto transactions, and I've read stories about people having their accounts frozen for days just because banks suspected they bought cryptocurrency.

The Central Bank has been clear that cryptocurrencies aren't legal tender in Jordan, and they've issued multiple warnings about fraud and volatility risks. This creates a huge problem for HopeChain because we'd naturally accept ETH donations directly. But if we did that, we'd probably get flagged by the banking system with almost no way to convert crypto donations into Jordanian Dinars for actual charitable use.

Cryptocurrencies aren't technically illegal in Jordan, but operating a cryptobased donation platform without access to traditional banking seems almost impossible practically.

Finding a Way Forward, so despite these challenges, can not give up on HopeChain to be working in Jordan. There are encouraging signs that the regulatory landscape might be developing. I've been reading about the efforts to legally recognize the cryptocurrencies as a digital assets, and there's even talk about Jordan launching a Central Bank about the digital Currency, means that would be suggesting to the government attitudes toward the digital financial technologies might be slowly changing.

I think the key is taking a collaboration approach from here to there rather than trying to launch and hope for the best. And we'd need to engage early with the regulators,means working with the Ministry of Social Development, and central Bank, and any of the other authorities to understand the developed regulations over time and maybe influenc the policy development.

And a hybrid approach might be most realistic using the blockchain for the transparency and recordkeeping while handling theactual money movement through traditional, a compliant banking relationships. The HopeChain could serve a primarily as a transparent tracking system, with an actual financial transactions happening through licensed institutions that provide necessary regulatory compliance.

This would mean starting with a pilot program focusing on the transparency features while using traditional payment methods for the money handling. As regulations evolve and authorities become more comfortable with blockchain technology, we could gradually expand cryptocurrency and smart contract features.

So a reality check**,** Looking at these challenges, I have to be honest launching HopeChain in Jordan would be complicated and expensive. The regulatory hurdles are substantial, and there's no guarantee the legal landscape will evolve quickly enough to make full blockchain functionality viable soon.

But I don't think these challenges are insurmountable. They're complex and require patience, legal expertise, and the willingness to work within existing systems while pushing for gradual change. What gives me hope is that Jordan seems genuinely interested in the financial innovation, even if they're approaching it cautiously.

The success of HopeChain in Jordan would depend more on our ability to work constructively with regulators than on pure technical capabilities. It would require treating regulatory compliance as a core feature and building relationships with initially skeptical authorities.

If we could pull it off, HopeChain could serve as a model for how blockchain innovation can work within traditional regulatory frameworks, showing that you can get blockchain benefits while meeting government oversight requirements. It's not the quick, disruptive launch some blockchain projects aim for, but it might be a more sustainable approach to bringing blockchain technology to charitable giving.

# Q4, Learning from The DAO Hack, Making HopeChain Bulletproof

When I was researching blockchain security for the HopeChain, I kept coming across references to The DAO hack from 2016. At first, I thought "that's ancient history in crypto terms," but then I realized this hack is basically the textbook example of what can go catastrophically wrong with the smart contracts. And honestly, it's exactly the kind of nightmare scenario about HopeChain.

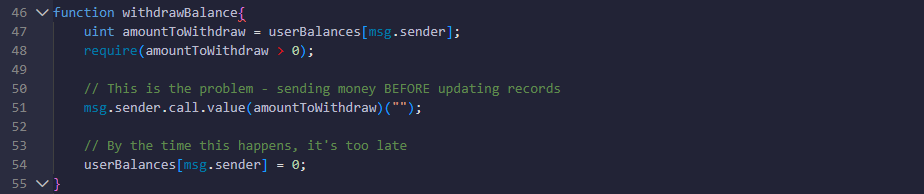
**What Actually Happened there**

The DAO was supposed to be this revolutionary decentralized investment fund. People poured in $150 million worth of ether, making it one of the biggest crowdfunding successes ever. Then, less than three months after launch, some hacker drained $60 million from it. Sixty million dollars. Gone. Just like that.

The part that I cant believe it happened that the vulnerability that enabled the hack was actually a very known issue and the DAO creators had even fixed this exact problem in an earlier version of the code, but somehow it made its way back into the final version with the lanching of it. It's like knowing your front door lock is broken, fixing it, then somehow ending up with the same broken lock again.

**The Reentrancy Attack**

Here's how the attack worked, and why it's so scary. The hacker found a function in The DAO's smart contract that would send ether to users when they requested withdrawals. The problem was in the order of the operations the contract would send the money first, then update its internal records to show the withdrawal had happened.



The hacker basically said "give me my money" and while the contract was sending it, they immediately said "give me my money" again. Since the contract hadn't updated its records yet, it thought they still had money to withdraw. And the hacker just kept doing this over and over in a loop, making it draining way more than they should have been able to do.

It wass like withdrawing money from an ATM that's so slow and it doesn't update the account balance until after it gives the cash. Here it could theoretically keep hitting "withdraw $100" and the machine would keep giving the money because it still thinks the account having the full balance.

What This Means for HopeChain?Looking at this hack made me realize I needed to completely rethink HopeChain's security. The current code isn't terrible, but it's not nearly robust enough for handling real people's donations.

The good news is that HopeChain doesn't have the same reentrancy vulnerability as The DAO. I'm using transfer() instead of call(), which only gives the receiving address a tiny amount of gas not enough to call back into the contract. But that's just one attack vector, and I learned from The DAO that you can't just fix one problem and call it secure.

**Security Improvements I'm Adding**

Even though the current code should be safe, I'm adding explicit reentrancy protection. It's like wearing a belt and suspenders maybe overkill, but better safe than sorry.

One of the most frustrating things about The DAO hack was that there was no way to stop it once it started. People knew what was happening, but the contract just kept bleeding money. I'm adding pause functionality so if something weird starts happening, we can stop everything while we figure out what's going on.

DAO had too much control concentrated in one place. For HopeChain, I'm requiring multiple signatures for large withdrawals. It's like requiring two keys to launch a nuclear missile no single person should be able to drain the entire donation pool.

Large withdrawals will have mandatory waiting periods. This gives the community time to notice if something suspicious is happening. Small withdrawals can happen immediately, but if someone tries to pull out a huge amount, there's a 24hour delay where other people can review and potentially stop it.

I'm adding much more detailed logging and event tracking. The goal is to make it obvious when unusual activity is happening, rather than discovering problems after the fact.

So the DAO hack taught the entire blockchain industry that tge smart contracts are basically permanent programs handling the real money, and can't just patch them later if found bugs. Once that code is deployed, it's out there forever, and any vulnerability becomes a permanent attack vector.

What really gets me is that this wasn't some complex one, zeroday exploit that no one ever could have predicted. The vulnerability was known, and the fix was known, and somehow it still made it into the production code. It's a sobering reminder that even when thinking that you've covered all the bases, then it probably haven't.

For HopeChain, this means I need to be absolutely scred about security. Every function, every state change, every interaction with external contracts needs to be designed with the assumption that someone is actively trying to exploit it. It's not enough to make the code work it has to work securely under attack conditions.

The DAO hack also led to Ethereum itself being controversially hardforked to undo the damage, which created a permanent split in the community. That's not an option for HopeChain if something goes wrong, there's no magical undo button. The security has to be right from day one.

I'll be honest, studying The DAO hack has made me more cautious about launching HopeChain, but also more confident that we can build something truly secure. The blockchain industry has learned a lot from this disaster, and there are now established patterns and tools for avoiding these kinds of vulnerabilities.

The key insight is that security isn't something you add to a smart contract it has to be built into the fundamental architecture from the beginning. Every design decision needs to consider potential attack vectors, and every line of code needs to be written with security as the primary concern.

It's definitely more work than just building a basic donation contract, but after seeing what happened to The DAO, I'd rather spend the extra time and effort upfront than watch HopeChain become another cautionary tale in blockchain security.

# Part D

# Q1, How HopeChain Keeps Donation Records Safe from Tampering

When I tell people that HopeChain donation records can't be tampered with, they usually give me this skeptical look like "come on, everything can be hacked." I get it we've all heard about data breaches and financial fraud. But blockchain protection is fundamentally different, and once you understand how it works, it's actually pretty amazing.

Digital Fingerprints That Can't Be Faked, Every donation in HopeChain gets turned into a unique digital fingerprint using SHA256 hashing. If I take your donation details and run them through this mathematical function, I get a completely unique string of characters that represents that exact transaction.

Here's the best part if someone tries to change even one small detail, like making a 100 donation into a 101, the entire hash will changes completely. It's not like it editing a document, the whole fingerpint transforms into something that is completely different. This means any tampering attempt is immediately obvious because the fingerprint won't match.

It has the Chain Reaction Effect, the really clever part is how these records are connected. Each block references the previous block's unique fingerprint. So if someone wanted to change the old donation record, they would need to have to recalculate not just that transaction there, but every single block that came after it. And they'd have to do this faster than the rest of the network is creatig new blocks.

It's like trying to edit a page in the middle of a book where every page references something specific from the previous page. You'd have to rewrite the entire rest of the book, and while you're doing that, thosands of other people are already writing new chapters.

Also, it has a NetworkWide Verification, the donation isn't stored on just one computer it's verified and stored on thousands of computers around the world. Before any donation gets permanently recorded, these computers have to agree that it's legitimate.

For someone to successfully tamper with records, they'd need to control more than half of these computers simultaneously. We're talking about hacking thousands of computers in different countries owned by different people. The cost would be astronomical, and if someone had that much computing power, they could make more money legitimately.

Another one is a Permanent Smart Contract Rules, Once HopeChain's code is deployed, it becomes permanent. I can't wake up one day and decide to change who can withdraw donations. The code is literally carved in digital stone. The withdrawal function will work exactly the same way forever no updates, no patches, no changing the rules.

The security isn't based on promises it's based on mathematics. Breaking the cryptographic protection would require more computational power than currently exists on Earth. Even if you could try a billion combinations every second, you'd need longer than the age of the universe to crack a single hash.

Unlike traditional systems where records are hidden in corporate databases, all HopeChain donations are publicly visible. Anyone can look up any donation and verify it actually happened. Thousands of people are essentially acting as auditors, making unauthorized changes nearly impossible to hide.

**RealWorld Protection**

Let me give a practical examples, If a hacker breaks into my laptop, they still can't change donation records because those aren't stored locally. If someone claims a donation wasn't made, the blockchain provides mathematical proof that it was. If I try to steal donations, the smart contract only allows legitimate withdrawals by the designated admin.

This Matters because, when you donate through HopeChain, you're not trusting me or any organization you're relying on mathematical certainty and a global network. Your donation becomes part of an unchangeable historical record that anyone can verify.

This is fundamentally different from traditional charity platforms where you have to trust that organizations won't manipulate their records or that their databases won't get hacked. With HopeChain, the integrity is guaranteed by the same mathematical principles that secure billions of dollars in cryptocurrency every day.

The protection isn't just a nice feature it's the entire foundation of why blockchain exists. Without thisthat guarantee, the whole systm would fall apart. That's what makes the HopeChain different, it creates an unbreakable record of charitable giving that no single person, organization, or government can manipulate. Your donation becomes part of history, permanently and verifiably.

# **Q2, HopeChain Access Control, AdminOnly Withdrawals**

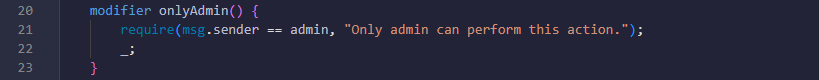
The key to the HopeChain's security is making sre only the designated admin can withdraw the funds and that’s also while keeping the donations open to everyone. And this is how this access control works





So when the contract is deployed, then whoever deploys it automatically becomes the admin. And their Ethereum address gets permanently stored, and this can never be changed.

Then this is the Access Control Gate



Here this modifier acts like a security guard. And before any of the protected function runs, it checks, "Is the person calling this function is the same person who deployed the contract?" and If it’s a yes, then it will proceed. If no, then reject the transaction completely.

Protected Withdrawal Function

A computer screen shot of text

AI-generated content may be incorrect.

So the onlyAdmin modifier runs before the withdrawal code. and only if that security check passes then does at the actual withdrawal happen.

**How It will Works in Practice**

Imagine an admin tries to withdraw, and the admin’s address, 0x1385735..., and the Contract's that is stored admin is, 0x1385735..., and it goes checking, 0x1385735... == 0x1385735... and its (right), then the result is (Withdrawal proceeds)

And another Scenario, but here we have a random user tries to withdraw and the User's address, 0x9876..., and the contract's stored admin, 0x1234..., and also the check, 0x9876... == 0x1234... (false), so the result is (Transaction immediately reverted, no funds transferred)

Why this Is secured? There are 2^160 possible Ethereum addresses (that's about 1.46 × 10^48 addresses). And only exactly one of those addresses can and will pass the admin check. And any Unauthorized attempts fail at the modifier level, and that’s before any withdrawal logic even runs. So the funds are never at risk. However, And there's no function to change the admin address, that would means no override mechanisms, and no emergency access. The rules are permanently locked in. but it has Transparency, means all the successful withdrawals are logged in the events that is there and anyone can see it on the blockchain.

Only the withdrawal function is restricted. Anyone can still donate, check balances, or view the transaction history. So even if someone tried to brute force attack the system, they'd need to guess the exact admin address out of 1.46 × 10^48 possibilities. That's computationally impossible. And any failed wthdrawal attempts cost very little gas because they're rejected immediately at the modifier level, but anyways at my website I disabled this function to be able to see it by only the admin account so they won’t cost anything by trying, and the access control logic is simple enough that anyone can verify it works correctly just by reading the code.

# Q3, Two Major Improvements I Needs to Make After the testing

So after actually testing my HopeChain Dapp code and web, I found some pretty glaring issues that I need to fix before this thing is ready for realworld use. Let me walk through the two biggest problems I discovered and how I plan to solve them.

**Enhancement 1, Emergency Admin Recovery (Because I Don't Want to Lock Up Everyone's Money Forever)**

The Problem I Found, So here's the thing I built this DApp with a single admin (me), and if I lose my private key, all the donated funds are basically gone forever. There's no transferOwnership() function, no recovery mechanism, nothing. It's a complete singlepointoffailure.

Why This is Actually Terrifying, Imagine someone donates $50,000 to help disaster victims, and then I accidentally delete my wallet or my laptop gets stolen. That money is just... gone. Forever. On the blockchain. Immutable. Oops.

My Solution, I need to implement a secure ownership transfer system

Plus maybe add an emergency multisig backup using a few trusted addresses that can recover the contract if I disappear.

This Matters, because Funds are never permanently lost due to my stupidity, and it makes the whole system way more reliable for real deployment, and nonprofits and donors expect this kind of basic continuity planning, also it shows I actually thought about longterm sustainability

**Enhancement 2, Stop People from Gaming the System with Fake Donations**

The Problem I Discovered, My current antispam protection is honestly pretty weak. I only check lastDonationTime[msg.sender] to prevent rapidfire donations from the same wallet within 5 seconds.

But here's the issue, Anyone can create hundreds of different wallet addresses and completely bypass this. They could make thousands of tiny donations to inflate the statistics, use the system for money laundering, and make the donation records completely meaningless

My Solution Basic Sybil Resistance

Plus maybe add more like ENS name verification for known donors, or frontend rate limiting by IP address, and donor reputation tracking over time

And again this matters because it prevents fake donation statistics that make the charity look more popular than it is, and stops system abuse and potential money laundering schemes, and it makes the data actually meaningful from real people not fake ones, not bots, and it be building trust with regulators and with the public, and it shows that Im really care about preventing fraud, not just be collecting donations.

# References

OpenZeppelin, 2024. *Smart Contract Security Guidelines*. [online] Available at: <https://docs.openzeppelin.com/contracts/4.x/> [Accessed 18 June 2025].

Rivest, R.L., Shamir, A. and Tauman, Y., 2001. How to leak a secret. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Berlin: Springer, pp.552-565.

Buterin, V., 2017. *Privacy on the Blockchain*. [blog] Ethereum Blog, 15 January. Available at: <https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain> [Accessed 18 June 2025].

Bonneau, J., Meckler, I., Rao, V. and Shapiro, E., 2020. Mina: Decentralized cryptocurrency at scale. *IACR Cryptology ePrint Archive*, 2020, p.352.

World Food Programme, 2020. *Building Blocks: Blockchain for Zero Hunger*. [online] Rome: WFP Innovation Accelerator. Available at: <https://innovation.wfp.org/project/building-blocks> [Accessed 18 June 2025].

Siegel, D., 2016. Understanding The DAO Attack. *CoinDesk*, [online] 25 June. Available at: <https://www.coindesk.com/markets/2016/06/25/understanding-the-dao-attack/> [Accessed 18 June 2025].

Anti-Money Laundering and Counter Terrorism Financing Unit, Jordan, 2007. *Law No. 46 of 2007*. [online] Amman: AMLU. Available at: <https://www.amlu.gov.jo/> [Accessed 18 June 2025].

Cosmos Network, 2019. *Cosmos Whitepaper: A Network of Distributed Ledgers*. [online] Available at: <https://cosmos.network/whitepaper> [Accessed 18 June 2025].

Central Bank of Jordan, 2017. *Warning Statement on Virtual Currencies*. [online] Amman: CBJ. Available at: <https://www.cbj.gov.jo/> [Accessed 18 June 2025].

Schär, F., 2021. Decentralized finance: On blockchain-and smart contract-based financial markets. *Federal Reserve Bank of St. Louis Review*, 103(2), pp.153-174.

UNICEF Innovation Fund, 2019. *Cryptocurrency Fund Annual Report*. [online] New York: UNICEF. Available at: <https://www.unicef.org/innovation/stories/cryptocurrency-fund> [Accessed 18 June 2025].

Ministry of Social Development, Jordan, 2008. *Societies Law No. 51 of 2008*. Amman: Official Gazette.

Ethereum Foundation, 2024. *Ethereum 2.0 Specification*. [online] Available at: <https://github.com/ethereum/consensus-specs> [Accessed 18 June 2025].

Douceur, J.R., 2002. The Sybil attack. In: *International workshop on peer-to-peer systems*. Berlin: Springer, pp.251-260.

Uniswap Labs, 2024. *Uniswap V3 Core Whitepaper*. [online] Available at: <https://uniswap.org/whitepaper-v3.pdf> [Accessed 18 June 2025].

Groth, J., 2016. On the size of pairing-based non-interactive arguments. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin: Springer, pp.305-326.

Zcash, 2023. *Sapling Protocol Specification*. [online] Available at: <https://zips.z.cash/protocol/protocol.pdf> [Accessed 18 June 2025].

Monero Research Lab, 2020. *Zero to Monero: Second Edition*. [online] Available at: <https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf> [Accessed 18 June 2025].

U.S. Department of Treasury, 2022. *U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash*. [press release] 8 August. Available at: <https://home.treasury.gov/news/press-releases/jy0916> [Accessed 18 June 2025].