# **1inch Hackathon: Extend Fusion+ to Near Action Plan**

**Timeline:** Tuesday, July 29th - Sunday, August 4th (Submission Deadline: Aug 3rd, but we'll use the 4th for buffer/submission)

### Primary Objective:

Create a bidirectional bridge for token swaps between Ethereum and NEAR, fulfilling all requirements: preserve hashlock/timelock functionality and demonstrate on-chain execution.

### Day 1: Tuesday, July 29th - Foundations & Setup

**Goal of the Day:** Establish a fully functional NEAR development environment and understand the core HTLC logic.

**Key Tasks:**

1. **Environment Setup:**
   * Install node.js and npm if you don't have them.
   * Install the NEAR CLI globally: npm install -g near-cli.
   * Create a NEAR Testnet Wallet: [wallet.testnet.near.org](https://wallet.testnet.near.org/). Fund it with testnet NEAR from the faucet.
   * Log in with the NEAR CLI: near login. This will connect your CLI to your testnet account.
2. **Project Scaffolding:**
   * Clone the official NEAR TypeScript smart contract template:  
     git clone https://github.com/near-examples/hello-near-ts.git 1inch-near-htlc  
     cd 1inch-near-htlc  
     npm install
   * Familiarize yourself with the project structure, especially src/contract.ts (where our logic will go) and src/model.ts (for data structures).
3. **Logic & Research:**
   * **Whiteboard the HTLC flow:** Draw out the exact steps for a swap from ETH to NEAR and from NEAR to ETH. Who creates the secret? Who creates the hash? Where are funds locked? When are they redeemed or refunded?
   * Read the [1inch Fusion+ Whitepaper](https://1inch.io/assets/1inch-fusion-plus.pdf) (Sections 3 & 4 are most relevant) to understand how they handle the EVM side of the swap.
   * Study a basic [HTLC implementation example](https://www.google.com/search?q=https://github.com/near-examples/htlc-contract/blob/master/src/contract.ts) in NEAR (if available) or even an Ethereum example in Solidity to understand the core functions (new\_swap, redeem, refund).

**✅ Success Metric:** You can successfully build and deploy the unmodified "hello-near-ts" contract to your testnet account.

### Day 2: Wednesday, July 30th - NEAR Contract Development (Part 1)

**Goal of the Day:** Implement the core data structures and the swap initiation logic on the NEAR contract.

**Key Tasks:**

1. **Define Data Structure (src/model.ts):**
   * Create a Swap class or object structure. It must contain:
     + sender: string (NEAR account)
     + receiver: string (NEAR account)
     + amount: bigint (using u128 for balance)
     + hashlock: string (the hash of the secret)
     + timelock: number (a Unix timestamp for the deadline)
     + status: 'Active' | 'Redeemed' | 'Refunded'
2. **Implement new\_swap function (src/contract.ts):**
   * This function must be payable to accept attached NEAR tokens.
   * It will take arguments: receiver: string, hashlock: string.
   * **Logic:**
     + Verify that attachedDeposit is greater than zero.
     + Calculate the timelock (e.g., env.block\_timestamp() + 1 hour).
     + Create a new Swap object with the details.
     + Store this swap in the contract's persistent storage (e.g., a PersistentMap or PersistentVector), using the hashlock as a unique key.
3. **Write Unit Tests:**
   * Create a test file (src/\_\_tests\_\_/main.spec.ts) to test the new\_swap function.
   * Simulate a user calling the function with the correct parameters and attached deposit.
   * Assert that a new swap is correctly stored in the contract's state.

**✅ Success Metric:** Unit tests for new\_swap pass. You can successfully call this function on-chain (via NEAR CLI) and lock funds in the contract.

### Day 3: Thursday, July 31st - NEAR Contract Development (Part 2)

**Goal of the Day:** Complete the NEAR smart contract by implementing the redeem and refund logic.

**Key Tasks:**

1. **Implement redeem function (src/contract.ts):**
   * It will take one argument: secret: string.
   * **Logic:**
     + Calculate the SHA-256 hash of the provided secret. This is the hashlock.
     + Look up the swap in storage using the calculated hashlock.
     + **CRITICAL CHECKS:**
       - Does the swap exist?
       - Is the swap status 'Active'?
       - Is the current block timestamp *before* the timelock?
     + If all checks pass, transfer the amount to the receiver's account.
     + Update the swap status to 'Redeemed'.
2. **Implement refund function (src/contract.ts):**
   * It will take one argument: hashlock: string.
   * **Logic:**
     + Look up the swap in storage using the hashlock.
     + **CRITICAL CHECKS:**
       - Does the swap exist?
       - Is the swap status 'Active'?
       - Is the current block timestamp *after* the timelock?
     + If all checks pass, transfer the amount back to the original sender.
     + Update the swap status to 'Refunded'.
3. **Write More Unit Tests:**
   * Test the redeem flow (both success and failure cases).
   * Test the refund flow (both success and failure cases).

**✅ Success Metric:** All unit tests pass. You have a fully functional and tested HTLC smart contract on NEAR.

### Day 4: Friday, August 1st - Off-Chain Script & Demo Prep

**Goal of the Day:** Create a script to demonstrate the full, bidirectional swap flow and start building the UI.

**Key Tasks:**

1. **Create an Off-Chain Script (Node.js/TypeScript):**
   * Use the near-api-js library.
   * This script will simulate the entire process for your demo video.
   * **Flow to script:**
     1. **Alice (User):** Generate a random secret. Hash it.
     2. **Alice on ETH (Simulated):** "I am locking 1 ETH with hash H." (Just a console.log statement).
     3. **Bob on NEAR (Script):** Call your contract's new\_swap function with the hash H and lock 10 NEAR.
     4. **Alice on NEAR (Script):** Call your contract's redeem function with the original secret to claim the 10 NEAR.
     5. **Bob on ETH (Simulated):** "I see the secret, now I'm claiming the 1 ETH." (console.log).
   * Also, script the refund scenario.
2. **Start UI Development (Stretch Goal):**
   * Use a simple framework like React/Vite or even plain HTML/JS.
   * Create a basic interface with:
     + A "Connect Wallet" button (using near-api-js wallet selector).
     + Input fields for Amount, Receiver, and Hashlock.
     + A button to execute the new\_swap.
     + An area to display active swaps for the user.

**✅ Success Metric:** Your off-chain script can successfully execute a new\_swap and a redeem call on the NEAR testnet, verified on the explorer.

### Day 5: Saturday, August 2nd - UI Polish & Video Recording

**Goal of the Day:** Finalize a simple, working UI and record a compelling demo video.

**Key Tasks:**

1. **Complete the UI:**
   * Add forms/buttons for the redeem and refund actions.
   * Make it look clean and presentable. This is a "stretch goal," but a good UI makes a huge difference in judging.
   * Deploy the UI to a free service like Vercel or Netlify.
2. **Prepare for Demo:**
   * Write a script for your video. It should be clear, concise, and under 3 minutes.
   * Explain the problem (cross-chain swaps).
   * Explain your solution (HTLC contract on NEAR).
   * **Walk through the live demo**, using your UI or script to interact with the testnet contract. Show the transactions on the NEAR Explorer.
3. **Record the Video:**
   * Use screen recording software (Loom, OBS).
   * Speak clearly and confidently.
   * Edit out any mistakes. Ensure the final video is high quality.

**✅ Success Metric:** You have a polished, sub-3-minute demo video that clearly shows your project working end-to-end.

### Day 6: Sunday, August 3rd - Submission Day

**Goal of the Day:** Submit a high-quality, complete project.

**Key Tasks:**

1. **Finalize GitHub Repository:**
   * Write a clear, detailed README.md. This is **EXTREMELY IMPORTANT**.
   * Include:
     + A short description of the project.
     + How it fulfills the hackathon requirements.
     + Instructions on how to build and test the contract.
     + A link to the deployed UI (if any).
     + A link to your demo video.
   * Ensure your commit history is clean and logical.
2. **Complete Submission Form:**
   * Fill out the ETHGlobal submission form carefully.
   * Double-check all links (GitHub, video, deployed app).
   * Write compelling descriptions.
3. **Submit & Relax:**
   * Submit well before the deadline to avoid last-minute issues.

**✅ Success Metric:** Your project is submitted with a great README, a compelling video, and a working on-chain demo.