1. **Technical Specifications:**
   * Outline the purpose and objectives of your dApp. (include why blockchain is good for this idea)
   * Describe the main features and functionalities it offers.
   * Provide a high-level architecture diagram of the system.
   * Explain the technologies used (Solidity, React, web3.js, etc.).
   * Detail any external APIs or services utilized.
2. **Process Flow / User Journey:**
   * Create a step-by-step user journey for interacting with the dApp.
   * Describe how users will navigate through the application.
   * Illustrate the process of executing various operations.
3. **Smart Contract Documentation:**
   * Provide a detailed explanation of each smart contract used.
   * Include descriptions of the contract's purpose and functions.
   * Document the contract's state variables and modifiers.
   * List the events emitted by the contract.
4. **Frontend Documentation:**
   * Describe the structure of the frontend components.
   * Explain how the frontend interacts with the smart contracts.
   * Include any external libraries or UI frameworks used.
5. **Deployment Instructions:**
   * Outline the steps to deploy the dApp to a test network (e.g., Ropsten).
   * Provide the necessary configuration details for deployment.
   * Include any pre-requisites for deployment (e.g., MetaMask setup).
6. **Testing and Quality Assurance:**
   * Document the testing approach for both smart contracts and frontend.
   * Describe how you ensured the dApp's functionality and security.
   * Include information about test cases and their outcomes.
7. **Challenges and Learnings:**
   * Discuss any challenges you faced during the development process.
   * Explain how you overcame those challenges.
   * Reflect on your learnings and improvements made.
8. **Code Repository Link:**
   * Share the link to the public code repository (e.g., GitHub).
   * Include clear instructions for running the project locally.

**Changes to the app made outside of the bootcamp:**

**Important**

* Allow the user creating the campaign to also define:
  + name & description of the campaign they are running (on smart contract) DONE
* Fix the number of approvers function so that it does not increase when the same address contributes to the campaign twice DONE
* Make it so that the manager & recipient cannot approve their own requests DONE
* Add a desired minimum balance on the campaigns. The estimated amount the project will need to succeed DONE
* Add a deadline on the campaign. DONE The time at which – if the desired minimum balance is not met – remainined $$ will be returned to contributors.DONE
  + E.g. if Alice gives $10 and Tom and Betty give $5 each…the total balance is $20. The desired balance of the project is $25. (The desired balance is all the money that is ever contributed…and does not factor in decreases due to requests) A request is made in the meantime for $7 that gets approved. Therefore the total balance is now $13. But all the money that was ever donated is still $20. The campaign does not meet it’s deadline. Therefore the remaining balance is given back to the contributors based on %. Alice gets $7.50 and Tom and Betty each get $3.75 back.
  + Create closed campaign page:
    - Campaign is now closed and did not meet it’s goal. Remaining balance is returned to contributors. Contributions, requests, and approvals are closed. Separate page for this (closed campaign) (you need to add front end validation to get the campaigns on each page as solidity cannot do that automatically…remove campaign disabled variable as it can’t actually link to the front end in real time)DONE
    - OR campaign stays on the open page and remains open if it meets its goal – just has a condition that says: Camapign met its goal, Contributions, requests, and approvals remain open.DONE
* Ensure min contribution is = or < than min balance in creating campaignDONE
* Style adjustments to the header – also include number of campaigns..$amount of ether donated..amount of projects that succeeded
* Include notes for all the different conditions of the requests.
  + Add scenario: Finalize should not be ready if the campaign balance does not meet the amount requested
  + Other scenarios should include a note on their existing conditions
* Add user guide/help text on how the requests for campign works
* Figure out why the application can only work with metamask and what users without metamask can do. If it only works with metamask, provider user instructions for downloading metamask IP – see chatGBT

**Nice to have**

* Allow the user creating the campaign to also define:
  + supporting documentation attachments, photo & video (on front end)
* Add a description of the recipient who the request $$ is going to
* Security so the methods on the smart contract that are restricted to the manager can only be seen by the manager in the styling of the web application
* Figure out why it takes so long to refresh each page 🡪 optimization
* Optional VIP tiers…contributors who donate up to $XX amount (defined by manager) will receive a reward. A function will automatically be available to VIPs when the contract meets the minimum desired balance. The VIPs can then be directed to an additional page that allows them to claim their digital reward.
* Create a feed of all contributors and their addresses . Including their stake % and weather they are VIP or not.
* Create a search bar and category tags for users to identify certain projects
* Include a page for ‘my campaigns’ or ‘my account’ to show all the campaigns you have donated to..or the campsigns you are running
* An actual deployment as a live site??