# What to build

1. Setup Github Repository and send the link to philipp@aeternity.com
2. Block Structure
    a. Header
        i. Previous block hash
        ii. Difficulty target (number of zeroes to be in beginning of hash)
        iii. Nonce (number to increase to get correct hash)
        iv. Chain state merkle root hash (can be ignored in beginning)
        v. Transactions merkle root hash (can be ignored in beginning)
    b. List of transactions
3. Transaction Structure
    a. From account public key
    b. To account public key
    c. Amount of transaction (integer)
    d. Elliptic curve signature for transaction, of “from” account
4. Public/Private Key Pair
    a. Generate new or derive public key from hardcoded private key
    b. Sign transactions data
5. Pool for transactions, newly ones created should be put there
6. Make a List of blocks, starting with a “empty/hardcoded” block
7. Build block candidate (wit block structure)
    a. Reference latest block hash
    b. Fixed difficulty target
    c. Nonce 0
    d. Generate a transaction to your generated public key with X tokens (coinbase), should have no sender and no signature
    e. Take all transactions from the pool and include them (remove from pool)
    f. Validate each transaction
        i. does it have a valid signature
        ii. does the “from”-account have the necessary amount
8. Proof of Work the candidate block
    a. Hash the block to see if target of 0 in the beginning is matched, if not increase nonce
    b. Repeat until the target is matched
9. Add newly generated block to list of blocks
    a. apply the transactions to the state of account balances (add received and subtract sent tokens), to check transaction spendings for the next block
    b. restart process of building candidate and mining

# Tools

Merkle trees library: https://github.com/aeternity/gb_merkle_trees

- Empty tree: `:gb_merkle_trees.empty()`
- Add to tree: `:gb_merkle_trees.enter(key, value, merkle_tree)`
- Get root hash: `:gb_merkle_trees.root_hash(merkle_tree)`
- Example: https://github.com/aeternity/elixir-research/blob/3b4d77bdccb49d76a9e4e6d4ca10e18ce581ea45/apps/aecore/lib/aecore/chain/block_validation.ex#L140

Erlang crypto library: http://erlang.org/doc/man/crypto.html

- Hash: `:crypto.hash(:sha256, data)`
- Generate private key:
  - `entropy_byte_size = 16` #16 bytes --> 128 bits
  - `:crypto.strong_rand_bytes(entropy_byte_size)`
- Private to public key:
  ```
  :crypto.generate_key(:ecdh, :secp256k1, private_key)
  ```
- Sign data with private key:
  ```
  :crypto.sign(:ecdsa, :sha256, data, [private_key,
                                       :secp256k1])
  ```
- Verify signature with public key:
  ```
  :crypto.verify(:ecdsa, :sha256, data, signature,
                 [pubic_key, :secp256k1])
  ```