BA-Chain

Max Hohlfeld & Martin Junghans

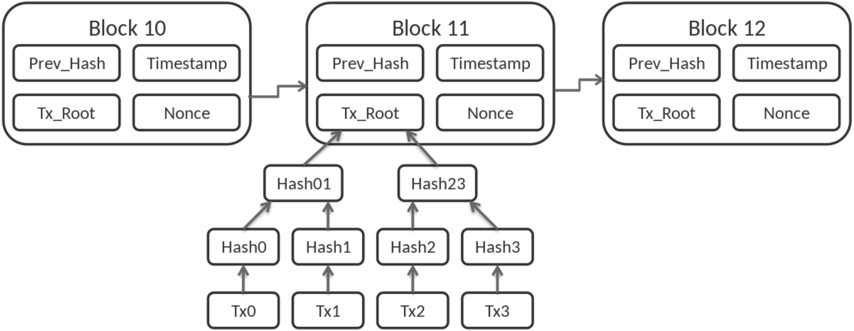
23.05.2022

Gliederung

Idee und Zielstellung

- Beispielanwendung für Verteiltes Rechen
- Aktuell populärer Anwendungsfall: **Kryptowährung**
- Entwicklung einer einfachen Blockchain à la Bitcoin
- Validierung der Blöcke durch Proof of Work
- Proof of Work im Verteilten Rechnen

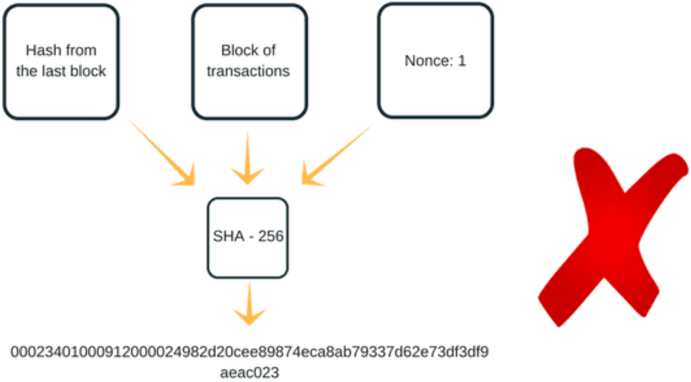
Konzept: Blockchain



---

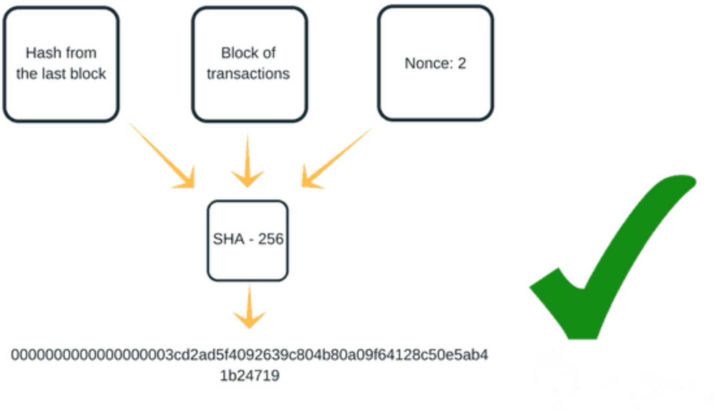
https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.svg

Konzept: Proof of Work



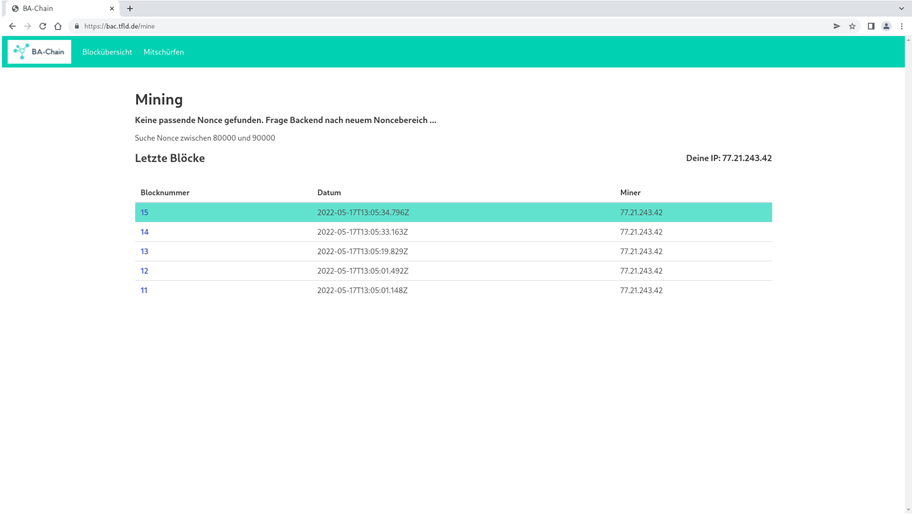
https://www.asynclabs.co/blog/blockchain-development/proof-of-work-what-it-is-and-how-does-it-work/

Konzept: Proof of Work



https://www.asynclabs.co/blog/blockchain-development/proof-of-work-what-it-is-and-how-does-it-work/

Demo



`https://bac.tfld.de`

Konzept: Funktionale Implementierung

1. Neue Transaktionen zufällig generieren
2. Transaktionen auswählen und zu einem Block zusammenfügen
3. Block und Schwierigkeit an Clients senden
4. Clients suchen Nonce um Schwierigkeit zu erfüllen
5. gefundene Nonce wird an Backend geschickt
6. Backend verifiziert Hash für erhaltene Nonce
7. wenn korrekt, wird Block gespeichert und beginnt Prozedur von vorn

Konzept: Funktionale Implementierung

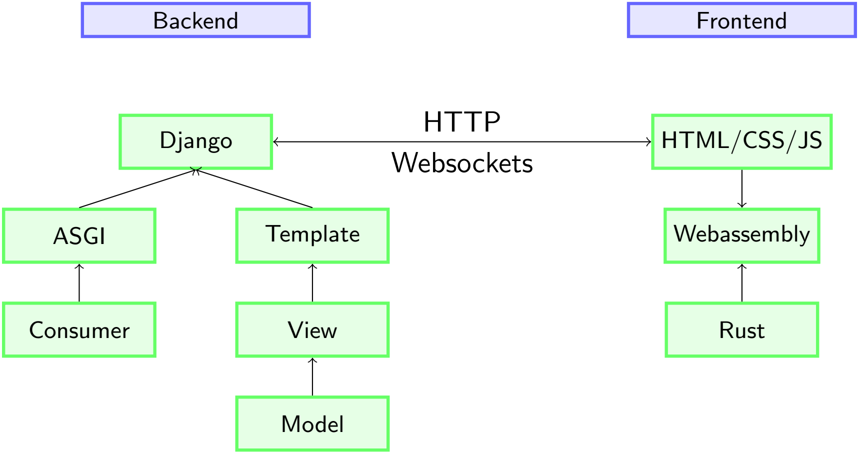
Vereinfachungen:

- Transaktionen werden automatisch generiert
- nächster Block wird durch den ersten Finder vorgegeben
- Schwierigkeit ist fix

gleiche Funktionsweise:

- Proof of Work Ansatz
- Aufbau der Blöcke

Konzept: Strukturelle Implementierung



Implementierung: Django - Model

```
class Block(models.Model):

    id = models.PositiveIntegerField(
        primary_key=True,
        verbose_name="Blocknummer",
        unique=True,
        null=False,
        editable=False,
    )
    acceptedAt = models.DateTimeField(
        verbose_name="verifiziert am", auto_now_add=True, null=True,
        ↪  editable=False
    )
    timestamp = models.DateTimeField(
        verbose_name="erstellt am", null=True, editable=False
    )
    transactionsIncluded = models.TextField(
        verbose_name="beinhaltete Transaktionen", null=True, editable=False
    )
    transactionsCount = models.PositiveIntegerField(
        verbose_name="Anzahl der Transaktionen", null=True, editable=False
    )
```

Implementierung: Django - View

```python
def coreUrl(request):
    if Block.objects.count() == 0:
        # generate genesis block mined by server
        fTG.transactionGeneration()
        fBG.blockGeneration()

        # generate next block mined by clients
        fTG.transactionGeneration()
        fBG.blockGeneration()

    context = fTCC.templateContextGeneration()
    return render(request, "coreLayout.html", context=context)


def blockUrl(request, value):
    context = fTCB.templateContextGeneration(value)
    return render(request, "blockLayout.html", context=context)

def miningUrl(request):
        return render(request, "miningTemplate.html")
```

Implementierung: Django - Template

```
<thead>
    <tr>
    <th>Blocknummer</th>
    <th>Datum</th>
    <th>Transaktionen</th>
    <th>Gesamtvolumen</th>
    <th>Miner</th>
    <th>Größe</th>
    </tr>
</thead>
<tbody>
    {% for element in blocks %}
        <tr>
          <th><a href="{% url 'single_block' element.id
↪   %}">{{element.id}}</a></th>
            <td>{{element.acceptedAt}} UTC</td>
            <td>{{element.transactionsCount}}</td>
            <td>{{element.transactionsValueTotal}} BAC</td>
            <td>{{element.miner}}</td>
            <td>{{element.size}}</td>
        </tr>
    {% endfor %}
<tbody/>
```

Implementierung: Django - Consumer I

```
class MiningConsumer(WebsocketConsumer):
    redis_instance = redis.StrictRedis(host=REDIS_HOST, port=REDIS_PORT, db=0)

    def connect(self):
        self.accept();
        async_to_sync(self.channel_layer.group_add)("mining",
        ↪   self.channel_name)

        newBlockJson = self.redis_instance.get("newBlock")
        newRange = CalculationRange(lowerBound=getNextLowerBound())
        newRange.save()

        lastBlock = getLastBlock()
        self.send(text_data=combineBlockJsonWithBound(newBlockJson,
        ↪   newRange.lowerBound, { "id": lastBlock.id, "timestamp":
        ↪   lastBlock.timestamp, "miner": lastBlock.miner }))

    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)("mining",
        ↪   self.channel_name)
        self.close()
```

Implementierung: Django - Consumer II

```python
def receive(self, text_data=None):
    message = json.loads(text_data);

    if message["op"] == "IP":
        self.redis_instance.set(name=self.channel_name, value=message["value"])
    elif message["op"] == "NEXT":
        range =
        ↪  CalculationRange.objects.filter(lowerBound=int(message["value"]))
        if range.count() == 1:
            range[0].status = CalculationRange.FINISHED
            range[0].save()

        newRange = CalculationRange(lowerBound=getNextLowerBound())
        newRange.save()

        self.send(text_data=str(newRange.lowerBound))
    elif message["op"] == "NONCE":
        nonce = message["value"]

        newBlockJson = self.redis_instance.get("newBlock")
        if newBlockJson is not None:
            block = json.loads(newBlockJson, object_hook=lambda d:
            ↪  PreBlock(**d))
```

Implementierung: Django - ASGI

```python
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'projectControl.settings')
from django.core.asgi import get_asgi_application
django_asgi_app = get_asgi_application()

from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
from django.urls import path
from core.consumers import MiningConsumer

application = ProtocolTypeRouter({
    "http": django_asgi_app,
    "websocket": AuthMiddlewareStack(
        URLRouter([
            path('mine', MiningConsumer.as_asgi())
        ])
    ),
})
```

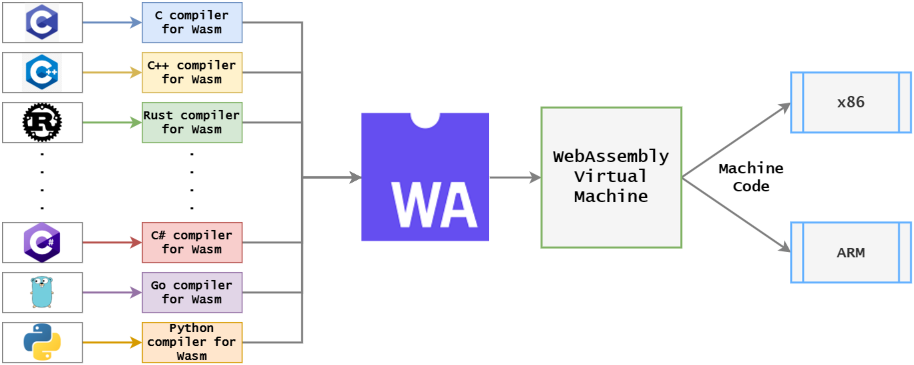
Implementierung: Frontend

- Django integrated HTML
- Bulma als CSS Bibliothek
- JavaScript nur, um WebAssembly aufzurufen:

```
<script type="module">
  import init, {get_ip, start_websocket} from "./static/pkg/calc.js";

  init().then(async () => {
    await get_ip();
    start_websocket();
  });
</script>
```

Implementierung: WebAssembly



https://arghya.xyz/articles/webassembly-wasm-wasi/

Implementierung: Rust - Nonce-Suche

```rust
fn calc(block: &BlockToVerify, lower_bound: u32) -> Option<u32> {
    let mut nonce: u32 = lower_bound.into();

    let mut s;
    let mut s_hashed;
    let mut s_hashed_hex;

    let challenge_pattern = "0".repeat(block.difficulty.try_into().unwrap());
    while nonce < (lower_bound + SIZE_OF_BOUNDS).into() {
        s = block.get_hashable_string(nonce);
        s_hashed = Sha256::new().chain_update(&s).finalize();
        s_hashed_hex = base16ct::lower::encode_string(&s_hashed);

        if s_hashed_hex.starts_with(&challenge_pattern) {
            return Some(nonce)
        }

        nonce = nonce + 1;
    }
    None
}
```

Implementierung: Rust - Websocket-Setup I

```rust
#[wasm_bindgen]
pub fn start_websocket() -> Result<(), JsValue> {
    console_error_panic_hook::set_once();
    let window = web_sys::window().expect("No window object found");
    let host = window.location().host().expect("No host field on object
↪   location found");
    let pathname = window.location().pathname().expect("No pathname field on
↪   object location found");

    let ws = WebSocket::new(&format!("wss://{}{}", host, pathname))?;
    let mut new_block = BlockToVerify { id: 0, timestamp: 0.,
↪   transactions_included: Vec::new(), transactions_count: 2, size: 2,
↪   prev_hash: "df".into(), difficulty: 0 };
    let mut last_blocks: VecDeque<InfoBlock> = VecDeque::new();

    let document = window.document().expect("No document object found");
    let status = document.get_element_by_id("status").unwrap();
    let lower_bound_span = document.get_element_by_id("lowerBound").unwrap();
    let upper_bound_span = document.get_element_by_id("upperBound").unwrap();

    let cloned_ws = ws.clone();
    let cloned_status = status.clone();
```

Implementierung: Rust - Websocket-Setup II

```
let onmessage_callback = Closure::wrap(Box::new(move |e: MessageEvent| {
    if let Ok(txt) = e.data().dyn_into::<js_sys::JsString>() {
        let mut lower_bound: u32 = 0;

        match txt {
            x if x.as_string().unwrap().starts_with("{") => {
                let packed_instructions: PackedInstructions =
↪   serde_json::from_str(&x.as_string().unwrap()).expect("Deserialization
↪   failed");
                new_block = packed_instructions.block.clone();

                if last_blocks.len() == 5 {
                    last_blocks.pop_back();
                }
                last_blocks.push_front(packed_instructions.last_block);
                display_last_blocks(&last_blocks);

                lower_bound = packed_instructions.lower_bound;
            },
            x if str::parse::<u32>(&x.as_string().unwrap()).is_ok() => {
                lower_bound =
↪   str::parse::<u32>(&x.as_string().unwrap()).unwrap();
            }
            x => console_log!("{}", x)
```

Implementierung: Rust - IP-Getter

```rust
#[wasm_bindgen]
pub async fn get_ip() -> Result<(), JsValue> {
    let request = Request::new_with_str("https://ip.tfld.de:50000")?;

    let window = web_sys::window().unwrap();
    let resp_value =
↪   JsFuture::from(window.fetch_with_request(&request)).await?;

    let resp: Response = resp_value.dyn_into().unwrap();
    let ip_value = JsFuture::from(resp.text()?).await?;

    window.document().unwrap().get_element_by_id("ip").unwrap()
        .set_text_content(Some(&format!("Deine IP: {}",
↪   ip_value.as_string().unwrap())));
    js_sys::Reflect::set(&window, &JsValue::from_str("ip"), &ip_value)?;

    Ok(())
}
```

Ausblick

- Verteiltes Rechnen allgegenwärtig
- umweltfreundliche Alternativen zu Proof of Work
- Digitale Währungen gewinnen an Einfluss

Vielen Dank!

Haben Sie noch Fragen?

`https://bac.tfld.de`

Martin Junghans & Max Hohlfeld