الكود الذي قدمته هو عبارة عن تنفيذ لبلوك تشين بسيط، وهو يتضمن عدة مفاهيم أساسية في البلوك تشين مثل الكتل، المعاملات، التعدين، وتبادل البيانات بين العقد عبر الشبكة باستخدام WebSockets. سأشرح لك فكرة الكود بشكل عام مع توضيح الأجزاء المسؤولة عن كل وظيفة.

1. فكرة البلوك تشين:

- **البلوك تشين** هو قاعدة بيانات موزعة تُخزن فيها المعلومات بشكل متسلسل وآمن باستخدام تقنية التشفير. تتكون "البلوك تشين من سلسلة من "الكتل (Blocks)، كل كتلة تحتوي على مجموعة من المعاملات.
- في هذا الكود، الكتل تحتوي على معاملات بين المستخدمين (مثل إرسال واستقبال العملات).

2. الجزء الأول: تعريف الكتلة (Block Class):

javascript

CopyEdit

```
class Block {
  constructor(timestamp, transactions, previousHash = '') {
    this.timestamp = timestamp;
    this.transactions = transactions;
    this.previousHash = previousHash;
    this.hash = this.calculateHash();
    this.nonce = 0;
  }
  calculateHash() {
    return crypto
      .createHash('sha256')
      .update(
        this.previousHash +
        this.timestamp +
        JSON.stringify(this.transactions) +
        this.nonce
      )
```

```
        .digest('hex');
    }
    mineBlock(difficulty) {
        while (
            this.hash.substring(0, difficulty) !== Array(difficulty + 1).join('0')
        ) {
            this.nonce++;
            this.hash = this.calculateHash();
        }
    }
}
```

المسؤوليات:

- **Block**: تمثل الكتلة التي تحتوي على مجموعة من المعاملات.
- **المعاملات**: (مثل إرسال أو استقبال العملات) هي العمليات التي يتم تنفيذها داخل الكتلة.
- **timestamp**: هو الوقت الذي تم فيه إنشاء الكتلة.
- **previousHash**: هو الـ hash الخاص بالكتلة السابقة، هذا يربط بين الكتل لتشكيل سلسلة.
- **calculateHash()**: تقوم بحساب الـ hash للكتلة باستخدام البيانات الموجودة في الكتلة (المعاملات، الوقت، الـ previousHash).
- **mineBlock()**: تقوم بتعدين الكتلة عن طريق إضافة nonce (رقم عشوائي) حتى يتم إيجاد قيمة hash تبدأ بعدد معين من الأصفار (تحديد مستوى الصعوبة).

3. الجزء الثاني: تعريف البلوك تشين (Blockchain Class):

```
javascript
CopyEdit
class Blockchain {
    constructor() {
        this.chain = [this.createGenesisBlock()];
        this.difficulty = 2;
```

```
    this.pendingTransactions = [];
    this.miningReward = 10;
  }
  createGenesisBlock() {
    return new Block(Date.now(), [], '0');
  }
  getLatestBlock() {
    return this.chain[this.chain.length - 1];
  }
  minePendingTransactions(miningRewardAddress) {
    const block = new Block(
      Date.now(),
      this.pendingTransactions,
      this.getLatestBlock().hash
    );
    block.mineBlock(this.difficulty);
    this.chain.push(block);
    this.pendingTransactions = [
      {
        from: 'network',
        to: miningRewardAddress,
        amount: this.miningReward,
      },
    ];
    return block;
  }
```

```
  addTransaction(transaction) {
    this.pendingTransactions.push(transaction);
  }
  getBalance(address) {
    let balance = 0;
    for (const block of this.chain) {
      for (const trans of block.transactions) {
        if (trans.from === address) {
          balance -= trans.amount;
        }
        if (trans.to === address) {
          balance += trans.amount;
        }
      }
    }
    return balance;
  }
}
```

المسؤوليات:

- **Blockchain**: تمثل سلسلة الكتل بأكملها.
- **createGenesisBlock()**: تقوم بإنشاء الكتلة الأولى (الكتلة الأساسية).
- **minePendingTransactions()**: تقوم بتعدين المعاملات المعلقة وتضاف كتلة جديدة تحتوي على تلك المعاملات.
- **addTransaction()**: تضيف معاملة جديدة إلى قائمة المعاملات المعلقة.
- **getBalance()**: تحسب الرصيد الإجمالي للمستخدم بناءً على المعاملات في السلسلة.

4. الجزء الثالث: WebSocket للخوادم الموزعة:

javascript

CopyEdit

```
const wss = new WebSocket.Server({ port: WS_PORT });
const sockets = [];

wss.on('connection', (socket) => {
  sockets.push(socket);
  console.log('New peer connected');
  socket.on('message', (message) => {
    const data = JSON.parse(message);
    if (data.type === 'NEW_BLOCK') {
      blockchain.chain.push(data.block);
      broadcast(message);
    }
  });
});

function broadcast(message) {
  sockets.forEach(socket => socket.send(message));
}
```

المسؤوليات:

- **WebSocket**: يتم استخدام WebSocket في الشبكة (العقد) لإنشاء اتصال دائم بين الخوادم.
- **broadcast()**: تقوم بنقل الرسائل إلى جميع العقد المتصلة.

5. الجزء الرابع: التوثيق واستخدام JWT (JSON Web Token):

javascript

CopyEdit

```
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.status(401).json({ error: 'No token provided' });
  jwt.verify(token, JWT_SECRET, (err, user) => {
    if (err) return res.status(403).json({ error: 'Invalid token' });
    req.user = user;
    next();
  });
};
```

المسؤوليات:

- **JWT**: يتم استخدام التوثيق باستخدام JWT للتحقق من هوية المستخدمين الذين يحاولون الوصول إلى بعض النقاط (مثل إجراء المعاملات أو التعدين).

6. الجزء الخامس: نقاط النهاية API:

توفر واجهات للتفاعل مع البلوك تشين (Routes) هذه النقاط:

- **تسجيل الدخول (login)**: يسمح للمستخدمين بتسجيل الدخول باستخدام بياناتهم الشخصية.
- **التحقق من الرصيد (balance)**: يعرض رصيد المستخدم من المعاملات.
- **إضافة معاملة (transaction)**: يسمح للمستخدمين بإرسال عملات إلى مستخدمين آخرين.
- **التعدين (mine)**: يسمح للمستخدمين بتعدين كتلة جديدة للحصول على مكافأة.

7. التشغيل والخوادم:

```
javascript
CopyEdit
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
  console.log(`WebSocket server running on port ${WS_PORT}`);
});
```

المسؤوليات:

- **app.listen()**: تقوم بتشغيل الخادم وتحديد المنفذ للاستماع.

الخلاصة:

الكود هو نموذج لتطبيق بلوك تشين بسيط يتيح للمستخدمين إرسال واستقبال العملات عبر معاملات مُدَارة باستخدام الكتل والتعدين، ويوفر واجهات API للتفاعل مع النظام مثل التحقق من الرصيد وإجراء المعاملات. يتم التعامل مع شبكات P2P باستخدام WebSockets لربط العقد في الشبكة.

4o mini