



4. Enable Web3j for Ethereum Spring Boot application

I think that now we have clarity on exactly what we want to do. So, let's proceed to the implementation. First, we should include all required dependencies in order to be able to use the **web3j** library inside the Spring Boot application. Fortunately, there is a starter that can be included.

```
<dependency>
  <groupId>org.web3j</groupId>
  <artifactId>web3j-spring-boot-starter</artifactId>
  <version>1.6.0</version>
</dependency>
```

Because we are running Ethereum Geth client on Docker container we need to change auto-configured client's address for **web3j**.

```
spring:
  application:
    name: transaction-service
server:
  port: ${PORT:8090}
web3j:
  client-address: http://192.168.99.100:8545
```

5. Building Ethereum Spring Boot applications

If we included **web3j** starter to the project dependencies all you need is to autowire **Web3j** bean. **Web3j** is responsible for sending transactions to Geth client node. It receives a response with transaction hash if it has been accepted by the node or error object if it has been rejected. While creating a transaction object it is important to set the gas limit to minimum 21000. If you send a lower value, you will probably receive error **Error: intrinsic gas too low**.

```

@Service
public class BlockchainService {

    private static final Logger LOGGER = LoggerFactory.getLogger(BlockchainService.class);

    @Autowired
    Web3j web3j;

    public BlockchainTransaction process(BlockchainTransaction trx) throws IOException {
        EthAccounts accounts = web3j.ethAccounts().send();
        EthGetTransactionCount transactionCount =
web3j.ethGetTransactionCount(accounts.getAccounts().get(trx.getFromId()),
DefaultBlockParameterName.LATEST).send();
        Transaction transaction =
Transaction.createEtherTransaction(accounts.getAccounts().get(trx.getFromId()),
transactionCount.getTransactionCount(), BigInteger.valueOf(trx.getValue()),
BigInteger.valueOf(21_000),
accounts.getAccounts().get(trx.getToId()), BigInteger.valueOf(trx.getValue()));
        EthSendTransaction response = web3j.ethSendTransaction(transaction).send();
        if (response.getError() != null) {
            trx.setAccepted(false);
            return trx;
        }
        trx.setAccepted(true);
        String txHash = response.getTransactionHash();
        LOGGER.info("Tx hash: {}", txHash);
        trx.setId(txHash);
        EthGetTransactionReceipt receipt = web3j.ethGetTransactionReceipt(txHash).send();
        if (receipt.getTransactionReceipt().isPresent()) {
            LOGGER.info("Tx receipt: {}",
receipt.getTransactionReceipt().get().getCumulativeGasUsed().intValue());
        }
        return trx;
    }
}

```

The `@Service` bean visible above is invoked by the controller. The implementation of POST method takes `BlockchainTransaction` object as parameter. You can send sender id, receiver id, and transaction amount. Sender and receiver ids are equivalent to index in query `eth.account[index]`.

```

@RestController
public class BlockchainController {

    @Autowired
    BlockchainService service;

    @PostMapping("/transaction")
    public BlockchainTransaction execute(@RequestBody BlockchainTransaction transaction) throws
NoSuchAlgorithmException, NoSuchProviderException, InvalidAlgorithmParameterException,
CipherException, IOException {
        return service.process(transaction);
    }
}

```

You can send a test transaction by calling POST method using the following command.

```

$ curl --header "Content-Type: application/json" --request POST --data
'{"fromId":2,"toId":1,"value":3}' http://localhost:8090/transaction

```

Before sending any transactions you should also unlock the sender account.

```

> personal.unlockAccount(eth.accounts[1])
Unlock account 0x2480be05af06376a9c2de9b1f98088b8a3deb976
Passphrase:
true
> personal.unlockAccount(eth.accounts[2])
Unlock account 0x7878017fb82099fafc715a93a7af907ccc2e0f9a
Passphrase:
true

```

Application **bonus-service** listens for transactions processed by the Ethereum node. It subscribes for notifications from Web3j library by calling **web3j.transactionObservable().subscribe(...)** method. It returns the amount of received transactions to the sender's account once per 10 transactions sent from that address. Here's the implementation of the observable method inside application **bonus-service**.

```
@Autowired
Web3j web3j;

@PostConstruct
public void listen() {
    Subscription subscription = web3j.transactionObservable().subscribe(tx -> {
        LOGGER.info("New tx: id={}, block={}, from={}, to={}, value={}", tx.getHash(),
tx.getBlockHash(), tx.getFrom(), tx.getTo(), tx.getValue().intValue());
        try {
            EthCoinbase coinbase = web3j.ethCoinbase().send();
            EthGetTransactionCount transactionCount = web3j.ethGetTransactionCount(tx.getFrom(),
DefaultBlockParameterName.LATEST).send();
            LOGGER.info("Tx count: {}", transactionCount.getTransactionCount().intValue());
            if (transactionCount.getTransactionCount().intValue() % 10 == 0) {
                EthGetTransactionCount tc = web3j.ethGetTransactionCount(coinbase.getAddress(),
DefaultBlockParameterName.LATEST).send();
                Transaction transaction = Transaction.createEtherTransaction(coinbase.getAddress(),
tc.getTransactionCount(), tx.getValue(), BigInteger.valueOf(21_000), tx.getFrom(), tx.getValue());
                web3j.ethSendTransaction(transaction).send();
            }
        } catch (IOException e) {
            LOGGER.error("Error getting transactions", e);
        }
    });
    LOGGER.info("Subscribed");
}
```

Conclusion

Blockchain and cryptocurrencies are not the easy topics to start. Ethereum simplifies development of applications that use blockchain, by providing a complete, scripting language. Using the web3j library together with Spring Boot and Docker image of Ethereum Geth client allows to quickly start local development of solutions implementing blockchain technology. IF you would like to try it locally just clone my repository available on GitHub <https://github.com/piomin/sample-spring-blockchain.git>