APPENDICES

Appendix 1. Smart Contract User Struct Code

```
var User = {
    secondaryWallet: ADDRESS,
    isRegistered: BOOLEAN
}

var users = MAPPING(ADDRESS - > User)
```

Appendix 2. Smart Contract registerSecondaryWallet Function Code

```
function registerSecondaryWallet(secondaryAddress:
ADDRESS) {
    if (secondaryAddress == ZERO_ADDRESS) {
        throw "Invalid address"
    }
    if (users[CALLER_ADDRESS].isRegistered == TRUE) {
        throw "Already registered"
    }

    users[CALLER_ADDRESS].secondaryWallet =
secondaryAddress
    users[CALLER_ADDRESS].isRegistered = TRUE

    emit UserRegistered(CALLER_ADDRESS, secondaryAddress)
}
```

Appendix 3. Smart Contract authenticate Function Code

```
function authenticate(messageHash, v1, r1, s1, v2, r2, s2)
returns BOOLEAN {
    var recoveredPrimary = RECOVER_ADDRESS(messageHash,
v1, r1, s1)
    var recoveredSecondary = RECOVER_ADDRESS(messageHash,
v2, r2, s2)

    var user_data = users[recoveredPrimary]

    if (user_data.isRegistered == TRUE and
user_data.secondaryWallet == recoveredSecondary) {
        return TRUE
    } else {
```

```
        return FALSE
    }
}
```

Appendix 4. Backend Database and Contract Configuration Code

```
// Initializes database connection pool and loads
blockchain configs
var DATABASE_POOL = CONNECT_TO_MYSQL_DB()
var RPC_URL = LOAD_ENV("RPC_URL")
var CONTRACT_ADDRESS = LOAD_ENV("CONTRACT_ADDRESS")

// Sets up Ethereum provider and smart contract instance
var ETHEREUM_PROVIDER = NEW_JSON_RPC_PROVIDER(RPC_URL)
var CONTRACT_ABI = [ /* ... contract function definitions
... */ ]
var SMART_CONTRACT_INSTANCE =
NEW_CONTRACT(CONTRACT_ADDRESS, CONTRACT_ABI,
ETHEREUM_PROVIDER)

// HTTPS Configuration
var SECURE_CONNECTION = LOAD_ENV("SECURE_CONNECTION") //
'true' or 'false'
var HTTPS_OPTIONS = LOAD_SSL_CERTIFICATES() // Loads key
and cert files
```

Appendix 5. Backend Registration Route Code

```
route POST / api / register {
    // Extracts and validates email/password
    var email = REQUEST.BODY.email
    var password = REQUEST.BODY.password

    // Validates input and checks if user already exists
    if (email or password MISSING) {
        return ERROR_400
    }
    if (USER_EXISTS_IN_DB(email)) {
        return ERROR_409
    }
```

```
    // Hashes password and saves new user to database
    var hashedPassword = HASH_PASSWORD(password)
    SAVE_USER_TO_DB(email, hashedPassword)

    return SUCCESS_201("User registered.")
}
```

Appendix 6. Backend Login Route Code

```
route POST / api / login {
    // Extracts and validates email/password
    var email = REQUEST.BODY.email
    var password = REQUEST.BODY.password

    // Finds user, compares password, and checks wallet
registration
    var user = FIND_USER_IN_DB(email)
    if (user NOT FOUND) {
        return ERROR_404
    }
    if (NOT COMPARE_HASH(password, user.password)) {
        return ERROR_401
    }

    // Check if wallets are registered; include primary
wallet address in JWT payload
    if (user.primary_wallet_address NOT SET or
user.secondary_wallet_address NOT SET) {
        return ERROR_403_NEEDS_SETUP(user.email)
    }

    // Creates and returns JWT token
    var token = CREATE_JWT({
        id: user.id,
        email: user.email,
        primaryWalletAddress: user.primary_wallet_address
    })
    return SUCCESS_200("Login successful.", token,
user.wallet_info)
}
```

Appendix 7. Backend Save Wallets Route

```
route POST / api / save - wallets {
    // Extracts user email and wallet addresses
    var email = REQUEST.BODY.email
    var primaryAddress = REQUEST.BODY.primaryAddress
    var secondaryAddress = REQUEST.BODY.secondaryAddress

    // Validates input
    if (email or addresses MISSING) {
        return ERROR_400
    }

    // Updates user's wallet addresses in the database
    var result = UPDATE_USER_WALLETS_IN_DB(email,
primaryAddress, secondaryAddress)

    if (result.AFFECTED_ROWS == 0) {
        return ERROR_404("User not found.")
    }

    return SUCCESS_200("Wallets saved.")
}
```

Appendix 8. Backend Challenge Message Route Pseudocode

```
route GET / api / challenge - message(PROTECTED BY
AUTH_TOKEN) {
    // Generates a unique timestamp-based challenge
message
    var message = "Login challenge at " +
CURRENT_TIMESTAMP()

    return SUCCESS_200({
        message: message
    })
}
```

Appendix 9. Backend Nonce Generation Route

```
route GET / api / nonce /: primaryAddress {
    var primaryAddress = REQUEST.PARAMS.primaryAddress
    if (primaryAddress IS NOT VALID_ETHEREUM_ADDRESS) {
        return ERROR_400
    }

    // Generates a new random UUID nonce
    var newNonce = GENERATE_UUID()

    // Stores/updates the nonce in the database for the
primary address
    SAVE_NONCE_TO_DB(primaryAddress, newNonce)

    return SUCCESS_200({
        nonce: newNonce
    })
}
```

Appendix 10. Backend 2FA Verification Route

```
route POST / api / verify - 2 fa {
    var originalMessage = REQUEST.BODY.originalMessage
    var sig1 = REQUEST.BODY.sig1
    var sig2 = REQUEST.BODY.sig2

    if (any_fields_missing) {
        return ERROR_400
    }

    var messageHash = HASH_MESSAGE(originalMessage)
    var [v1, r1, s1] = SPLIT_SIGNATURE(sig1)
    var [v2, r2, s2] = SPLIT_SIGNATURE(sig2)
    var isAuthenticated =
SMART_CONTRACT_INSTANCE.CALL_AUTHENTICATE(messageHash, v1,
r1, s1, v2, r2, s2)

    if (isAuthenticated) {
```

```
        return SUCCESS_200("2FA successful!")
    } else {
        return SUCCESS_200("2FA failed.")
    }
}
```

Appendix 11. Frontend App Component State Management and Navigation

```
component App {
    var screenState = 'login'
    var sessionState = {
        isAuthenticated: FALSE,
        user: NULL,
        token: NULL
    }

    function navigateTo(newScreen) {
        screenState = newScreen
    }

    render {
        if (screenState == 'register') {
            return RegisterPage(...)
        } else if (screenState == 'setup-wallets') {
            return SetupWalletsPage(...)
        } else if (screenState == '2fa') {
            return TwoFactorAuthApp(...)
        } else if (screenState == 'home') {
            return HomePage(...)
        } else {
            return LoginPage(...)
        }
    }
}
```

Appendix 12. Frontend LoginPage Component Pseudocode

```
component LoginPage(onProceedTo2FA, onNavigateToRegister,
onNeedsWalletSetup) {
    var emailState = ''
    var passwordState = ''
```

```
    function handleLogin(event) {
        PREVENT_DEFAULT(event) try {
            var response = FETCH_POST(BACKEND_URL +
'/login', {
                emailState,
                passwordState
            }) if (response.status != OK) {
                if (response.needsWalletSetup) {
                    CALL
onNeedsWalletSetup(response.email)
                } else {
                    throw response.error
                }
            } else {
                CALL onProceedTo2FA(response.user,
response.token)
            }
        } catch (error) {
            /* ... display error ... */ }
    }
    render {
        /* ... login form UI ... */ }
}
```

Appendix 13. Frontend RegisterPage Component Pseudocode

```
component RegisterPage(onNavigateToLogin,
onRegistrationSuccess) {
    var emailState = ''
    var passwordState = ''

    function handleRegister(event) {
        PREVENT_DEFAULT(event)
        try {
            var response = FETCH_POST(BACKEND_URL +
'/register', {
                emailState,
                passwordState
            })
            if (response.status != OK) {
```

```
                throw response.error
            }
            SET_TIMEOUT(function() {
                CALL onRegistrationSuccess(emailState)
            }, 1500)
        } catch (error) {
            /* ... display error ... */ }
    }

    render {
        /* ... registration form UI ... */ }
}
```

Appendix 14. Frontend SetupWalletsPage Component

```
component SetupWalletsPage(email, onSetupComplete,
onBackToLogin) {
    var primaryAddressState = NULL
    var secondaryAddressInputState = ''
    var onChainStatusState = NULL

    function connectPrimaryWallet() {
        primaryAddressState = CONNECTED_METAMASK_ADDRESS
        CALL checkRegistrationStatus(primaryAddressState,
primaryProviderState)
    }

    function handleFinalizeSetup() {
        if (onChainStatusState.isRegistered) {
            CALL saveWalletsToDB(primaryAddressState,
onChainStatusState.secondaryWallet)
        } else {
            var tx = AWAIT
CONTRACT_INSTANCE.registerSecondaryWallet(secondaryAddress
InputState)
            AWAIT tx.WAIT_FOR_CONFIRMATION()
            CALL saveWalletsToDB(primaryAddressState,
secondaryAddressInputState)
        }
    }
```

```
    function saveWalletsToDB(primary, secondary) {
        var response = FETCH_POST(BACKEND_URL + '/save-
wallets', {
            email,
            primaryAddress: primary,
            secondaryAddress: secondary
        })
        if (response.status != OK) {
            throw response.error
        }
        SET_TIMEOUT(CALL onSetupComplete(), 2000)
    }

    render {
        /* ... wallet setup UI ... */ }
}
```

Appendix 15. Frontend TwoFactorAuthApp Component

```
component TwoFactorAuthApp(user, token, on2FAComplete,
onBackToLogin) {
    var primaryAddressState = NULL
    var secondaryWalletAddressState = NULL
    var loginMessageState = ''
    var sig1State = ''
    var sig2State = ''

    function handleInitiateLogin() {
        var challenge = AWAIT FETCH_CHALLENGE_MESSAGE()
        loginMessageState = challenge.message
        sig1State = AWAIT
SIGN_MESSAGE(PRIMARY_WALLET_SIGNER, loginMessageState)
        AWAIT
CONNECT_SECONDARY_WALLET_AND_SIGN(loginMessageState)
    }

    function
CONNECT_SECONDARY_WALLET_AND_SIGN(messageToSign) {
```

```
        sig2State = AWAIT
SIGN_MESSAGE(SECONDARY_WALLET_SIGNER, messageToSign)
    }

    function handleVerify2FA(msg, s1, s2_val, pAddr) {
        var response = FETCH_POST(BACKEND_URL + '/verify-
2fa', {
            msg,
            s1,
            s2_val,
            pAddr
        }, AUTH_HEADER: token)
        if (response.status != OK) {
            throw response.message
        }
        if (response.success) {
            CALL on2FAComplete()
        } else {
            /* ... display failure ... */ }
    }

    render {
        /* ... 2FA setup and login UI ... */ }
}
```

The complete code can be found on the GitHub repository linked below, but you'll need to contact the author directly to get the access for the full implemented code.

Link: https://github.com/chiknwy/DualWalletAuth

BIOGRAPHY

Chiko Gita Satria was born on April 19, 2004, in Jakarta, Indonesia. He currently lives in his hometown, South Jakarta. He completed his elementary education at SDN 09 Jakarta, graduating in 2016, and continued his studies at SMPN 178 Jakarta, graduating in 2019. He then attended SMAN 87 Jakarta for his senior high school education, where he concentrated in Mathematics and Science. After graduating from high school, he aimed to continue his studies at a university and was accepted into Universitas Pendidikan Ganesha, where he chose Computer Science as his major. He has been an active student at the university since 2022 and remains enrolled at the time of writing this undergraduate thesis.