

Τεχνολογικό Υπόβαθρο

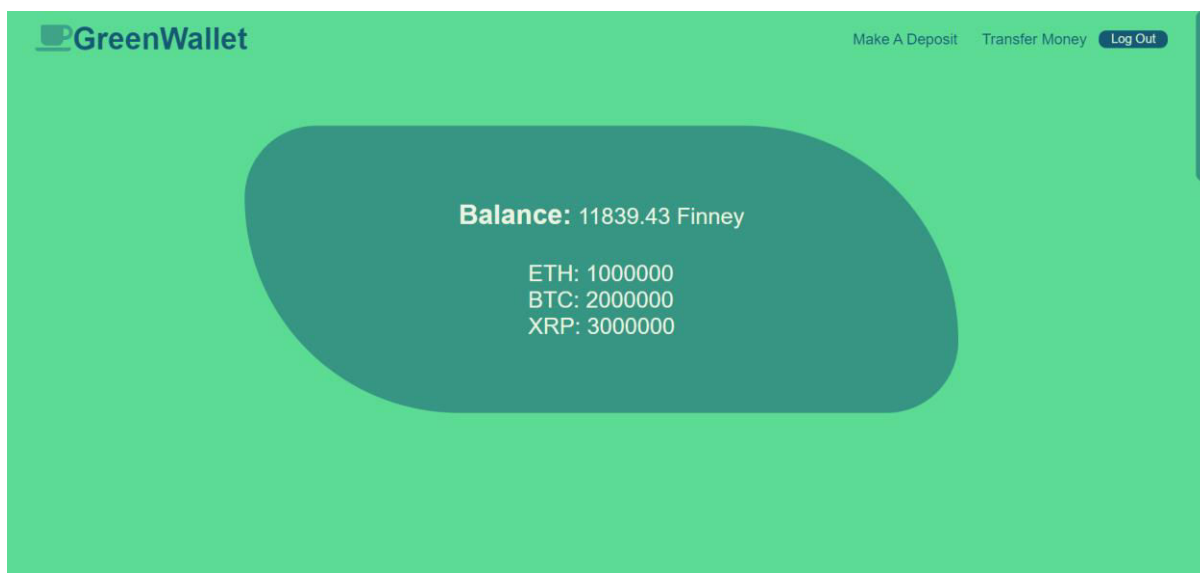
- Blockchain:
- Blockchain Block:
- Ethereum:
- Smart Contract:
- Server:
- JWT:
- SQLite:
- SHA224:
- API:
- Hash:
- Query:
- HS256:
- Hexadecimal Seed:
- JSON-RPC:
- React:
- Prop:
- JSON:
- GET Request:
- POST Request:
- Heroku:

Συντομογραφίες

- F/E - f/e: Front end, το κομμάτι της εφαρμογής με το οποίο διαδρά ο χρήστης
- B/E - b/e: Back end, το κομμάτι της εφαρμογής που υλοποιεί βασικές λειτουργίες οι οποίες σχετίζονται με την βάση δεδομένων
- CRUD: Create / Read / Update / Delete
- JWT: JSON Web Tokens
- ETH: Ethereum
- BTC: Bitcoin
- XRP: Ripple
- ORM: Object Relational Mapper
- API: Application Programming Interface
- DOM: Document Object Model

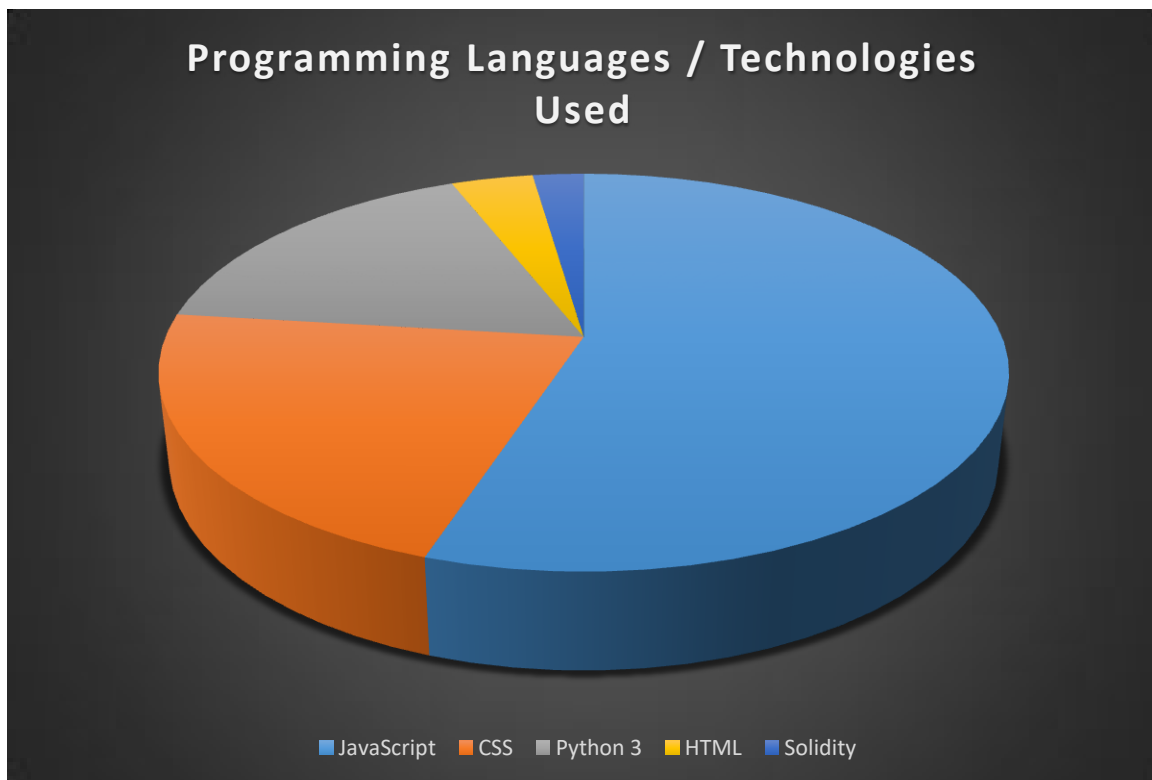
Εφαρμογή GreenWallet

Το GreenWallet είναι μια εφαρμογή η οποία αποτελεί ένα ψηφιακό πορτοφόλι το οποίο δίνει την δυνατότητα στους χρήστες του να εκτελούν συναλλαγές με καταστήματα τα οποία επιτρέπουν την πληρωμή μέσω του Ethereum Blockchain. Οι συναλλαγές πραγματοποιούνται με την μεταφορά χρημάτων από το πορτοφόλι του χρήστη στο smart contact του αντίστοιχου μαγαζιού. Επιπλέον υποστηρίζονται συναλλαγές μεταξύ χρηστών, από πορτοφόλι σε πορτοφόλι χωρίς την χρήση smart contact, αλλά και η κατάθεση χρημάτων από την χρεωστική κάρτα του χρήστη στο ψηφιακό του πορτοφόλι.



Γλώσσες προγραμματισμού, βιβλιοθήκες και λοιπές τεχνολογίες

Οι γλώσσες προγραμματισμού / τεχνολογίες διαδικτύου που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής, κατά ποσοστό σύμφωνα με το GitHub (<https://github.com/JohnKazantzis/Scooter-Transactions>) είναι:



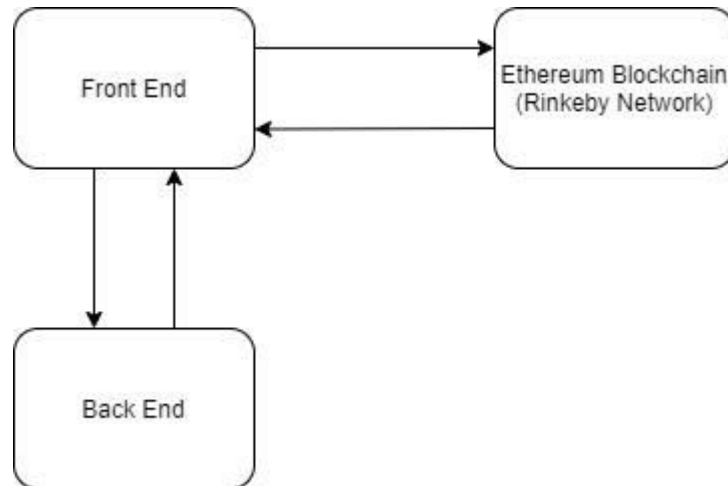
Χρησιμότητα κάθε γλώσσας / τεχνολογίας:

- **JavaScript:** Χειρισμός του DOM (Document Object Model), διαχείριση συμβάντων (events), κλήσεις API τα οποία χρησιμοποιούν μυστικά κλειδιά, διαδραση με το b/e και το Ethereum Blockchain
- **CSS:** Δημιουργία ομορφότερης γραφικής διασύνδεσης και animation
- **Python 3:** Δημιουργία του b/e, υποδοχή και διαχείριση αιτημάτων από το f/e
- **HTML:** Δημιουργία της βασικής γραφικής διασύνδεσης
- **Solidity:** Δημιουργία των smart contracts των καταστημάτων που υποστηρίζουν πληρωμές μέσω του Ethereum Blockchain

Βιβλιοθήκες / Frameworks που χρησιμοποιούνται ανά γλώσσα προγραμματισμού:

JavaScript	Python 3
React	sqlite3
web3	flask
axios	flask_classful
@truffle/hdwallet-provider	flask_cors
@fortawesome/react-fontawesome	requests
@fortawesome/free-solid-svg-icons	json
react-elastic-carousel	sqlalchemy
	hashlib
	pywallet
	jwt
	datetime

Αρχιτεκτονική Υψηλού Επιπέδου



Για την λειτουργία της εφαρμογής, το κομμάτι με το οποίο διαδρά ο χρήστης (f/e) πρέπει να επικοινωνεί με το b/e και το Ethereum blockchain. Το b/e αναλαμβάνει κυρίως το βάρος της ταυτοποίησης του χρήστη και της αποθήκευσης βασικών δεδομένων για την ομαλή λειτουργία της εφαρμογής. Με την επικοινωνία με το Ethereum blockchain γίνεται δυνατή η μεταφορά χρημάτων στα smart contracts των συνεργαζόμενων μαγαζιών ή στο πορτοφόλι κάποιου άλλου χρήστη του GreenWallet. Η μεταφορά χρημάτων σε δημόσια διεύθυνση που δεν ανήκει στο GreenWallet είναι επίσης δυνατή. Πιο συγκεκριμένα:

Λειτουργίες που εκτελεί το b/e είναι:

- Αποθήκευση βασικών πληροφοριών για τα καταστήματα / smart contracts που έχει αποθηκεύσει ο χρήστης
- Αποθήκευση αναγκαίων πληροφοριών για την ταυτοποίηση του χρήστη
- Ταυτοποίηση του χρήστη
- Δημιουργία “session” για την ομαλότερη λειτουργία της εφαρμογής και την καλύτερη εμπειρία περιήγησης του χρήστη (JWT)
- Ενημέρωση της εφαρμογής με τις πιο πρόσφατες συναλλαγματικές αξίες των τριών πιο χρησιμοποιημένων κρυπτονομισμάτων, συγκεκριμένα του ETH, του BTC και του XRP
- Παραγωγή mnemonic για την δημιουργία καινούργιων πορτοφολιών

Λειτουργίες που εκτελούνται στο Blockchain:

- Μεταφορά χρημάτων από πορτοφόλι σε πορτοφόλι μεταξύ χρηστών χωρίς την μεσολάβηση smart contract
- Πληρωμές σε καταστήματα που υποστηρίζουν την πληρωμή μέσω του GreenWallet και την χρήση smart contracts τα οποία βρίσκονται στο Rinkeby testnet του Ethereum Blockchain
- Ενημέρωση για το υπόλοιπο του λογαριασμού του χρήστη

Λειτουργίες του f/e:

- Σύνδεση και προβολή όλων των λειτουργιών που προσφέρουν το b/e και το blockchain
- Δημιουργία μιας ευχάριστης εμπειρίας για τον χρήστη

Ανάλυση υλοποίησης των λειτουργιών της εφαρμογής GreenWallet

Αποθήκευση αναγκαίων πληροφοριών για την ταυτοποίηση του χρήστη:

Η διαδικασία της αποδοχής των στοιχείων, τα οποία εισάγει ο χρήστης, και η εισαγωγή τους στην βάση γίνεται στο b/e κομμάτι της εφαρμογής. Η εφαρμογή χρησιμοποιεί την SQLite για την διαχείριση της σχεσιακής βάσης δεδομένων στην οποία αποθηκεύονται όλα τα δεδομένα της εφαρμογής. Για την ευκολότερη ενσωμάτωση και χρήση της βάσης δεδομένων, χρησιμοποιείται ένα ORM το SQLAlchemy (<https://www.sqlalchemy.org/>).

Στην βάση δεδομένων ένας χρήστης αναπαρίσταται από τα παρακάτω στοιχεία:

- **Id:** Ένας μοναδικός ακέραιος αριθμός ο οποίος ξεχωρίζει τον χρήστη από όλους τους άλλους χρήστες. Ο αριθμός αυτός συμπληρώνεται απευθείας από την βάση δεδομένων κατά την δημιουργία του χρήστη. Το Id αποτελεί το κύριο κλειδί του πίνακα.
- **Username:** Το όνομα με το οποίο αναπαρίσταται ο χρήστης μέσα στην εφαρμογή. Δύο χρήστες δεν επιτρέπεται να έχουν το ίδιο Username.
- **Password:** Το συνθηματικό του χρήστη, το οποίο είναι αναγκαίο για την ταυτοποίηση του στην εφαρμογή. Ο κάθε χρήστης μπορεί να επιλέξει το συνθηματικό του κατά την δημιουργία του λογαριασμού του. Για την μεγαλύτερη ασφάλεια των δεδομένων των χρηστών, κανένα συνθηματικό δεν αποθηκεύεται στην βάση δεδομένων. Στην βάση δεδομένων αποθηκεύεται το αποτέλεσμα που προκύπτει από την εφαρμογή της συνάρτησης κατακερματισμού SHA224 στον κωδικό που παρέχει ο χρήστης.
- **Mnemonic:** Μια σειρά 12 τυχαίων λέξεων, στην αγγλική γλώσσα, οι οποίες χρησιμοποιούνται για την παραγωγή του ιδιωτικού κλειδιού του πορτοφολιού του χρήστη. Παραδείγματος χάριν: 'almost spice garment loop slight blanket copper sun empty cement work sail'. Το Mnemonic παράγεται αυτόματα από την εφαρμογή, όταν ο χρήστης δημιουργεί τον λογαριασμό του.

```
class User(base):  
    __tablename__ = 'Users'  
  
    Id = Column(Integer, primary_key=True, autoincrement=True)  
    Username = Column(String)  
    Password = Column(String)  
    Mnemonic = Column(String)
```

// Κλάση η οποία περιγράφει την δομή του πίνακα του χρήστη στο ORM

Οι βασικές λειτουργίες που υλοποιούνται πάνω στα δεδομένα του χρήστη είναι:

- **Η δημιουργία νέου χρήστη:**

Η δημιουργία ενός νέου χρήστη γίνεται με την υποβολή της αντίστοιχης φόρμας στο f/e κομμάτι της εφαρμογής. Κατά την υποβολή γίνεται χρήση του εκτεθειμένου API υπεύθυνου για την δημιουργία νέων χρηστών, το οποίο βρίσκεται (τοπικά) στο `'http://127.0.0.1:5000/createUser/'`. Ο κώδικας που τρέχει με την κλήση του API είναι υπεύθυνος για την υποδοχή των δεδομένων, την παραγωγή ενός μνημονικού, την δημιουργία του hash που θα αποθηκευτεί στην θέση του συνθηματικού του χρήστη και τελικά την αποθήκευση των δεδομένων στην βάση. Για την συγγραφή πιο κατανοητού αλλά και επεκτάσιμου κώδικα, όλες αυτές οι λειτουργίες δεν εκτελούνται μέσα σε μια μέθοδο αλλά σε πολλές μεθόδους οι οποίες εκτελούν μια συγκεκριμένη λειτουργία.

Αποθήκευση βασικών πληροφοριών για τα καταστήματα / smart contracts που έχει αποθηκεύσει ο χρήστης

Η αποθήκευση των βασικών πληροφοριών ενός smart contract / καταστήματος ξεκινάει με την εισαγωγή των στοιχείων από τον χρήστη και την μεταφορά τους b/e της εφαρμογής (περισσότερες πληροφορίες στο «Πληρωμές σε καταστήματα που υποστηρίζουν τις πληρωμές μέσω του Ethereum Blockchain» κομμάτι της εργασίας).

Στην βάση δεδομένων ένα smart contract / κατάστημα αναπαρίσταται από τα παρακάτω στοιχεία:

- **Name:** Το όνομα με το οποίο θα εμφανίζεται το αποθηκευμένο κατάστημα στην γραφική διασύνδεση του χρήστη. Το όνομα μπορεί να είναι οποιοδήποτε αλφαριθμητικό θελήσει ο χρήστης.
- **Address:** Η διεύθυνση στην οποία βρίσκεται το smart contract στο Rinkeby testnet του Ethereum Blockchain.
- **FunctionName:** Το όνομα της μεθόδου του smart contract η οποία υλοποιεί την διαδικασία της πληρωμής (Payable function). Παραδείγματος χάριν:

```
function makePayment() public payable {  
    assert(totalBalance + msg.value >= totalBalance);  
    totalBalance += msg.value;  
}
```

- **UserId:** Το Id του χρήστη στον οποίο ανήκει το αποθηκευμένο κατάστημα. Το UserId είναι απαραίτητο για να γνωρίζει το σύστημα ποια smart contracts / καταστήματα να επιστρέψει στην γραφική διασύνδεση του χρήστη.

```
class Contract(base):  
    __tablename__ = 'Contracts'  
  
    Name = Column(String)  
    Address = Column(String, primary_key=True)  
    FunctionName = Column(String)  
    userId = Column(Integer)
```


Οι βασικές λειτουργίες που υλοποιούνται πάνω στα δεδομένα του χρήστη είναι:

- **Η δημιουργία νέου smart contract / καταστήματος:**

Περισσότερες πληροφορίες στο «Πληρωμές σε καταστήματα που υποστηρίζουν τις πληρωμές μέσω του Ethereum Blockchain» κομμάτι της εργασίας.

Ταυτοποίηση του χρήστη:

Η διαδικασία της ταυτοποίησης του χρήστη ξεκινάει με το που μπει στην εφαρμογή. Η διαδικασία αυτή διαχωρίζεται σε δύο κυρίως βήματα:

1. Τον έλεγχο των δεδομένων που εισάγει ο χρήστης
 2. Την έκδοση ενός JSON Web Token το οποίο επιστρέφεται στον χρήστη
-
1. Για να μπορέσει ένας χρήστης να χρησιμοποιήσει την εφαρμογή, πρέπει πρώτα είτε να συνδεθεί στον ήδη υπάρχον λογαριασμό του ή να φτιάξει έναν καινούργιο και να συνδεθεί σε αυτόν. Ο χρήστης παρέχει τα στοιχεία του σε μια φόρμα στο f/e κομμάτι της εφαρμογής τα οποία με την κλήση του αντίστοιχου API, τοπικά στο `'http://127.0.0.1:5000/login/'`, μεταφέρονται στο b/e. Ο κώδικας που τρέχει με την κλήση του API είναι υπεύθυνος για:
 - Να μετατρέψει το συνθηματικό που έχει εισάγει ο χρήστης στην φόρμα και έχει περαστεί στις παραμέτρους της κλήσης του API σε hash. Για τον κατακερματισμό του συνθηματικού (SHA224) είναι υπεύθυνη η μέθοδος `passwordHashing(password)` η οποία κάνει χρήση της hashlib βιβλιοθήκης της python και πιο συγκεκριμένα της μεθόδου `sha224(password)` (python docs : <https://docs.python.org/3/library/hashlib.html>)
 - Μετά την επιστροφή του hash του συνθηματικού, γίνεται ένα query στην βάση, το οποίο μας επιστρέφει όλες τις εγγραφές που υπάρχουν, φιλτραρισμένες με το Username και το Hash. Είναι πιθανό να γυρίσει μια ή καμία εγγραφή μιας και το Username είναι μοναδικό. Η περίπτωση που δεν υπάρχει καμία εγγραφή μας υποδεικνύει ότι ο χρήστης έχει εισάγει λάθος στοιχεία. Στην περίπτωση που επιστραφεί μια εγγραφή, σημαίνει ότι ο χρήστης έχει πληκτρολογήσει σωστά τα στοιχεία του και επομένως το σύστημα συνεχίζει με την έκδοση ενός JSON Web Token
 2. Για την έκδοση του JSON Web Token γίνεται χρήση της βιβλιοθήκης `pyjwt` (<https://pyjwt.readthedocs.io/en/latest/>). Όλες οι λειτουργίες που συσχετίζονται με το JSON Web Token βρίσκονται στο αρχείο `JWTTokenUtils.py`. Οι βασικοί μέθοδοι που περιέχονται είναι οι εξής:
 - `createToken(userId)`: Η μέθοδος δέχεται ως όρισμα το Id του χρήστη που επιστράφηκε κατά τον έλεγχο των δεδομένων του, το προσθέτει στο ωφέλιμο φορτίο και με την χρήση ενός μυστικού κλειδιού δημιουργεί το JSON Web Token. Κάθε token δίνει την δυνατότητα στον χρήστη να αποδεικνύει την ταυτότητα του για 20 λεπτά μετά την έκδοση του ή μέχρι αυτός να κλείσει τον

περιηγητή του. Για την δημιουργία του token χρησιμοποιείται ο συμμετρικός αλγόριθμος HS256 (HMAC with SHA-256).

- **checkToken(token):** Η μέθοδος δέχεται ως όρισμα το JSON Web Token που παρέχει ο χρήστης με κάθε API call μετά την είσοδο του στο σύστημα και ελέγχει αν αυτό είναι έγκυρο. Δυο βασικοί λόγοι που μπορεί ένα token να μην είναι έγκυρο είναι να έχει λήξει ή να έχει τροποποιηθεί από κάποιον κακόβουλο χρήστη. Και στις δυο περιπτώσεις το token απορρίπτεται από το σύστημα.

```
def createToken(userId):  
    # Create JWT token  
    payload = {  
        'exp': datetime.datetime.utcnow()+datetime.timedelta(minutes=20),  
        'iat': datetime.datetime.utcnow(),  
        'sub': userId  
    }  
  
    token = jwt.encode(payload, SECRET_KEY, algorithm='HS256')  
  
    return token
```

// The method responsible for creating the JSON Web Token

```
def checkToken(token):  
    # Decoding the token  
    try:  
        payload = jwt.decode(token, SECRET_KEY)  
        return payload['sub']  
    except jwt.ExpiredSignatureError:  
        print('Token Expired')  
        return None  
    except jwt.InvalidTokenError:  
        print('Invalid Token')  
        return None
```

// The method responsible for checking the JSON Web Token

Ενημέρωση της εφαρμογής με τις πιο πρόσφατες συναλλαγματικές αξίες των τριών πιο χρησιμοποιημένων κρυπτονομισμάτων:

Η συναλλαγματική αξία του ETH δεν είναι σταθερή, γεγονός που σημαίνει ότι η αξία των χρημάτων του χρήστη αλλάζει συνεχώς. Ο χρήστης πρέπει να είναι ικανός να γνωρίζει την συναλλαγματική αξία του ETH ώστε να μπορεί να υπολογίσει πότε είναι προτιμότερο να προσθέσει ή να αφαιρέσει χρήματα από το λογαριασμό του. Οι τιμές του ETH, του BTC και του XRP γνωστοποιούνται στο σύστημα με ένα API call στο CoinLayer API (<https://coinlayer.com/>). Το API call θα ήταν δυνατό να γίνει απευθείας στο f/e κομμάτι της

εφαρμογής, παρ' όλ' αυτά επειδή χρησιμοποιείται ένα μυστικό κλειδί το οποίο πρέπει να παραμείνει κρυφό, γίνεται στο b/e. Οι πιο πρόσφατες τιμές επιστρέφονται στο b/e και από εκεί αποστέλλονται στο f/e όπου και εμφανίζονται στην αρχική σελίδα του χρήστη.

```
def getExchangeRates() :  
    #  
    # Making an API call to the CoinLayer API  
    # to get the current exchange rates  
    #  
    data = request.args.to_dict()  
  
    coin = data['target']  
    COINLAYER_URL = 'http://api.coinlayer.com/live'  
    COINLAYER_API_KEY = 'eb3c76bbef63fe4aa71b802bc25bef0c'  
    PARAMS = { 'access_key': COINLAYER_API_KEY, 'target': coin }  
    HEADERS = { 'content-type': 'application/json' }  
  
    response = requests.get(url=COINLAYER_URL, params=PARAMS,  
                            headers=HEADERS)  
    data = json.loads(response.text)  
  
    dataToSend = { 'ETH': data['rates']['ETH'],  
                  'BTC': data['rates']['BTC'],  
                  'XRP': data['rates']['XRP']  
    }  
  
    responseToReact = flask.Response(json.dumps(dataToSend))  
    responseToReact.headers['Access-Control-Allow-Origin'] = '*'  
    return responseToReact
```

// The method responsible for calling the CoinLayer API in order to get the new exchange rates

Παραγωγή mnemonic για την δημιουργία καινούργιων πορτοφολιών

Όπως προαναφέρθηκε στο κεφάλαιο ‘Αποθήκευση αναγκαίων πληροφοριών για την ταυτοποίηση του χρήστη’ μια από τις βασικές ευθύνες της εφαρμογής κατά την δημιουργία ενός νέου χρήστη, είναι η παραγωγή ενός μνημονικού το οποίο αποθηκεύεται στην βάση δεδομένων μαζί με όλα τα υπόλοιπα στοιχεία του χρήστη. Η χρήση του μνημονικού αρχικά προτάθηκε στο Bitcoin Improvement Proposal 39

(<https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>) και έκτοτε υλοποιείται στις περισσότερες εφαρμογές Blockchain πορτοφολιών. Το μνημονικό είναι πολύ χρήσιμο μιας και από αυτό μπορεί να παραχθεί ένα 512-bit hexadecimal seed, από το οποίο μετά μπορούν να παραχθούν τα ιδιωτικά και τα δημόσια κλειδιά του λογαριασμού. Ένα μεγάλο όφελος της χρήσης του μνημονικού έναντι της χρήσης μόνο του hexadecimal seed, είναι ότι είναι πολύ πιο εύκολο να απομνημονευθεί από τον χρήστη μιας και είναι ουσιαστικά μια σειρά από 12-24 λέξεις. Στην εφαρμογή GreenWallet γίνεται η χρήση μνημονικών των 12 λέξεων. Για την παραγωγή του μνημονικού χρησιμοποιείται η μέθοδος `generate_mnemonic()` της κλάσης `wallet` από την βιβλιοθήκη της python pywallet (<https://pypi.org/project/pywallet/>).

```
from pywallet import wallet
```

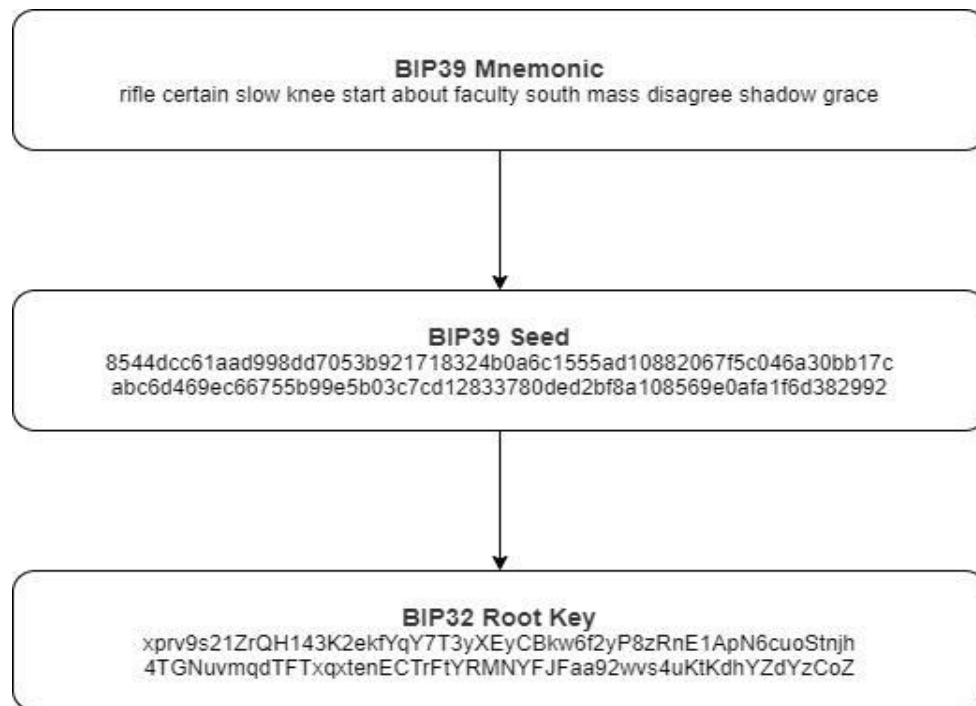
```
class walletUtils:
```

```
    @staticmethod
```

```
    def createSeed():
```

```
        return wallet.generate_mnemonic()
```

// The method responsible for the generation of the new wallet's mnemonic



Αρχικοποίηση του πορτοφολιού

Αμέσως μετά την επιτυχή σύνδεση του χρήστη στην εφαρμογή, καλείται η μέθοδος `initialiseWallet()`. Σε αυτή την μέθοδο γίνεται η δημιουργία του βασικού αντικειμένου της βιβλιοθήκης της JavaScript Web3. Όλες οι λειτουργίες που σχετίζονται με την διάδραση της εφαρμογής με το Ethereum Blockchain βασίζονται στην χρήση αυτού του αντικειμένου (<https://web3js.readthedocs.io/en/v1.2.11/web3.html>). Για την δημιουργία του αντικειμένου χρειάζεται να περάσουμε ως όρισμα έναν προμηθευτή (Provider). Ο provider είναι το μέσο με το οποίο η εφαρμογή «μιλάει» με το Ethereum Blockchain και είναι υπεύθυνο για την υποδοχή JSON-RPC αιτημάτων και την επιστροφή της απάντησης τους. Το GreenWallet χρησιμοποιεί έναν HTTP provider, ο οποίος δημιουργείται με την χρήσης της βιβλιοθήκης της JavaScript `@truffle/hdwallet-provider` (<https://www.npmjs.com/package/@truffle/hdwallet-provider>). Επίσης γίνεται η χρήση ενός Infura Ethereum node έναντι της χρήσης ενός προσωπικού Ethereum node. Κάποια από τα πλεονεκτήματα της χρήσης του Infura Ethereum node είναι:

- Ταχύτητα

- Επεκτασιμότητα
- Ευκολία χρήσης

Μετά την δημιουργία του web3 αντικειμένου, η εφαρμογή ανακτά την διεύθυνση του λογαριασμού του χρήστη, την οποία και αποθηκεύει για μετέπειτα χρήση στο state του React Web Application.

```

initialiseWallet = async () => {
  const infuraApiKey = "97bda0faaf4b44cb87286a8c0dd98e16";

  // Creating a rinkeby TestNet provider
  const web3 = new Web3(
    new HDWalletProvider(localStorage.getItem("Mnemonic"),
      'https://rinkeby.infura.io/v3/'+infuraApiKey));

  // Getting accounts
  const accounts = await web3.eth.getAccounts();

  // Get balance for account 0
  let walletBalance = await web3.eth.getBalance(accounts[0]);
  walletBalance = web3.utils.fromWei(walletBalance, 'finney');

  // Setting the state
  this.setState({
    accounts: accounts,
    walletBalance: walletBalance,
    web3instance: web3
  });
}

```

// The method responsible for initialization of the wallet

Ενημέρωση για το υπόλοιπο του λογαριασμού του χρήστη

Μια πολύ σημαντική δυνατότητα, η οποία είναι απαραίτητη για κάθε εφαρμογή wallet, είναι η ενημέρωση του χρήστη για το υπόλοιπο του αμέσως όταν αυτό αλλάζει. Το υπόλοιπο του ο χρήστης μπορεί να το δει στην αρχική σελίδα της εφαρμογής. Για την ανάκτηση του υπολοίπου του χρήστη γίνεται η χρήση της μεθόδου (`web3.eth.getBalance(accountAddress)`), του web3 αντικειμένου (Περισσότερες πληροφορίες στο μέρος «Αρχικοποίηση του πορτοφολιού») και καλείται πάντα αμέσως μετά την κλήση μεθόδων οι οποίες μπορεί να έχουν ως αποτέλεσμα την αλλαγή της κατάστασης του Blockchain και συγκεκριμένα της αλλαγής του υπολοίπου του λογαριασμού του χρήστη. Λειτουργίες που αλλάζουν το υπόλοιπο του χρήστη είναι η μεταφορά χρημάτων από την χρεωστική του κάρτα στο πορτοφόλι του, η μεταφορά χρημάτων σε πορτοφόλια άλλων χρηστών και τέλος η μεταφορά χρημάτων σε καταστήματα μέσω της χρήσης smart contract.

Ένα πολύ σημαντικό μέρος της υλοποίησης είναι η απεικόνιση του υπολοίπου του χρήστη σε μονάδες Finney αντί για Eth ή Wei που αποτελούν τις πιο γνωστές αλλά και σημαντικές μονάδες του κρυπτονομίσματος του Ethereum blockchain. Η χρήση του Finney είναι απαραίτητη μιας και η εφαρμογή GreenWallet χρησιμοποιείται κυρίως για τις καθημερινές συναλλαγές του χρήστη κατά τις οποίες η μονάδα ενός Eth είναι τεράστια αλλά και μονάδα

ενός Wei πολύ μικρή. Η μονάδα Finney είναι η καταλληλότερη για την μεγιστοποίηση της ευχρηστίας της εφαρμογής για τον χρήστη.

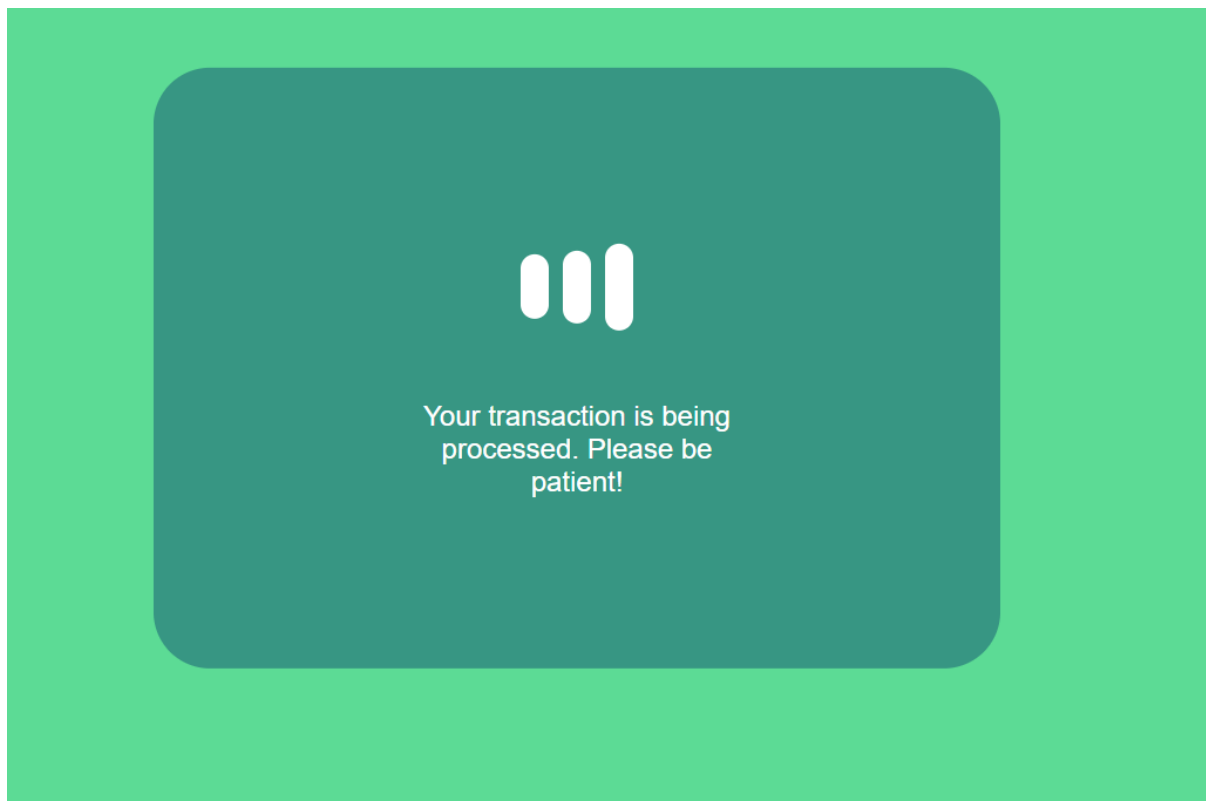
Η αξία των μονάδων την Δευτέρα 10/9/2020 ήταν:

Ethereum Unit	Euros (€)
1 Eth	337,27 €
1 Finney	0.3372 €
1 Wei	$337,27 \times 10^{-18} \text{ €}$

Ethereum Units Comparison			
Unit/Unit	Eth	Finney	Wei
Eth		10^3	10^{-18}
Finney	10^{-3}		10^{-15}
Wei	10^{18}	10^{15}	

Μεταφορά χρημάτων από το πορτοφόλι του χρήστη στο πορτοφόλι ενός άλλου χρήστη:

Μια σημαντική δυνατότητα που παρέχει το GreenWallet στους χρήστες του, είναι η δυνατότητα μεταφοράς χρημάτων μεταξύ των πορτοφολιών δύο διαφορετικών χρηστών. Η εκτέλεση μιας τέτοιας λειτουργίας είναι πολύ απλή για τον χρήστη αφού χρειάζεται μόνο να παρέχει την διεύθυνση του χρήστη στον οποίο θέλει να μεταφέρει τα χρήματα και το επιθυμητό ποσό.



Με το πάτημα του κουμπιού «Transfer Funds» από τον χρήστη, τα δεδομένα που έχουν περαστεί στην φόρμα χρησιμοποιούνται ως ορίσματα για την κλήση της μεθόδου. Για να ολοκληρωθεί μια τέτοια συναλλαγή, η οποία αλλάζει την κατάσταση του Blockchain και επομένως απαιτεί να γίνει εξόρυξη ενός καινούργιου block, απαιτείται παραπάνω χρόνος από ότι θα χρειαζόταν μια απλή κλήση μεθόδου ενός smart contract η οποία δεν αλλάζει την κατάσταση του Blockchain. Για την καλύτερη εμπειρία του χρήστη, κατά την διάρκεια της εκτέλεσης της συναλλαγής, η γραφική διασύνδεση δεν παγώνει. Ο χρήστης μπορεί να συνεχίσει την πλοήγηση του στην εφαρμογή και να ξεκινήσει την εκτέλεση επιπλέον συναλλαγών. Κατά την διάρκεια της εκτέλεσης της συναλλαγής, στην θέση της φόρμας εμφανίζεται ένα spinner component. Σύμφωνα με το <https://www.rinkeby.io/> ο μέσος όρος εξόρυξης ενός block (Average Block Time) στο Rinkeby network είναι 15 δευτερόλεπτα. Ο χρόνος εξόρυξης δεν είναι σταθερός αλλά αλλάζει ανάλογα με το πόσο γρήγορα μπορεί να βρεθεί το «hash στόχος» από τα μηχανήματα που χρησιμοποιούνται για την εξόρυξη των καινούργιων block. Η δυσκολία της εύρεσης του «hash στόχου» γίνεται πιο εύκολη/δύσκολη αυτόματα, για να διατηρηθεί ο χρόνος εξόρυξης στα 15 δευτερόλεπτα. Στην παρακάτω εικόνα εμφανίζονται ο μέσος όρος εξόρυξης ενός block και ένα διάγραμμα με τους χρόνους εξόρυξης των τελευταίων block.



Μετά το τέλος της εκτέλεσης της συναλλαγής, μπορούμε να την δούμε στο <https://rinkeby.etherscan.io/> όπως στην παρακάτω εικόνα:

Η μέθοδος η οποία ξεκινάει την εκτέλεση της συναλλαγής είναι:

```
handleSubmit = async event => {
  event.preventDefault();

  // Getting web3 instance and accounts
  const web3 = this.props.instance;
  const accounts = this.props.accounts;

  // Converting Finney to Wei
  let ammountInWei = web3.utils.toWei(this.state.amount.toString(),
    'finney');

  // Start Loading
  this.setState({ loading: true });
```



```

    // Send Money
    await web3.eth.sendTransaction({
      from: accounts[0],
      to: this.state.address,
      value: ammountInWei
    });

    // Get Wallet Balance
    let walletBalance = await web3.eth.getBalance(accounts[0]);
    walletBalance = web3.utils.fromWei(walletBalance, 'finney');

    // Setting App cmp state
    this.props.onBalanceChange (walletBalance += this.state.amount);

    // Stop Loading
    this.setState({ loading: false });
  }

```

// The method responsible for the user-to-user fund transfer

Η μέθοδος λαμβάνει το αντικείμενο της βιβλιοθήκης web3 και τους λογαριασμούς του πορτοφολιού από τον component πατέρα του (App component) με την χρήση των props (τρόπος επικοινωνίας μεταξύ εμφωλευμένων components) του React Framework, μετατρέπει το ποσό που παρέχει ο χρήστης στην φόρμα του component από Finney σε Wei και ξεκινάει την εκτέλεση της συναλλαγής. Μετά το πέρας της συναλλαγής, η μέθοδος ενημερώνει το υπόλοιπο της εφαρμογής, εκτελώντας την μέθοδο `BalanceChange(Balance)` του App component, η οποία και αυτή δίνεται στην μέθοδο ως prop.

```

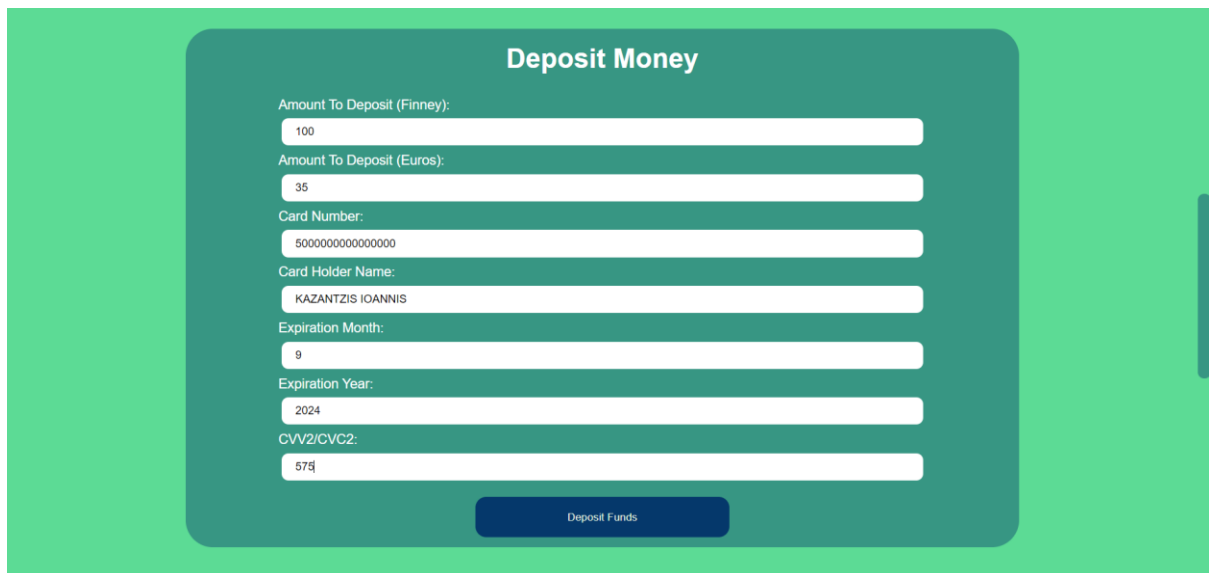
balanceChange = (balance) => {
  this.setState({walletBalance: balance});
}

```

// The method responsible for updating the wallet's balance

Κατάθεση χρημάτων από την χρεωστική του χρήστη στο πορτοφόλι του

Μια επιπλέον σημαντική λειτουργία του ψηφιακού πορτοφολιού GreenWallet, είναι η δυνατότητα κατάθεσης χρημάτων απευθείας από την χρεωστική κάρτα του χρήστη στο πορτοφόλι του. Για την εκτέλεση της λειτουργίας, ο χρήστης χρειάζεται να εισάγει το ποσό που θέλει να μεταφέρει στο πορτοφόλι του, τα απαραίτητα στοιχεία της κάρτας του και να πατήσει το κουμπί «Deposit Funds». Όπως προαναφέρθηκε και στο μέρος «Μεταφορά χρημάτων από το πορτοφόλι του χρήστη στο πορτοφόλι ενός άλλου χρήστη», για την καλύτερη εμπειρία του χρήστη, η γραφική διασύνδεση δεν παγώνει κατά την εκτέλεση της συναλλαγής με σκοπό ο χρήστης να μπορεί να συνεχίσει την χρήση της εφαρμογής. Λόγο της αδυναμίας της εφαρμογής να κάνει πραγματικές συναλλαγές μέσω τράπεζας, επειδή αναπτύσσεται στα πλαίσια της εκπόνησης μιας πτυχιακής εργασίας, στην εφαρμογή η λειτουργία αυτή πραγματοποιείται με την μεταφορά χρημάτων από ένα άλλο λογαριασμό (ο οποίος έχει το ρόλο της χρεωστικής κάρτας) στο λογαριασμό του χρήστη.

A screenshot of a web form titled "Deposit Money" on a green background. The form is a dark green rounded rectangle containing several input fields. The fields are labeled: "Amount To Deposit (Finney):" with value "100", "Amount To Deposit (Euros):" with value "35", "Card Number:" with value "5000000000000000", "Card Holder Name:" with value "KAZANTZIS IOANNIS", "Expiration Month:" with value "9", "Expiration Year:" with value "2024", and "CVV2/CVC2:" with value "575". At the bottom right of the form is a dark blue button labeled "Deposit Funds".

Deposit Money

Amount To Deposit (Finney):
100

Amount To Deposit (Euros):
35

Card Number:
5000000000000000

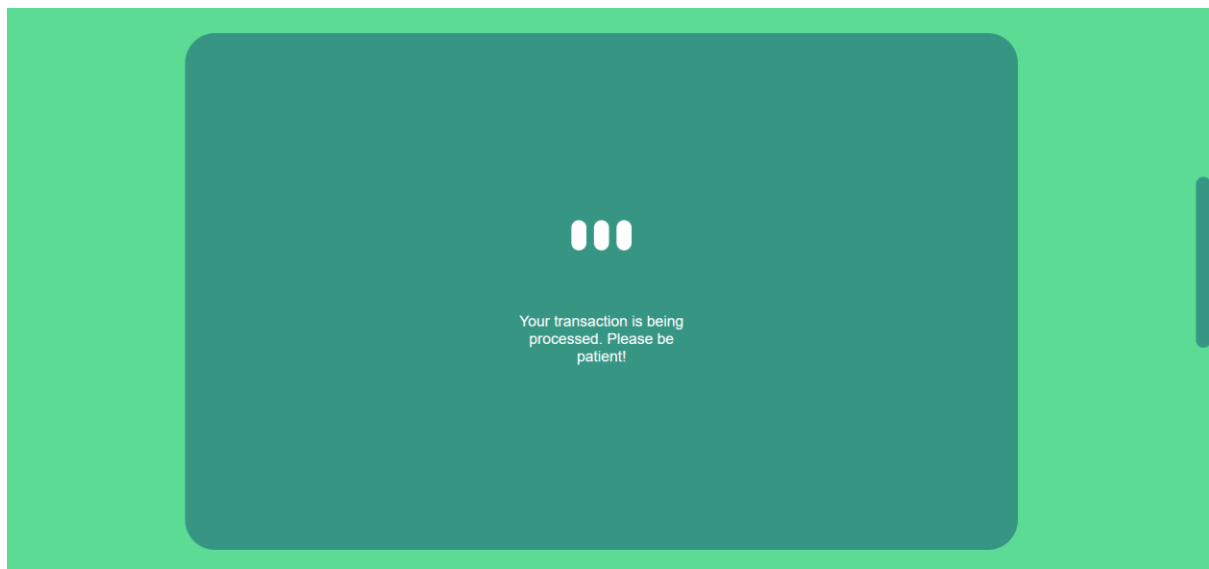
Card Holder Name:
KAZANTZIS IOANNIS

Expiration Month:
9

Expiration Year:
2024

CVV2/CVC2:
575

Deposit Funds



Στις παραπάνω εικόνες φαίνονται η φόρμα που πρέπει να συμπληρωθεί από τον χρήστη για την εκκίνηση της συναλλαγής και το spinner component που εμφανίζεται κατά την διάρκεια εκτέλεσης της συναλλαγής.

Το πεδίο της φόρμας που απεικονίζει το ποσό σε Ευρώ είναι μόνο για ανάγνωση και συμπληρώνεται αυτόματα από την εφαρμογή την στιγμή που ο χρήστης εισάγει το ποσό που θέλει να μεταφέρει σε Finney.

Ο κώδικας για την μεταφορά χρημάτων από την χρεωστική κάρτα στο πορτοφόλι του χρήστη:

```
handleSubmit = async event => {  
  event.preventDefault();
```

```

// Getting web3 instance and accounts
const web3 = this.props.instance;
const accounts = await web3.eth.getAccounts();

// Converting Finney to Wei
let ammountInWei = web3.utils.toWei(this.state.amount.toString(),
                                     'finney');

// Start Loading
this.setState({ loading: true });

// Send Money
await web3.eth.sendTransaction({
  from: accounts[1],
  to: accounts[0],
  value: ammountInWei
});

let walletBalance = await web3.eth.getBalance(accounts[0]);
walletBalance = web3.utils.fromWei(walletBalance, 'finney');

// Setting App cmp state
this.props.onBalanceChange(walletBalance += this.state.amount);

// Stop Loading
this.setState({ loading: false });
}

```

// The method responsible for the deposit functionality

Η μέθοδος της κατάθεσης χρημάτων είναι παρόμοια με αυτή της μεταφοράς χρημάτων σε άλλο λογαριασμό μιας και εκτελούν την ακριβώς «αντίστροφη» λειτουργία.

Ένα ενδιαφέρον κομμάτι κώδικα είναι επίσης η αυτόματη συμπλήρωση του πεδίου “Amount To Deposit (Euros)”:

```

handleChange = event => {
  // Function that saves the new values of the form inputs
  // to the react state
  if(event.target.name === 'cardNumber') {
    this.setState({cardNumber: event.target.value});
  }
  else if(event.target.name === 'chname') {
    this.setState({chname: event.target.value});
  }
  else if(event.target.name === 'expMonth') {
    this.setState({expMonth: event.target.value});
  }
  else if(event.target.name === 'expYear') {
    this.setState({expYear: event.target.value});
  }
  else if(event.target.name === 'CVV2CVC2') {
    this.setState({CVV2CVC2: event.target.value});
  }
  else if(event.target.name === 'amount') {
    this.setState({amount: event.target.value});
  }
}

```

```

        // Auto-filling the Amount To Deposit (Euros) input
        if(event.target.value) {
            const web3 = this.props.instance;
            const amountInWei = web3.utils.toWei(
                event.target.value.toString(),
                'finney');
            const amountInEth = web3.utils.fromWei(amountInWei);
            const amountToDisplay = amountInEth * this.props.ethRate;

            this.setState({amountInEuros: amountToDisplay});
        }
        else {
            this.setState({amountInEuros: ""});
        }
    }
}

```

// The method responsible for storing the form inputs, provided by the user, to React state variables

Οι πρώτες συνθήκες (Conditional Statements) καταχωρούν τις εισόδους του χρήστη από κάθε πεδίο της φόρμας, στην αντίστοιχη μεταβλητή του React state, την στιγμή που αυτός την συμπληρώνει. Σε αυτό διαφέρει η τελευταία συνθήκη, η οποία εκτελείται όταν υπάρχει κάποια αλλαγή στο πεδίο “Amount To Deposit (Finney)”, η οποία συμπληρώνει αυτόματα και το πεδίο “Amount To Deposit (Euros)”. Ο υπολογισμός του ποσού σε ευρώ μπορεί να βρεθεί με την μετατροπή του ποσού, που παρέχει ο χρήστης, από Finney σε Eth και τον πολλαπλασιασμό του με το κόστος μιας μονάδας Eth σε ευρώ. Το κόστος μιας μονάδας Eth σε ευρώ παρέχεται από το Rates component το οποίο συνεχώς ενημερώνεται με την συναλλαγματική του αξία. Πιο συγκεκριμένα η συναλλαγματική αξία του Eth μεταφέρεται από το Rates component στο App component, μέσω της κλήσης της μεθόδου `getEthRate(ethRate)` του App component, η οποία δίνεται σαν prop στο Rates component και από εκεί η αξία του Eth, η οποία έχει καταχωρηθεί σε μια μεταβλητή του React state δίνεται ως prop στο Deposit component. Αυτή η διαδικασία είναι απαραίτητη επειδή τα Rates και Deposit components δεν συνδέονται άμεσα μεταξύ τους, είναι όμως και τα δύο παιδιά του App component

```

<Rates onEthRateChange={this.getEthRate} />

```

// Passing the getEthRate method as a prop to the Rates component

```

getEthRate = (ethRate) => {
    this.setState({ethRate: ethRate});
}

```

// The getEthRate method

```

<Deposit instance={this.state.web3instance}
    onBalanceChange={this.balanceChange}
    ethRate={this.state.ethRate} />

```

// Passing the Eth rate as a prop to the Deposit component

Πληρωμές σε καταστήματα που υποστηρίζουν τις πληρωμές μέσω του Ethereum Blockchain

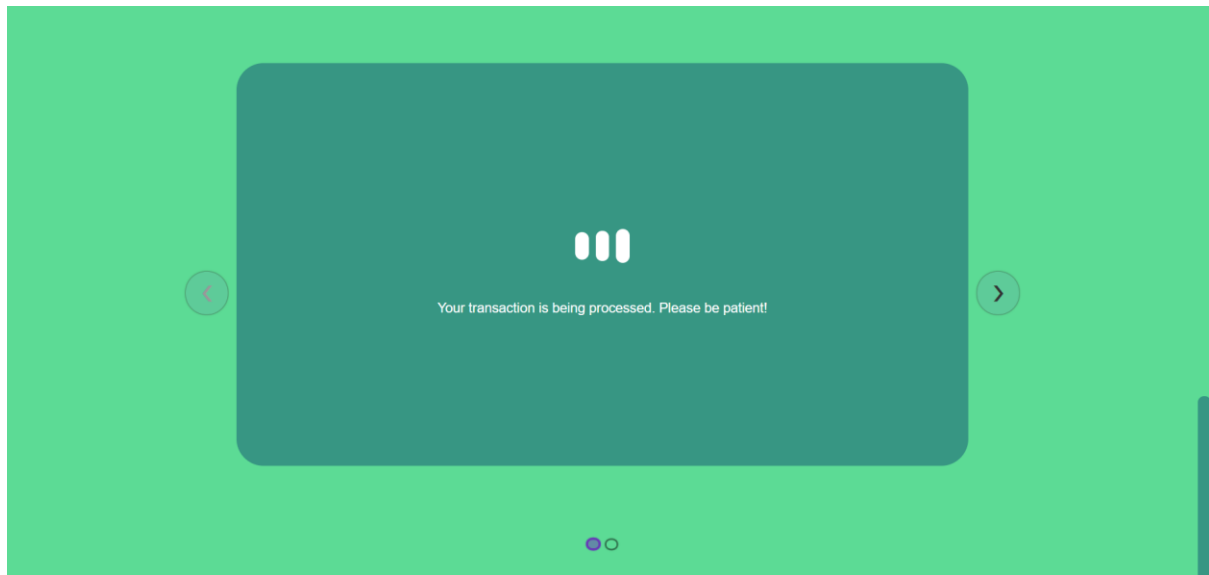
Η τελευταία δυνατότητα που προσφέρει το GreenWallet είναι η πληρωμή προϊόντων σε καταστήματα τα οποία υποστηρίζουν τις πληρωμές μέσω του Ethereum Blockchain και πιο συγκεκριμένα με την μεταφορά χρημάτων στο αντίστοιχο smart contract του μαγαζιού (οι βασικές μέθοδοι του smart contract εξηγούνται στο αντίστοιχο κομμάτι της εργασίας).

Ο χρήστης έχει τις παρακάτω δυνατότητες:

- Να αποθηκεύσει ένα μαγαζί, συμπληρώνοντας τα απαραίτητα στοιχεία της φόρμας που ζητούνται και να πατήσει το κουμπί «Save Contract». Μετά την επιτυχή αποθήκευση, ένα κουμπί με το όνομα του μαγαζιού εμφανίζεται στην γραφική διασύνδεση της εφαρμογής. Τα στοιχεία που ζητούνται είναι:
 - ο Το όνομα του μαγαζιού (Name), το οποίο μπορεί να είναι οτιδήποτε θέλει ο χρήστης
 - ο Η διεύθυνση του smart contract (Contract Address), η οποία παρέχεται από το μαγαζί στον χρήστη
 - ο Το όνομα της μεθόδου που υλοποιεί την πληρωμή (Payable Function), η οποία παρέχεται από το μαγαζί στον χρήστη
- Να πληρώσει σε ένα μαγαζί, το οποίο έχει ήδη αποθηκεύσει, συμπληρώνοντας το ποσό που θέλει να μεταφέρει και να πατήσει το κουμπί με το όνομα του μαγαζιού.
- Να πληρώσει σε ένα μαγαζί το οποίο δεν είναι ήδη αποθηκευμένο αλλά και δεν θέλει να το αποθηκεύσει. Σε αυτή την περίπτωση ο χρήστης πρέπει να συμπληρώσει όλα τα πεδία της φόρμας, εκτός από το όνομα, και να πατήσει το κουμπί «Submit».

Η λειτουργία της πληρωμής σε κατάστημα αλλάζει την κατάσταση του Blockchain επομένως χρειάζεται μερικά δευτερόλεπτα για την ολοκλήρωση της, ακριβώς όπως και στις λειτουργίες της κατάθεσης και της μεταφοράς χρημάτων. Κατά την διάρκεια της εκτέλεσης της συναλλαγής ένα spinner component εμφανίζεται στην θέση της φόρμας.

The screenshot shows a mobile application interface with a green background. In the center is a dark green rounded rectangle titled "Transfer Money To Contract". At the top of this rectangle is a dark blue button labeled "Coffee Story". Below it are four white input fields with labels: "Amount:", "Name:", "Contract Address:", and "Payable Function:". At the bottom of the rectangle are two dark blue buttons: "Submit" and "Save Contract". On the left and right sides of the rectangle are circular navigation buttons with left and right arrows. At the bottom center of the screen, outside the main form, are two small circles, one purple and one white.



Στην παρακάτω εικόνα φαίνονται οι πιο πρόσφατες συναλλαγές που έγιναν στο smart contract και το υπόλοιπο του (0.28900000000000112 Ether), πριν πληρώσει ο χρήστης. Το website που χρησιμοποιείται είναι το (<https://rinkeby.etherscan.io/>):

The screenshot shows the Etherscan interface for the Rinkeby Testnet Network. The top navigation bar includes the Etherscan logo, a search bar, and links for Home, Blockchain, Tokens, and Misc. The main content area displays the 'Contract Overview' for the address 0x6027921cc511D3cAEF3BF08efCbE6045251A7cA2. The balance is shown as 0.28900000000000112 Ether. The 'More Info' section shows 'My Name Tag' as 'Not Available' and 'Contract Creator' as 0xb34bbf8c3e8994078... at txn 0x96bb9a559a8f24a63... The 'Transactions' tab is selected, showing a list of the latest 25 transactions from a total of 45. The table includes columns for Txn Hash, Block, Age, From, To, Value, and [Txn Fee].

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0xd37ec7ab7354a6f04...	7061841	48 mins ago	0xb34bbf8c3e8994078...	0x6027921cc511d3cae...	0.009 Ether	0.000028679
0xda6be8904b63d6865...	7046141	2 days 18 hrs ago	0xb34bbf8c3e8994078...	0x6027921cc511d3cae...	0.01 Ether	0.000028679
0xfdd586b0ba0f1c147f...	7045913	2 days 19 hrs ago	0xb34bbf8c3e8994078...	0x6027921cc511d3cae...	0.01 Ether	0.000028679

Στην επόμενη εικόνα φαίνεται η κατάσταση του smart contract μετά την πληρωμή του χρήστη. Το νέο υπόλοιπο είναι 0.38900000000000112 Ether. Εμφανίζεται επίσης και η συναλλαγή που οδήγησε στην αύξηση του υπολοίπου.

The screenshot shows the Etherscan interface for a contract on the Rinkeby Testnet. The contract address is 0x6027921cc511D3cAEF3BF08efCbE6045251A7cA2. The contract overview shows a balance of 0.3890000000000112 Ether. The 'More Info' section shows 'My Name Tag' as 'Not Available' and 'Contract Creator' as '0xb34bbf8c3e8994078... at txn 0x96bb9a559a8f24a63...'. The 'Transactions' tab is active, showing a list of transactions. The table has columns: Txn Hash, Block, Age, From, To, Value, and [Txn Fee].

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0xe25947dbb921dd50e...	7062079	10 secs ago	0xb34bbf8c3e8994078...	0x6027921cc511d3cae...	0.1 Ether	0.000028679
0xd37ec7ab7354a6f04...	7061841	59 mins ago	0xb34bbf8c3e8994078...	0x6027921cc511d3cae...	0.009 Ether	0.000028679
0xda6be8904b63d6865...	7046141	2 days 18 hrs ago	0xb34bbf8c3e8994078...	0x6027921cc511d3cae...	0.01 Ether	0.000028679

Η μέθοδος η οποία καλείται με το πάτημα το κουμπιού «Save Contract» δημιουργεί ένα JSON αντικείμενο με τα στοιχεία της φόρμας, το οποίο είναι ήδη αποθηκευμένα σαν μεταβλητές του React state και έπειτα κάνει ένα POST αίτημα στον server με σκοπό να κληθεί η μέθοδος του b/e που δημιουργεί μια νέα καταχώρηση στην βάση για το κατάστημα. Αν το αίτημα είναι επιτυχές τότε καλείται η μέθοδος `getExistingContracts()`, η οποία κάνει ένα GET αίτημα στον server και επιστέφει εκ νέου όλα τα καταστήματα αποθηκευμένα από τον χρήστη. Για κάθε ένα από τα αποθηκευμένα καταστήματα δημιουργείται ένα κουμπί το οποίο εμφανίζεται στην γραφική διασύνδεση και επιτρέπει στο χρήστη να χρησιμοποιήσει ξανά το αποθηκευμένο κατάστημα. Αν το αίτημα αποτύχει τότε η μέθοδος ενημερώνει την μεταβλητή του React state που υποδεικνύει αν υπάρχει κάποιο πρόβλημα στην εφαρμογή.

```
handleSubmit = async event => {
  event.preventDefault();
  // Adding new Contract

  const params = {
    name: this.state.name,
    functionName: this.state.functionName,
    address: this.state.address,
    token: this.props.token
  }

  const response = await axios.post(
    'https://green-wallet.herokuapp.com/updateAddContract/',
    JSON.stringify(params));

  if(response.status === 200) {
    await this.getExistingContracts();
  }
  else {
    this.setState({ error: true });
  }
}
```

// The method responsible for saving new contracts

```

getExistingContracts = async () => {
  // Getting the exchange rates from the Coinlayer API

  const headers = {'content-type':'application/json'}
  const params = {
    token: this.props.token
  }

  const response = await axios.get(
    'https://green-wallet.herokuapp.com/getContracts/',
    {headers, params});

  if(response.status === 200) {
    const existingContracts =
      Object.entries(response.data).map(([key, value]) => {
        return <button
          key={value.Address}
          onClick={e => this.submitTransactionButton(e, value)}>
          {value.Name}
        </button>
      ));

    this.setState({ existingContracts: existingContracts});
  }
  else {
    this.setState({ error: true });
  }
}

```

// The method responsible for retrieving the saved contracts

Η μέθοδος η οποία καλείται με το πάτημα ενός κουμπιού που εκπροσωπεί ένα από τα αποθηκευμένα καταστήματα δέχεται αρχικά ως props το web3 αντικείμενο και τον λογαριασμό του χρήστη, από το App component, τα οποία είναι απαραίτητα για την εκτέλεση της συναλλαγής. Για να είναι δυνατή η κλήση κάποιας μεθόδου του smart contract πρέπει να δημιουργηθεί ένα αντικείμενο του. Για την δημιουργία του αντικειμένου είναι απαραίτητη η διεύθυνση του smart contract και το ABI (Application Binary Interface) του. Το ABI είναι ένα αντικείμενο JSON το οποίο περιγράφει το smart contract, τις λειτουργίες του και είναι ο βασικός τρόπος διάδρασης μαζί του (περισσότερες πληροφορίες στο <https://solidity.readthedocs.io/en/v0.7.0/abi-spec.html>). Η διεύθυνση του smart contract είναι γνωστή και δίνεται σαν όρισμα ως τμήμα του τοπικού αντικειμένου *value* στο οποίο είναι αποθηκευμένες όλες οι πληροφορίες του εκάστοτε καταστήματος, ενώ το ABI μπορεί να γίνει γνωστό με μια κλήση σε ένα από τα APIs του Etherscan (<https://etherscan.io/apis#contracts>) εφόσον το smart contract είναι επικυρωμένο (Η διαδικασία της επικύρωσης του κώδικα του smart contract περιγράφεται σε άλλο κομμάτι της εργασίας). Μετά την δημιουργία του αντικειμένου του smart contract, μπορούμε να καλέσουμε την μέθοδο που υλοποιεί την πληρωμή, της οποίας το όνομα είναι γνωστό. Η μέθοδος της πληρωμής δέχεται ως όριασμα τον λογαριασμό από τον οποίο στέλνονται τα χρήματα και το ποσό που θα αποσταλεί. Μετά την επιτυχή εκτέλεση της συναλλαγής η μέθοδος ενημερώνει το υπόλοιπο του λογαριασμού με τον ίδιο τρόπο όπως στις λειτουργίες της κατάθεσης και της μεταφοράς χρημάτων.


```

submitTransactionButton = async (event, value) => {
  event.preventDefault();

  // Start Loading
  this.setState({ loading: true });

  // Getting the web3 instance and the account list
  // from the props passed by the parent component
  const web3 = this.props.instance;
  let accounts = this.props.accounts;

  // Getting ABI from etherscan so we dont have to
  // have every contract source code locally
  const headers = {'content-type': 'application/json'}
  const response = await axios.get(
    `https://apirinkeby.etherscan.io/api
    ?module=contract&action=getabi&address=${value.Address}
    &apikey=${this.GETABIKEY}`, {headers});

  // Creating the contract instance
  const contractInstance = new web3.eth.Contract(
    JSON.parse(response.data.result),
    value.Address
  );

  // Converting Finney -> Wei
  let ammountInWei = web3.utils.toWei(this.state.amount.toString(),
    'finney');

  // Sending Wei
  let options = {
    from: accounts[0],
    value: ammountInWei
  }
  await contractInstance.methods[value.FunctionName]()
    .send(options);

  // Get Wallet Balance
  let walletBalance = await web3.eth.getBalance(accounts[0]);
  walletBalance = web3.utils.fromWei(walletBalance, 'finney');

  // Setting App cmp state
  this.props.onBalanceChange(walletBalance += this.state.amount);

  // Stop Loading
  this.setState({ loading: false });
}

```

//The method responsible for the transfer of money to an already stored smart contract

Ο κώδικας της μεθόδου η οποία υλοποιεί την πληρωμή όταν ο χρήστης δεν θέλει να αποθηκεύσει το κατάστημα είναι πανομοιότυπος με τον κώδικα της πληρωμής σε ήδη αποθηκευμένο κατάστημα. Η διαφορά είναι ότι κώδικας πρέπει να πάρει την διεύθυνση του smart contract από την μεταβλητή *address* του React state του component.

```

submitTransaction = async event => {
  event.preventDefault();

  // Start Loading
  this.setState({ loading: true });

  // Getting the web3 instance and the account list
  // from the props passed by the parent component
  const web3 = this.props.instance;
  let accounts = this.props.accounts;

  // Getting ABI from etherscan so we dont have to
  // have every contract source code locally
  const headers = {'content-type': 'application/json'}
  const response = await axios.get(
    `https://apirinkeby.etherscan.io/api
    ?module=contract&action=getabi&address=${this.state.address}
    &apikey=${this.GETABIKEY}`, {headers});

  // Creating the contract instance
  const contractInstance = new web3.eth.Contract(
    JSON.parse(response.data.result),
    this.state.address
  );

  // Converting Finney -> Wei
  let ammountInWei = web3.utils.toWei(this.state.amount.toString(),
    'finney');

  // Sending Wei
  let options = {
    from: accounts[0],
    value: ammountInWei
  }
  await contractInstance.methods[this.state.functionName]()
    .send(options);

  // Get Wallet Balance
  let walletBalance = await web3.eth.getBalance(accounts[0]);
  walletBalance = web3.utils.fromWei(walletBalance, 'finney');

  // Setting App cmp state
  this.props.onBalanceChange(walletBalance += this.state.amount);

  // Stop Loading
  this.setState({ loading: false });
}

```

//The method responsible for the transfer of money to a smart contract that the user doesn't want to store

Δημιουργία και deployment του smart contract

Η δημιουργία των smart contracts μπορεί να γίνει στο online εργαλείο Remix (<https://remix.ethereum.org/>). Το Remix είναι ένα πολύ δυνατό εργαλείο μιας και μπορούμε να δημιουργήσουμε, να κάνουμε compile, να κάνουμε deploy σε ένα τοπικό blockchain και να ελέγξουμε την λειτουργικότητα ενός καινούργιου smart contract.

Όπως προαναφέρθηκε, κάθε smart contract με το οποίο επικοινωνεί η εφαρμογή GreenWallet αποτελεί ένα κατάστημα το οποίο υποστηρίζει τις συναλλαγές μέσω του Ethereum Blockchain.

Ένα ενδεικτικό smart contract μπορεί να έχει τις παρακάτω μεθόδους:

- **Constructor:** Εκτελείται κατά το deployment του smart contract και αρχικοποιεί την μεταβλητή **owner** με την διεύθυνση που χρησιμοποιήθηκε για το deployment. Η μεταβλητή **owner** είναι πολύ σημαντική για την ασφάλεια του smart contract, εφόσον κάποιες μέθοδοι πρέπει να μπορούν να καλεστούν μόνο από τον ιδιοκτήτη.
- **DestroyContract:** Η μέθοδος αυτή καταστρέφει το smart contract, ώστε να μην μπορεί να χρησιμοποιηθεί. Το υπολειπόμενο ποσό που βρίσκεται στο smart contract μεταφέρεται αυτόματα στον ιδιοκτήτη. Μπορεί να καλεστεί μόνο από τον ιδιοκτήτη.
- **MakePayment:** Η μέθοδος αυτή υλοποιεί την μεταφορά των χρημάτων στο smart contract. Πριν την πρόσθεση του εισερχόμενου ποσού στο υπόλοιπο του smart contract, γίνεται έλεγχος αν η πράξη θα οδηγήσει σε υπερχειλίση.
- **WithdrawPayments:** Η μέθοδος αυτή επιτρέπει στον ιδιοκτήτη να μεταφέρει το υπόλοιπο του smart contract στον λογαριασμό του. Μπορεί να καλεστεί μόνο από τον ιδιοκτήτη.

```
pragma solidity >=0.4.21 <0.7.0;

contract scooterTransactions {

    uint public totalBalance;
    address payable owner;

    // Stores the address of the person who deployed
    // the contract
    constructor() public {
        owner = msg.sender;
    }

    // Destroys the contract and sends all the funds
    // to the owner
    function destroyContract() public {
        require(msg.sender == owner, 'Only the owner can destroy the
            smart contract!');

        selfdestruct(owner);
    }

    // Makes a payment to the smart contract
```

```

function makePayment() public payable {
    assert(totalBalance + msg.value >= totalBalance);
    totalBalance += msg.value;
}

// Sends all the payments to the owner
function withdrawPayments() public {
    require(msg.sender == owner, 'Only the owner can withdraw payments!');

    owner.transfer(totalBalance);
    totalBalance = 0;
}
}

```

Όταν το smart contract είναι έτοιμο για χρήση μπορούμε να το μεταφέρουμε τοπικά στο project και από εκεί να το κάνουμε deploy στο αντίστοιχο blockchain. Η εφαρμογή χρησιμοποιεί το Rinkeby testnet.

Για λόγους ευκολίας γίνεται χρήση του (<https://www.trufflesuite.com/>).

Με την δημιουργία ενός καινούργιου Truffle project, με την εντολή `truffle init`, έχουμε αρχικά στον φάκελο `contracts/` μόνο ένα smart contract, το `Migrations.sol`. Ο φάκελος `contracts/` περιέχει όλα τα smart contracts τα οποία θέλουμε να κάνουμε compile/deploy και έχουν επέκταση `.sol`

Βήματα για να κάνουμε compile/deploy ένα smart contract:

1. Δημιουργία/μεταφορά του smart contract στον φάκελο `contracts/`
2. Εκτέλεση της εντολής `truffle compile`
3. Δημιουργία ενός καινούργιου αρχείου JavaScript στον φάκελο `Migrations/`, ο οποίος περιέχει απαραίτητα αρχεία για το deployment. Το αρχείο που χρησιμοποιούμε εισάγει μια αφαίρεση του smart contract (artifact) και το κάνει deploy στο blockchain. Παραδείγματος χάριν για το smart contract `scooterTransactions.sol`:

```

var scooterTransactions =
artifacts.require("./scooterTransactions.sol");

module.exports = function(deployer) {
    deployer.deploy(scooterTransactions);
};

```

4. Καθορισμός του blockchain στο οποίο θα γίνει το deployment, κάνοντας μικρές αλλαγές στο αρχείο `truffle-config.js`. Στο αρχείο γίνεται η αρχικοποίηση ενός provider (περισσότερες πληροφορίες στο κομμάτι «Αρχικοποίηση του πορτοφολιού») και η εισαγωγή της καινούργιας εγγραφής Rinkeby (network id: 4) στο networks JSON. Το αρχείο μετά τις αλλαγές περιέχει τον παρακάτω κώδικα:

```

const path = require("path");
const HDWalletProvider = require("@truffle/hdwallet-provider");
const fs = require('fs');

let secrets;

```

```
if (fs.existsSync('secrets.json')) {
  secrets = JSON.parse(fs.readFileSync('secrets.json', 'utf8'));
}

module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
  contracts_build_directory: path.join(__dirname,
                                         "client/src/contracts"),
  networks: {
    develop: {
      port: 8545
    },
    rinkeby: {
      provider: new HDWalletProvider(secrets.mnemonic,
                                     'https://rinkeby.infura.io/v3/'+secrets.infuraApiKey),
      network_id: '4'
    }
  }
};
```

5. Εκτέλεση της εντολής `truffle migrate -network rinkeby`

Heroku deployment

Το τελευταίο κομμάτι στην δημιουργία της εφαρμογής είναι το deployment στην πλατφόρμα Heroku, ώστε να είναι δυνατή η πρόσβαση από οποιοδήποτε χρήστη. Ο σύνδεσμος στον οποίο μπορεί να βρεθεί η εφαρμογή GreenWallet είναι ο (<https://green-wallet-frontend.herokuapp.com/>).

Για να δουλέψει σωστά η εφαρμογή πρέπει να γίνει το deployment των τριών βασικών στοιχείων της εφαρμογής:

- Front-End
- Back-End
- Βάση δεδομένων

Front-End Deployment

Βήματα για το deployment του f/e:

1. Εγκατάσταση του Heroku CLI
2. Εκτέλεση της εντολής `heroku login`, για την σύνδεση στον λογαριασμό του προγραμματιστή στην πλατφόρμα Heroku
3. Αρχικοποίηση ενός git repository στον φάκελο με τον κώδικα του f/e, με την εντολή `git init`
4. Εκτέλεση της εντολής `git add .` ή `git add [Αρχεία]`
5. Εκτέλεση της εντολής `git commit -m "[Μήνυμα περιγραφής]"`
6. Δημιουργία μιας νέας εφαρμογής από την γραφική διασύνδεση της πλατφόρμας Heroku
7. Εκτέλεση της εντολής `heroku git:remote -a [Όνομα της εφαρμογής που δημιουργήθηκε προηγουμένως στην γραφική διασύνδεση της πλατφόρμας]`
8. Εκτέλεση της εντολής `git push heroku master`
9. Διόρθωση πιθανών προβλημάτων

Back-End Deployment

Βήματα για το deployment του b/e:

1. Εγκατάσταση του πακέτου της Python 3 gunicorn (<https://gunicorn.org/>) με την εκτέλεση της εντολής `pip3 install gunicorn`
2. Εκτέλεση της εντολής `pip3 freeze > requirements.txt` για την δημιουργία ενός αρχείου με όλα τα εγκαταστημένα πακέτα του project. Το αρχείο μπορεί να περιέχει πακέτα τα οποία δεν χρησιμοποιούνται και δημιουργούν σφάλματα κατά το deployment. Σε αυτή την περίπτωση οι εγγραφές αυτές πρέπει να σβηστούν χειροκίνητα.
3. Δημιουργία του Procfile το οποίο περιέχει την εντολή `web: gunicorn app:[Όνομα του αρχείου που τρέχει τον flask development server χωρίς την επέκταση .py]`.

4. Δημιουργία μιας νέας εφαρμογής από την γραφική διασύνδεση της πλατφόρμας Heroku
5. Εκτέλεση της εντολής `heroku login`
6. Εκτέλεση της εντολής `git init`
7. Εκτέλεση της εντολής `heroku git:remote -a` [Όνομα της εφαρμογής που δημιουργήθηκε προηγουμένως στην γραφική διασύνδεση της πλατφόρμας]
8. Εκτέλεση της εντολής `git add .` ή `git add [Αρχεία]`
9. Εκτέλεση της εντολής `git commit -m "[Μήνυμα περιγραφής]"`
10. Εκτέλεση της εντολής `git push heroku master`
11. Διόρθωση πιθανών προβλημάτων

Database Deployment

Η δημιουργία μιας βάσης δεδομένων στο Heroku μπορεί να γίνει εύκολα από την γραφική διασύνδεση της πλατφόρμας. Αμέσως μετά την δημιουργία, ο προγραμματιστής μπορεί να αντικαταστήσει το αλφαριθμητικό της σύνδεσης της βάσης δεδομένων στον κώδικα της εφαρμογής.

Χρήσιμες πληροφορίες για την δοκιμή της εφαρμογής

Στοιχεία χρήστη

Username: root

Password: pamak2020

Διεύθυνση του πορτοφολιού του χρήστη root

0xb34BbF8c3e899407854F76831fF5b85f60eE44F3

Διεύθυνση του πορτοφολιού άλλου χρήστη φτιαγμένου για την δοκιμή

0x000137cF38e05d65393440EBC92aE6a67ed1a0c2

Διεύθυνση smart contract μαγαζιού για δοκιμή

0x6027921cc511D3cAEF3BF08efCbE6045251A7cA2

Το όνομα της μεθόδου του smart contract που υλοποιεί την πληρωμή

makePayment