Node.js: implementare i pagamenti con Authorize.Net in ExpressJS

In questo articolo vedremo come implementare i pagamenti con Authorize.Net in ExpressJS.

Possiamo creare un account sulla sandbox di Authorize.Net per ottenere le credenziali di accesso alle API.

Salviamo le nostre credenziali in un file dedicato.

```
'use strict';

module.exports = {
    loginId: '',
    transactionKey: ''
};
```

Quindi installiamo il modulo NPM ufficiale.

```
npm install authorizenet --save
```

A differenza di altre soluzioni, nel caso di Authorize.Net i pagamenti tramite carta di credito prevedono che il nostro frontend raccolga i dati essenziali di una carta, ossia:

1. Numero della carta di credito (senza spazi).
2. Codice CVV.
3. Data di scadenza (nel formato `mmaa`).

L'importo da pagare è in formato decimale ma l'SDK di Authorize.Net lo elabora come stringa decimale, quindi non è necessario usare il metodo

`parseFloat()` prima di effettuare la richiesta.

Possiamo creare una funzione di utility per la validazione dei dati inviati.

```
'use strict';

const validator = require('validator');

module.exports = {
    validateForm(req) {
        const { cc, cvv, expire, amount } = req.body;

        const errors = [];

        if(!validator.isCreditCard(cc)) {
            errors.push({
                param: 'cc',
                msg: 'Invalid credit card number.'
            });
        }

        if(!/^\d{3}$/.test(cvv)) {
            errors.push({
                param: 'cvv',
                msg: 'Invalid CVV code.'
            });
        }

        if(!/^\d{4}$/.test(expire)) {
            errors.push({
                param: 'expire',
                msg: 'Invalid expiration date.'
            });
        }
```

```
        if(!validator.isDecimal(amount)) {
            errors.push({
                param: 'amount',
                msg: 'Invalid amount.'
            });
        }

        return errors;
    }
};
```

Usiamo il modulo *validator* per la validazione del numero della carta di credito. Nella sandbox si devono usare numeri fittizi, come ad esempio 4242424242424242.

Ora possiamo di fatto creare la route per l'elaborazione del form.

```
'use strict';

const express = require('express');
const { loginId, transactionKey } =
require('./config');
const ApiContracts =
require('authorizenet').APIContracts;
const ApiControllers =
require('authorizenet').APIControllers;
const SDKConstants =
require('authorizenet').Constants;
const app = express();
const port = process.env.PORT || 3000;
const { validateForm } = require('./lib');

app.post('/checkout', (req, res) => {
```

```
    const validationErrors = validateForm(req);

    if(validationErrors.length > 0) {
        res.json({ errors: validationErrors });
        return;
    }

    // Richiesta
});

app.listen(port);
```

La richiesta che da il via alla transazione può avvenire solo dopo la validazione dei dati.

Tale richiesta, relativamente elaborata, si divide in due fasi. Nella prima viene effettuato il setup delle classi usate dal modulo per creare la transazione.

```
const { cc, cvv, expire, amount } = req.body;

    const merchantAuthenticationType = new
ApiContracts.MerchantAuthenticationType();
    merchantAuthenticationType.setName(loginId);

merchantAuthenticationType.setTransactionKey(transact
ionKey);

    const creditCard = new
ApiContracts.CreditCardType();
    creditCard.setCardNumber(cc);
    creditCard.setExpirationDate(expire);
    creditCard.setCardCode(cvv);
```

```
    const paymentType = new
ApiContracts.PaymentType();
    paymentType.setCreditCard(creditCard);

    const transactionSetting = new
ApiContracts.SettingType();

transactionSetting.setSettingName('recurringBilling')
;
    transactionSetting.setSettingValue('false');

    const transactionSettingList = [];
    transactionSettingList.push(transactionSetting);

    const transactionSettings = new
ApiContracts.ArrayOfSetting();

transactionSettings.setSetting(transactionSettingList
);

    const transactionRequestType = new
ApiContracts.TransactionRequestType();

transactionRequestType.setTransactionType(ApiContract
s.TransactionTypeEnum.AUTHCAPTURETRANSACTION);
    transactionRequestType.setPayment(paymentType);
    transactionRequestType.setAmount(amount);

transactionRequestType.setTransactionSettings(transac
tionSettings);

    const createRequest = new
ApiContracts.CreateTransactionRequest();
```

```
createRequest.setMerchantAuthentication(merchantAuthe
nticationType);

createRequest.setTransactionRequest(transactionReques
tType);
```

Nella seconda, viene effettuata la richiesta remota e vengono gestiti i risultati restituiti dalle API di Authorize.Net.

```
const ctrl = new
ApiControllers.CreateTransactionController(createRequ
est.getJSON());

    ctrl.execute(() => {
        const apiResponse = ctrl.getResponse();
        const response = new
ApiContracts.CreateTransactionResponse(apiResponse);

        if(response !== null) {
            if(response.getMessages().getResultCode()
=== ApiContracts.MessageTypeEnum.OK) {

if(response.getTransactionResponse().getMessages()
!== null) {
                    res.json({ success: 'Transaction
was successful.' });
                } else {

if(response.getTransactionResponse().getErrors() !==
null) {
                        let code =
response.getTransactionResponse().getErrors().getErro
r()[0].getErrorCode();
```

```
                        let text =
response.getTransactionResponse().getErrors().getErro
r()[0].getErrorText();
                        res.json({
                            error: `${code}: ${text}`
                        });
                    } else {
                        res.json({ error:
'Transaction failed.' });
                    }
                }
            } else {
                if(response.getTransactionResponse()
!== null &&
response.getTransactionResponse().getErrors() !==
null){
                    let code =
response.getTransactionResponse().getErrors().getErro
r()[0].getErrorCode();
                    let text =
response.getTransactionResponse().getErrors().getErro
r()[0].getErrorText();
                    res.json({
                        error: `${code}: ${text}`
                    });
                } else {
                    let code =
response.getMessages().getMessage()[0].getCode();
                    let text =
response.getMessages().getMessage()[0].getText();
                    res.json({
                        error: `${code}: ${text}`
                    });
                }
```

```
            }

        } else {
            res.json({ error: 'No response.' });
        }
    });
```

Se la transazione ha avuto successo, riceverete un'e-mail come quella mostrata di seguito.

Codice sorgente

GitHub