```
const request = require('request');
const moment = require('moment');
const async = require('async');
const config = require('../../config');
const Joi = require('joi');
const mcache = require('memory-cache');
const { clear, show, remove } = require('../../services/mcache');
const base64 = require('js-base64').Base64;
const parser = require('xml2js').parseString;
const { Order } = require('../orders/model');
const logger = require('../../services/logger');
const responseHandler = require('../../services/response');
const _ = require('lodash');

exports.wirecardProcessing = (obj, callback) => {
  let req = obj.req
  let res = obj.res
  // wirecard payments are handled specially because of post redirect
  if (req.body.eppresponse) {
    parser(base64.decode(req.body.eppresponse), { explicitArray: false },
function (err, result) {
      if(!err)  {
        console.log(result)
        logger.log('base64', req.headers.correlation_id, null, 'info', { headers:
req.headers, url: '', body: req.body.eppresponse }, null, result);

        if (result.payment.statuses.status instanceof Array &&
result.payment.statuses.status[0].$.code === '201.0000' &&
result.payment['transaction-state'] === 'success' &&
result.payment['transaction-type'] === 'purchase') {
          req.body.transaction_id = result.payment['transaction-id'];
          req.body.request_id = result.payment['request-id'];
          req.body.payment_timestamp = result.payment['completion-time-
stamp'];

          req.body.digits = result.payment['card-token']['masked-account-
number'];
          req.body.bank_approval_code = result.payment['authorization-code'];

          req.body.payment_type = exports.getPaymentType(req.body.digits);

          wirecard = true;
          callback(null);
        } else {
```

```
        //retrieve error code & description
        let errorCode = '';
        let errorDescription = '';
        if (result.payment.statuses.status instanceof Array) {
          errorCode = result.payment.statuses.status[0].$.code;
          errorDescription = result.payment.statuses.status[0].$.description;
        } else {
          errorCode = result.payment.statuses.status.$.code;
          errorDescription = result.payment.statuses.status.$.description;
        }

        req.body.digits = result.payment['card-token']['masked-account-
number'];
        req.body.payment_type = exports.getPaymentType(req.body.digits);

        //retrieve order ID from db for reference
        const validate_id = Joi.validate(req.params.id, Joi.number().positive(),
{ stripUnknown: true })
        if(validate_id.error === null)  {
          req.params.id = validate_id.value
          const query = { 'reservation_id': req.params.id }
          Order.findOneAndUpdate(query, {$set: {'issues': errorCode + ' ' +
errorDescription, 'status': 'failed', 'card_digits': String(req.body.digits),
'payment_type': req.body.payment_type}}, { __v: 0, _id: 0, new: true})
            .exec((err, order) => {
              if(!err)  {
                if(order !== null)  {
                  logger.log('cosmodb', res.req.headers.correlation_id, null, 'info',
query, null, order);
                  responseHandler.paymentUnsuccessful(res, { order_id:
order.order_id });
                } else {
                  logger.log('cosmodb', res.req.headers.correlation_id, null, 'info',
query, null, order);
                  responseHandler.paymentUnsuccessful(res, {});
                }
              } else {
                logger.log('cosmodb', res.req.headers.correlation_id, null, 'error',
query, null, err);
                responseHandler.paymentUnsuccessful(res, {});
              }
            })
        } else { // invaild transaction_id
          voidSACTSales(reservation_id, (err, response) => {
            flagCriticalError(res, reservation_id, transaction_id, 'invalid schema
error')
```

```
              let detailed_error = result.error.details[0].message // detailed error
msg
              responseHandler.invalidFields(res, detailed_error)
            })
          }
        }
      } else { // parsing failed
        logger.log('base64', req.headers.correlation_id, null, 'error', { headers:
req.headers, url: '', body: req.body.eppresponse }, null, err);
        callback(err)
      }
    });
  } else { // not wirecard payment, skip processing and continue
    callback(null);
  }
}

// triggered when payment is made, but something went wrong midway in /
confirm endpoint
exports.flagCriticalError = (res, reservation_id, transaction_id, issue) => {
  const query = { 'reservation_id': reservation_id }
  Order.findOneAndUpdate(query, {$set: {'issues': issue, 'transaction_id':
transaction_id }}, { __v: 0, _id: 0})
    .exec((err, order) => {
      if(!err)  {
        logger.log('cosmodb', res.req.headers.correlation_id, null, 'info', query,
null, order);
      } else {
        logger.log('cosmodb', res.req.headers.correlation_id, null, 'error', query,
null, err);
      }
    })
}

exports.voidSACTSales = (reservation_id, callback) => {
  let xml = '<?xml version="1.0" encoding="utf-8"?><soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/
soap-envelope"><soap12:Body><VoidConfirmSales xmlns="http://
SACT.SentosaOnlineWebService.Webservices/"><vcfmIn><DeviceID>02310</
DeviceID><ReservationID>' + reservation_id + '</ReservationID><Password>'
+ config.ticketing.dry.password + '</
Password><RequestStatus><Status>Successful</Status><Message></
Message></RequestStatus></vcfmIn></VoidConfirmSales></soap12:Body></
soap12:Envelope>'
  request.post({
    url: config.ticketing.dry.url + '/OnlineWebService.asmx?WSDL=',
    body: xml,
    agentOptions: {
      rejectUnauthorized: false
    },
    headers: { 'Content-Type': 'text/xml' }
  }, function(err, response, body)  {
    if(!err)  {
      return callback(null, body)
    } else {
      return callback(err)
    }
  })
}

exports.voidWOTSales = (transaction_id, pincode, callback) => {
  let xml = '<?xml version="1.0" encoding="utf-8"?><soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/
soap-envelope"><soap12:Body><VoidSales xmlns="http://
SACT.WOTWebService.Webservices/"><VoidIn><Password>' +
config.ticketing.wot.password + '</Password><Pincode>' + pincode + '</
Pincode><TransactionID>' + transaction_id + '</
TransactionID><VoidType>SystemVoid</VoidType></VoidIn></VoidSales></
soap12:Body></soap12:Envelope>'
  request.post({
    url: config.ticketing.wot.url + '/OnlineWebService.asmx?WSDL=',
    body: xml,
    agentOptions: {
      rejectUnauthorized: false
    },
    headers: { 'Content-Type': 'text/xml' }
  }, function(err, response, body)  {
    if(!err)  {
      return callback(null, body)
    } else {
      return callback(err)
    }
  })
}

exports.releaseWOT = (transaction_id, pincode, callback) => {
  let xml = '<?xml version="1.0" encoding="utf-8"?><soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/
soap-envelope"><soap12:Body><ReleaseTicket xmlns="http://
```

```
SACT.WOTWebService.Webservices/"><RelIn><Password>' +
config.ticketing.wot.password + '</Password><Pincode>' + pincode + '</
Pincode></RelIn></ReleaseTicket></soap12:Body></soap12:Envelope>'
  request.post({
    url: config.ticketing.wot.url + '/OnlineWebService.asmx?WSDL=',
    body: xml,
    agentOptions: {
      rejectUnauthorized: false
    },
    headers: { 'Content-Type': 'text/xml' }
  }, function(err, response, body)  {
    if(!err)  {
      return callback(null, body)
    } else {
      return callback(err)
    }
  })
}

exports.voidWirecardPayment = (res, payment_timestamp, request_id,
transaction_id, callback) => {
  //10pm daily limit
  let limitTimestamp = moment({ hour: 22 });
  //get time now
  let nowTimestamp = moment();
  //convert UTC to moment
  let paymentTimestamp = moment(payment_timestamp, 'YYYY-MM-
DDTHH:mm:ss.SSSZ');

  //default set to void-purchase
  let transactionType = 'void-purchase';
  let requestId = request_id + '-void';

  //check if payment is made before 10pm & current time is after 10pm
  if (paymentTimestamp < limitTimestamp && nowTimestamp >=
limitTimestamp) {
    transactionType = 'refund-purchase';
    requestId = request_id + '-refund';
  }

  //default config
  let url = config.wirecard.url;
  let username = config.wirecard.username;
  let password = config.wirecard.password;
  let merchantId = config.wirecard.merchant_id;
```

```
//toggle between uat & production
if (config.wirecard.env === 'uat') {
  url = config.wirecard.uat.url;

  //toggle between onus & offus
  if (res.req.query.dbs) {
    username = config.wirecard.uat.onus.username;
    password = config.wirecard.uat.onus.password;
    merchantId = config.wirecard.uat.onus.merchant_id;
  } else {
    username = config.wirecard.uat.offus.username;
    password = config.wirecard.uat.offus.password;
    merchantId = config.wirecard.uat.offus.merchant_id;
  }
} else if (wirecardEnv === 'production') {
  url = config.wirecard.production.url;

  //toggle between onus & offus
  if (res.req.query.dbs) {
    username = config.wirecard.production.onus.username;
    password = config.wirecard.production.onus.password;
    merchantId = config.wirecard.production.onus.merchant_id;
  } else {
    username = config.wirecard.production.offus.username;
    password = config.wirecard.production.offus.password;
    merchantId = config.wirecard.production.offus.merchant_id;
  }
}

const wirecardRequest = {
  url: url + '/payments',
  auth: {
    user: username,
    pass: password
  },
  method: 'POST',
  json: true,
  body: {
    payment: {
      'merchant-account-id': {
        'value': merchantId
      },
      'request-id': requestId,
      'transaction-type': transactionType,
      'parent-transaction-id': transaction_id
    }
    }
  }

  request(wirecardRequest, function(err, response, body) {
    if (!err) {
      if (body.payment['transaction-state'] === 'success') {
        logger.log('wirecard', res.req.headers.correlation_id, null, 'info', { url:
wirecardRequest.url, method: wirecardRequest.method, headers: null, body:
wirecardRequest.body }, null, body);
        callback(null, body);
      } else {
        logger.log('wirecard', res.req.headers.correlation_id, null, 'fatal', { url:
wirecardRequest.url, method: wirecardRequest.method, headers: null, body:
wirecardRequest.body }, null, body);
        responseHandler.voidPaymentUnsuccessful(res);
      }
    } else {
      logger.log('wirecard', res.req.headers.correlation_id, null, 'fatal', { url:
wirecardRequest.url, method: wirecardRequest.method, headers: null, body:
wirecardRequest.body }, null, err);
      responseHandler.voidPaymentUnsuccessful(res);
    }
  });
}

exports.toggleEnv = (environment) => {
  if(environment === 'production') {
    return 'http://ospdirectoriesapp-prod.ospapps.net/'
  } else {
    return 'https://ospdirectoriesapp.azurewebsites.net/'
  }
}

exports.getPaymentType = (digits) => {
  let paymentType = 'Credit Card / Debit Card';

  if (digits.startsWith('4')) {
    paymentType = 'Visa';
  } else if (digits.startsWith('50') || digits.startsWith('51') ||
digits.startsWith('52') || digits.startsWith('53') || digits.startsWith('54') ||
digits.startsWith('55')) {
    paymentType = 'MasterCard';
  }

  return paymentType;
}
```