su-si-ism-354-2018 / bos_api

Branch: master ▾ bos_api / server.js Find file Copy path

barnsza Added Generic API e743dcb 11 days ago

1 contributor

205 lines (165 sloc) 7.47 KB

```
const restify = require('restify');
const MongoClient = require('mongodb').MongoClient;
const jsSHA = require('jssha');
const Passport = require('passport');
const PassportHTTPBearer = require('passport-http-bearer');
const uuidv3 = require('uuid/v3');
const errs = require('restify-errors');
const restify_cors = require('restify-cors-middleware')

const UUID_NAMESPACE = "353ac1c0-cab3-11e8-9211-4f01514d4e40";

(async () => {
  const db = (await MongoClient.connect('mongodb://localhost:27017',
    { useNewUrlParser: true })).db('bos');

  // Simple HMAC based bearer token strategy, not explicitly secure.
  Passport.use(new PassportHTTPBearer.Strategy(async (token, next) => {
    const [hash, username] = token.split('%');

    const user = await db.collection('users').findOne({ username: username });
    if (!user) return next(false);

    const hmac = new jsSHA('SHA-256', 'TEXT');
    hmac.setHMACKey(user['password'], 'TEXT');
    hmac.update(user['username']);
    hmac.update(Date.now().toString(36).substring(0, 4));

    if (hash === hmac.getHMAC('HEX')) return next(null, username);
    return next(false);
  }));

  const cors = restify_cors({
    origins: ['*'],
    allowHeaders: ['Authorization']
  })

  const server = restify.createServer();
  server.pre(restify.pre.sanitizePath());
  server.use(restify.plugins.bodyParser());
  server.pre(cors.preflight)
  server.use(cors.actual)

  server.post('/user', // Create New User
    async (req, res, next) => {
      if (!(req.body.username && req.body.password)) return next(
        new errs.MissingParameterError("Must supply username and password."));

      const id = uuidv3(req.body.username, UUID_NAMESPACE);
      const user = {
        _id: id,
        username: req.body.username,
        password: req.body.password
      };

      const collection = db.collection('users');
      try {
        const new_user = await collection.insertOne(user);
        res.send(new_user);
        next();
      }
      catch(e) {
```

```
        return next(new errs.PreconditionFailedError("Could not create user."));
      }

});

server.post('/apply', // Apply for New Account
  Passport.authenticate('bearer', { session: false }),
  async (req, res, next) => {
    res.send();
    next();
});

server.get('/clients', // Get Clients for Current User
  Passport.authenticate('bearer', { session: false }),
  async (req, res, next) => {
    const collection = db.collection('clients');
    const clients = await collection.find({ usernames: req.user }).toArray();
    res.send(clients);
    next();
});

server.get('/accounts/:client', // Get Clients for Current User
  Passport.authenticate('bearer', { session: false }),
  async (req, res, next) => {
    // Assert Client == User
    const cli_col = db.collection('clients');
    const client = await cli_col.findOne({ usernames: req.user, _id: req.params.client })
    if (!client) return next(new errs.NotAuthorizedError('Not Authorized.'));

    const acc_col = db.collection('accounts');
    const accounts = await acc_col.find({ client: req.params.client }).toArray();
    res.send(accounts);
    next();
});

const get_transactions = // Get Transactions for Account
async (req, res, next) => {
  const acc_col = db.collection('accounts');
  const account = await acc_col.findOne({ _id: req.params.account });

  // Assert Client == User
  const cli_col = db.collection('clients');
  const client = await cli_col.findOne({ usernames: req.user, _id: account.client })
  if (!client) return next(new errs.NotAuthorizedError('Not Authorized.'));

  const search_query = { src: req.params.account, ref: {'$regex': req.params.search} };
  if (req.params.from || req.params.to) { search_query.time = {} };
  if (req.params.from) search_query.time['$gte'] = parseInt(req.params.from);
  if (req.params.to) search_query.time['$lte'] = parseInt(req.params.to);

  const trans_col = db.collection('transactions');
  const transactions = await trans_col.find(search_query).toArray();
  res.send(transactions);
  next();
};

server.get('/transactions/:account/:search',
  Passport.authenticate('bearer', { session: false }), get_transactions);

server.get('/transactions/:account/:search/:from',
  Passport.authenticate('bearer', { session: false }), get_transactions);

server.get('/transactions/:account/:search/:from/:to',
  Passport.authenticate('bearer', { session: false }), get_transactions);

server.post('/transactions', // Create Transaction
  Passport.authenticate('bearer', { session: false }),
  async (req, res, next) => {
    if (!(req.body.type && req.body.src && req.body.dest && req.body.amount && req.body.ref ))
      return next(new errs.MissingParameterError("Missing transaction fields."));

    if (req.body.time && (parseInt(req.body.time) <= Date.now()))
      return next(new errs.PreconditionFailedError("Invalid future transaction."));
```

```
135
136         const acc_col = db.collection('accounts');
137         const account = await acc_col.findOne({ _id: req.body.src });
138
139         // Assert Client == User
140         const cli_col = db.collection('clients');
141         const client = await cli_col.findOne({ usernames: req.user, _id: account.client })
142         if (!client) return next(new errs.NotAuthorizedError('Not Authorized.'));
143
144         if (parseInt(req.body.amount) > account.balance)
145           return next(new errs.PreconditionFailedError("Insufficient balance."));
146
147         const transaction = {
148           _id: uuidv3(`${req.body.src}:${req.body.time || Date.now()}`, UUID_NAMESPACE),
149           time: (parseInt(req.body.time) || Date.now()),
150           type: req.body.type,
151           src: req.body.src,
152           dest: req.body.dest,
153           amount: parseInt(req.body.amount),
154           balance: account.balance - parseInt(req.body.amount),
155           ref: req.body.ref
156         }
157         account.balance = transaction.balance;
158
159         await acc_col.replaceOne({ _id: account._id }, account, { upsert: true });
160         const trans_col = db.collection('transactions');
161         await trans_col.replaceOne({ _id: transaction._id }, transaction, { upsert: true });
162
163         res.send(transaction);
164         next();
165     });
166
167     server.get('/generic/:id', // Get Generic Item by _id, owned by a single user.
168       Passport.authenticate('bearer', { session: false }),
169       async (req, res, next) => {
170         const collection = db.collection('generic');
171         const item = await collection.findOne({ username: req.user, _id: req.params.id })
172         if (!item) return next(new errs.NotAuthorizedError('Not Authorized.'));
173         res.send(item);
174         next();
175     });
176
177     server.post('/generic', // New generic item
178       Passport.authenticate('bearer', { session: false }),
179       async (req, res, next) => {
180         req.body.username = req.user;
181         req.body._id = uuidv3(`${Date.now()}:${req.user}`, UUID_NAMESPACE);
182
183         const collection = db.collection('generic');
184         const item = await collection.replaceOne({ _id: req.body._id }, req.body, { upsert: true });
185
186         res.send(item);
187         next();
188     });
189
190     server.del('/generic/:id', // Delete Generic Item by _id, owned by a single user.
191       Passport.authenticate('bearer', { session: false }),
192       async (req, res, next) => {
193         const collection = db.collection('generic');
194         const item = await collection.findOne({ username: req.user, _id: req.params.id })
195         if (!item) return next(new errs.NotAuthorizedError('Not Authorized.'));
196
197         await collection.deleteOne({ _id: req.params.id })
198
199         res.send(item);
200         next();
201     });
202
203     server.listen(8080, () => { console.log(`Listening: ${server.url}`) });
204 })();
```