Create SDKs for your API

Making APIs easy to use for increased developer efficiency and adaption

Burton Rheutan

@_burtonr

What is an SDK?

- Software Development Kit
  - A kit to enable developing software
- Code to make it easier to run your code
  - Abstract the specifics to allow common patterns to be used
- A library, or package that can be included in other projects that contain pre-defined calls to an API
  - Make 1 or many calls with one line of code in the consumer application

Why Create an SDK For Your API?

- Make it easy to consume without in-depth knowledge
  - Abstract versioning, specific headers, authentication, etc
- Enforce best practices
  - Default parameters, paging, timeouts, and more
- Error handling
  - Use language specific features to present understandable errors
  - What's a 418?!?!
- Nobody likes implementing an HTTP client every time
  - Wrap the boiler plate code in a function/method

Customer API

- Traditional APIs are data-centric
  - Requiring several calls back and forth
  - With obscure ids as part of the path

- To use this API to check an order, you would need to
  - GET -> all customers
  - Find your user to get the ID
  - GET -> all orders
  - Find the ID of the order you're interested in
  - GET -> specific order
  - Do work

```
server.get('/customers', () => {...})
server.post('/customers', () => {...})
server.get('/customers/:id', () => {...})
server.put('/customer/:id', () => {...})
server.del('/customers/:id', () => {...})

server.get('/customers/:id/orders', () => {...})
server.post('/customer/:id/orders', () => {...})
server.get('/customers/:id/orders/:orderId', () => {...})
server.put('/customers/:id/orders/:orderId', () => {...})
server.del('/customers/:id/orders/:orderId', () => {...})
```

Client implementation of Customer API

Direct

```
let https = require('https');

function getCustomerFromStore(custId) {
    let req = https.request({
        host: 'store.business.com',
        path: '/customers/' + custId,
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
        }
    }, function (res) {
        res.setEncoding('utf-8');

        let customer = {};

        res.on('data', function (data) {
            customer = JSON.parse(data);
            success(customer);
        });
    });

    req.end();
}
```

With SDK

```
let storeSDK = require('storeSDK');

function getCustomerFromStore(custId) {
    let customer = storeSDK.getCustomer(custId);
    return customer;
}
```

Behind the scenes...

- Make sane defaults
  - application/json
- Handle expected errors
  - 404, 503, etc
- Prevent mistakes
  - Validate required fields
  - Only known methods/content

```
getCustomer(customerId) {
    if (!customerId || typeof customerId !== 'number') {
        throw new Error('Customer ID must be a number');
    }

    let headers = {
        'Content-Type': 'application/json'
    };

    options.headers = headers;

    let customer = {};

    https.get(options, function(res) {
        if (res.statusCode == 404) {
            return {};
        }

        if (res.statusCode == 503) {
            throw new Error('Customer API currently unavailable');
        }

        res.on('data', (c) => {
            customer = JSON.parse(c);
        })
    }).on('error', (e) => {
        throw new Error('Unexpected error attempting to get customer. Error: ' + e.message);
    });

    req.end();
    return customer;
}
```

Make it intuitive...

- Accept known parameters
  - Is customerId commonly known?
  - getCustomerByName(name)
- Use naming conventions
  - addOrder(order)
  - updateCustomer(customer)
- Lean on the language features
  - Throw errors
  - Default/Optional parameters
  - Strongly typed parameters

```
getCustomerByName(name) {
    https.get(`customerUrl?name=${name}`);
}

addOrder(order) {
    if (!order.Status) {
        order.Status = 'Pending';
    }

    let options = {
        method: 'POST'
    };

    let req = https.request(options, (res) => {
        ...
    });

    req.write(order);
    req.end();
}

updateCustomer(customer) {
    if (!customer.Id) {
        throw new Error("Customer Id is required");
    }

    let options = {
        method: 'PUT'
    };

    let req = https.request(options, (res) => {
        ...
    });

    req.write(customer);
    req.end();
}
```

Adjust to use-cases

- Combine multiple calls
  - getOrdersByCustomer()
- Intelligently handle errors
  - upsertOrder()

```
getOrdersByCustomer(name) {
    let customer = getCustomer(name);
    let orders = [];
    let req = https.get(`customerUrl/${customer.Id}/orders`, (res) => {
        ...
    });
    req.close();
    return orders;
}

upsertOrder(order) {
    try {
        updateOrder(order);
    } catch (err) {
        if (err.message.includes('not found')) {
            createOrder(order);
        } else {
            throw err;
        }
    }
}
```

Benefits of an SDK

- Fast and easy adoption
  - One-line access to API
- Safe and optimized
  - Use sane defaults and limits
- Expand API Functionality
  - Chain calls to make it easy

- Start with the API's language
- Avoid unnecessary dependencies
- Add other languages to meet user demand/popularity
- Backwards compatible

Thank you

Burton Rheutan

@_burtonr