GOOP REST API

I Core principles

The GOOP REST API is both powerful and secure embracing MVC architecture on Express 4(NodeJS). It exposes CRUD operations designed using best practices. MongoDB is the chosen data storage technology and is implemented using Mongoose for object modeling. All API routes are simple and follow the well-known convention two resources - four HTTP verbs.Here are some sample routes starting from the top-level:

Resource	POST create	GET read	PUT update	DELETE delete
/api/v1/classes	Create a new class	List classes	Bulk update classes	Delete all classes
/api/v1/classes/:classId	Error	Show class	Update class if exists	Delete class

II Versioning and simple URLs

As you see versioning is a feature we consider and we stick to using plural nouns for all resources consistently. The API itself is designed to be used without documentation - all routes are intuitive and self-explanatory for application developers and users.

III Partial response and filtering

It doesn't end with the slash, the fun starts under the '?' where we sweep complexity to keep URLs clean. We support partial response and pagination using limit/offset parameters. For example, */api/v1/classes?offset=20&limit=50* will skip the first 20 records and take the next 50. What is more, filtering comes in to reduce valuable network traffic and get developers only what they need. A call to */api/v1/classes?inherits=Car* will return only classes that inherit the class named "Car".

IV Security

Of course users are supported. We've done our best to maximize privacy and secure data using passport.js to authenticate and authorize requests. For instance the following post request: */api/v1/classes* will require a user to log in with their account before they can create a class, similarly a delete request to */api/v1/classes/:classId* will first require a login and then check if the user from the request is the actual owner of the class to be modified. All passwords are properly hashed using best practices in creating salts and generating HMACs.

V Love JSON, embrace HTTP

Finally, currently we support JSON responses which is the preferred format by most developers, but XML is an option in mind for future versions. Better client - server communication is accomplished using suitable HTTP response codes familiar to developers.

VI Conclusion

The source code of the API will soon be published on Github as well as hosted on Heroku. This API is designed for internal use in our SPA application but it isn't tied to it in any way and can be consumed by everything capable of handling HTTP requests and responses.