TADAPI

Software Requirement Specification v0.2

March 7th, 2013

Novacoast

Nibble

Disclaimer

Novacoast™, Inc. makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any expressed or implied warranties of merchantability or fitness for any particular purpose.

Trademarks

The Novacoast name and logo are registered trademarks of Novacoast, Inc. in the United States and other countries. The Novacoast Symbol is a trademark of Novacoast, Inc.

All third-party trademarks are property of their respective owner.

Copyright

Change Control Process

The Change Control Process governs changes to the scope of this project throughout the project's duration. It applies to new components and to enhancements of existing components. A written Change Request communicates any desired changes to this project. It describes the proposed change, the reason for the change, and the effect the change might have on the project. The Novacoast project manager supplies the appropriate Change Management documents.

Both Novacoast and the customer review the Change Request and approve or reject it. Both parties must sign the approval portion of the Change Request to authorize the implementation of any change that affects the project’s scope, schedule, or fee.

Document Change Tracking

Contributors: Andrew Duong, Tong Wu, Jimmy Le, Mark Oparowski, Eron Howard, Renato Untalan, David Parker

Table of Contents

1. Novacoast Contact Information 5
2. Project Overview: The Automated Discovery of Application Programming Interfaces 6
    2.1 Executive Summary 6
        2.1.1 Main Idea 6
        2.1.2 Main Goals 6
        2.1.3 Future Goals 6
        2.1.4 Outcome 6
        2.1.5 Implementation 7
3. Product Requirements 8
    3.1 Major Components 8
    3.2 Product Requirements & Design (PRD) Tables 8
        3.2.1 Sniffer 9
        3.2.2 Processor 9
        3.2.3 Manager 9
        3.2.9 Database 10
        3.2.10 UI 11
        3.2.10.2 In-Application Page Frame 11
        3.2.10.3 Search Display 11
4. Assumptions and Risks 12
    4.1 Assumptions 12
        4.1.1 Server for subversion 12
        4.1.2 AWS account 13
        4.1.3 UI Designer 13
        4.1.4 Initial Reviews 13
        4.1.5 Training 13
        4.1.6 GoMockingbird 13
        4.1.7 Access to Novacoast R&D 13
        4.1.8 Access to Office and Staff 13
        4.1.9 Mobile App Software/Hardware 13
        4.1.10 LinkedIn API 13
        4.1.11 Developer Access 12
    4.2 Risks 13
        4.2.1 SQLite Overburdening 13
        4.2.2 Insufficient Skill Set Among Team Members 13
        4.2.3 Scheduling Conflict 13
        4.2.4 Complexity Underestimated 14
        4.2.5 Scalability 14
5. Support, Platforms, and Localization 15
    5.1 Supported platforms 15
    5.1.1 Supported browsers 15

5.1.2 Supported Mobile Platforms (future) 15
5.2 Localization 15
5.3 Documentation 15
6. Technology Stack 16
6.1 Overview of Technologies Used 16
7. Planning 17
7.1 Client Component - Novacoast 17
7.1.1 Novacoast Resources 18
7.1.2 Development Timeline 18
8. GUI Mock-ups 19
8.1 Landing Page 19
8.2 Discovery: Hidden API Feed* 20
8.3 Discovery: Visible API Feed 21
8.4 API List from Search 22
8.5 API Definition of Searched API 23
I. Change Log 24

1. Novacoast Contact Information

Novacoast Corporate Office
1505 Chapala Street
Santa Barbara, CA 93101
Voice: 805.949.9933
Fac: 805.564.1809

Project Lead
Eron Howard
Voice: 805.453.2885
E-mail: ehoward@novacost.com

Developer
Tong Wu
Voice: 408.600.8331
Email: chronus.mc@gmail.com

Developer
Jimmy Le
Voice: 916.627.9145
Email: meh38p@gmail.com

Developer
Mark Oparowski
Voice: 530.798.3440
Email: mark72x@gmail.com

Developer
Andrew Duong
Voice: 310.529.3561
Email: andyd.iso@gmail.com

2. Project Overview: The Automated Discovery of Application Programming Interfaces

2.1 Executive Summary

2.1.1 Main Idea:

Documentation is one of the most prominent issues for a developer creating or using an application program interface (API). The software developer of the API must create and maintain the document, a process considered by most programmers to be the most tedious part of the software development process. Meanwhile, developers wanting to use the API must hunt down the documentation and hope that it exists, is complete, and is up-to-date. TADAPI (The Automated Discovery of Application Program Interfaces) from Novacoast x Nibble is an automated API discovery web application that produces deliverable API documentation in the most intuitive way: through the simple navigation of websites or applications that use the API calls.

2.1.2 Main Goals:

Develop a web application to discover API functionalities.

2.1.3 Future Goals:

Implement searches for associated API calls.
Implement auditing schemes for traffic monitoring.

2.1.4 Outcome:

Documentation is currently written in two methods, either after the software is developed, or while the software is being developed. The first method requires that the developer create a finished product, then backtrack and tediously write out the API functionality by going through their code and memory. This often yields incomplete and sometimes inaccurate information. The second method requires that the developer take intermittent breaks throughout the coding process to document their code, distracting the coders from their main purpose and thought process. TADAPI simplifies the documentation process and addresses the downside of the two current methods by creating archives for API functionality through an intuitive automated tool. TADAPI runs in the background by scanning HTTP requests, then analyzing the responses in order to create and log API functionality as the user navigates through the web application that uses the API calls. Even non-technical application users can help generate documentation simply by running TADAPI as they navigate through these web applications. TADAPI also makes the maintenance of documentation after API updates extremely simple because it updates its own API database immediately after TADAPI is used on the updated web application. TADAPI is an easy and

foolproof way to comprehensively document API functionality without having to backtrack through code or breaking the development process.

Similarly, writing a web application that uses external API calls requires developers to search up documentation for the API's they wish to use. If they are lucky, they will find up-to-date and accurate listings of the API's functionalities, otherwise, the developer would have to run repetitive tests on the API to determine its functionalities. TADAPI creates an up-to-date archive for API's as they are discovered, but if a certain web application does not have its API functions documented, the developer can run TADAPI in the background and discover its functions by navigating the application instead of tediously running numerous sets of repetitive tests.

TADAPI makes API discovery and documentation easy enough for developers to delegate these same tasks to non-developers.

2.1.5 Implementation

TADAPI is primarily interfaced through a website with the backend utilizing the Ruby programming language. Traffic monitoring is implemented via the PCAP libraries and CouchDB will serve as the database for TADAPI. The front end will utilize Ruby on Rails along with other traditional web technologies such as javascript, CSS, and HTML.

This document is the authoritative blueprint for Novacoast developers and engineers, who will work on the implementation of TADAPI. It includes a development plan, component tables with estimate of hours, UI mockups and storyboards, as well as a development timeline. It excludes the backend, web service, and content uploader components of TADAPI.

3. Product Requirements

3.1 Major Components

The following components are fundamental pieces of the application.

Component Name	Description
3.2.1 Sniffer	The sniffer gathers internet traffic, storing the file path into a database. All capturable packets are accepted as proper input.
3.2.2 Processor	The processor parses the network information in a well-defined manner, and displays the information in a manageable way.
3.2.3 Manager	The main controller of the program. Calls appropriate functions based on user interaction, and controls interoperation between different components of the program. Processed data can be downloaded into html or pdf formats through the manager.
3.2.4 Database	All processed data will have their path on disk be stored in a database for future references. Periodic database backups will be performed to ensure that the data collected will be safe in cases of hardware failure.
3.2.5 User Interface	User interface acts as the front end of the application. It allows for users to easily use the application through an intuitive design. Displayed data can be managed by search/sort functions.

3.2 Product Requirements & Design (PRD) Tables

The following table explains the priority column in the PRD tables outlined within each component.

Priority	Description
1	Mandatory for this release. They are required for initial functionality of the first major release.
2	Increases application functionality and robustness. Will most likely be included in initial release but not absolutely necessary.

3	Adds significant customer value. Stretch targets for initial release.
4	Improves overall functionality, but not vital to initial release.
0	Rejected requirement.

3.2.1 Sniffer

3.2.1.1 Packet Sniffer

Requirement	Description	Hours	Priority	Comments
Sniff Packets 3.2.1.1a	The sniffer gathers internet traffic, storing it into the provided PCAP file. All capturable packets are accepted as proper input.	4	1	
Stop Sniffing 3.2.1.1b	Stops sniffing internet traffic, and stops writing to the PCAP file.	1	1	

3.2.2 Processor

3.2.2.1 Analyze and Process

Requirement	Description	Hours	Priority	Comments
Start Processor 3.2.2.1a	Loads in the processor with processor id, and begins executing the processor	3	1	
Display Processed Data 3.2.2.1b	Display the data after processing the JSON file in a well-defined structure	2	1	
Parse Data 3.2.1.1c	The processor will parse the data from the captured packets in a well-defined manner	8	1	
Build JSON File 3.2.1.1d	The processor will build the parsed data into a JSON file	4	1	

3.2.3 Manager

3.2.3.1 Program controller

Requirement	Description	Hours	Priority	Comments
Initialize PCAP File 3.2.3.1a	The sniffer will initialize and prepare a PCAP file prior to calling the sniffer.	2	1	
Start Sniffer 3.2.3.1b	Call the sniffer and pass in the PCAP file to start the sniffing process	1	1	
Stop Sniffer 3.2.3.1c	Calls the stop sniffer function in the sniffer, ending the internet sniffing and receive a returned JSON file with the parsed PCAP of API calls	1	1	
Start Processor 3.2.3.1d	Start a processor, passing in the processor id to use and the PCAP id to process.	1	1	
Stop Processor 3.2.3.1e	Ends a processor by passing in a processor id, causing the display to change to the landing	2	1	
Download JSON File 3.2.3.1f	Allows user to save API documentation in .html or .pdf formats for offline reference	18	3	
Get session list 3.2.3.1g	Access the database, obtaining a list of session identifiers that have been stored	2	1	
Change Sessions 3.2.3.1h	End current session and link a previous session stored in the database	4	1	

3.2.4 Database

3.2.4.1 Operations on database

Requirement	Description	Hours	Priority	Comments
Auto Backup 3.2.4.1a	The database will periodically backup its' contents to another backup partition	10	3	

Initialize Database 3.2.4.1b	Create efficient database schema.	8	1	
Database Scalability 3.2.4.1c	Able to scale and store large amounts of data.		0	
Automatic Database 3.2.4.1d	Automatically updates old API function documentation when newer ones are discovered.		0	

3.2.5 UI

3.2.5.1 Task Selector

Requirement	Description	Hours	Priority	Comments
Tasks 3.2.5.1a	Switch between API discovery, traffic auditing, or custom mode	4	2	Buttons that switch the view

3.2.5.2 In-Application Page Frame

Requirement	Description	Hours	Priority	Comments
URL Bar 3.2.5.2a	URL bar for specifying API URL	4	3	

3.2.5.3 Search Display

Requirement	Description	Hours	Priority	Comments
Search Bar 3.2.5.3a	Search bar that accepts keywords and searches through path files in the database for the keywords	4	3	
Search List 3.2.5.3b	Lists API documentation based on keyword search	4	3	
Basic View 3.2.5.1c	Search for keywords within an API function's name or type of data it returns.	8	3	
Advanced View 3.2.5.1d	Search for multiple and different criteria on same search	8	4	

Related API's View 3.2.5.1e	Able to search for API functions that work closely with each other or return similar data.	12	3	Group API calls into buckets defined by Timestamp

4. Assumptions and Risks

4.1 Assumptions

4.1.1 Server for subversion

Novacoast will provide a server for us to use for version control (Git), and document protection.

4.1.2 UI Designer

Novacoast will provide a user interface designer who will complete all the UI pages on time and not at any point leave us waiting for them to finish.

4.1.3 Training

Novacoast will provide guidance for learning all necessary software and technology for us to complete the project to their specifications (i.e. Ruby on Rails, EC2, javascript/JQuery, CouchDB, CSS, HTML, and AWS S3).

4.1.4 GoMockingbird

Novacoast will provide gomockingbird.com accounts to create mock-ups of the product.

4.1.5 Access to Novacoast R&D

Novacoast will provide access to staff for help with adapting system architecture from old designs.

4.1.6 Access to Office and Staff

Novacoast will allow at least weekly access to conference room and staff to assist with questions about direction and implementation of project.

4.2 Risks

4.2.1 CouchDB Overburdening

CouchDB may not be able to handle size of databases.

4.2.2 Insufficient Skill Set Among Team Members

Team members do not have, or too slowly learn, the necessary skills needed.

4.2.3 Scheduling Conflict

A team member(s) does not have the time to complete an assigned task due to either more pressing school/work/family engagements.

4.2.4 Complexity Underestimated

The complexity is above what we are capable of doing in the allotted amount of time.

4.2.5 Scalability

The code written does not scale up well, causing servers to crash regardless of the number of servers.

5. Support, Platforms, and Localization

5.1 Supported platforms

5.1.1 Supported browsers

We plan to support Chrome, Firefox 3, 4 and above, Safari 4 and 5.

5.1.2 Supported Mobile Platforms (future)

The system will support the following handhelds:

- Apple iPhone/iPad
- Android

A separate UI per device will need to be built.

5.2 Localization

Localization is the modification of software, web content, and documentation to meet the language and cultural differences among the targeted market. The app will be localized to the major languages determined by the stakeholders. The client's default language will be automatically determined by the users' browser. Users will have the ability to freely change between versions within the interface.

5.3 Documentation

To be determined

6. Technology Stack

6.1 Overview of Technologies Used

The following app will be built on a stack of these technologies, infrastructure, and third party tools.

a. HTML5
b. Ruby
c. javascript
d. Jquery
e. CSS
f. Ruby on Rails
g. CouchDB
h. VMWare
i. Twitter Bootstrap

7. Planning

7.1 Client Component - Novacoast

7.1.1 Novacoast Resources

The following Novacoast resources will be needed to complete the project as outlined in the next section:

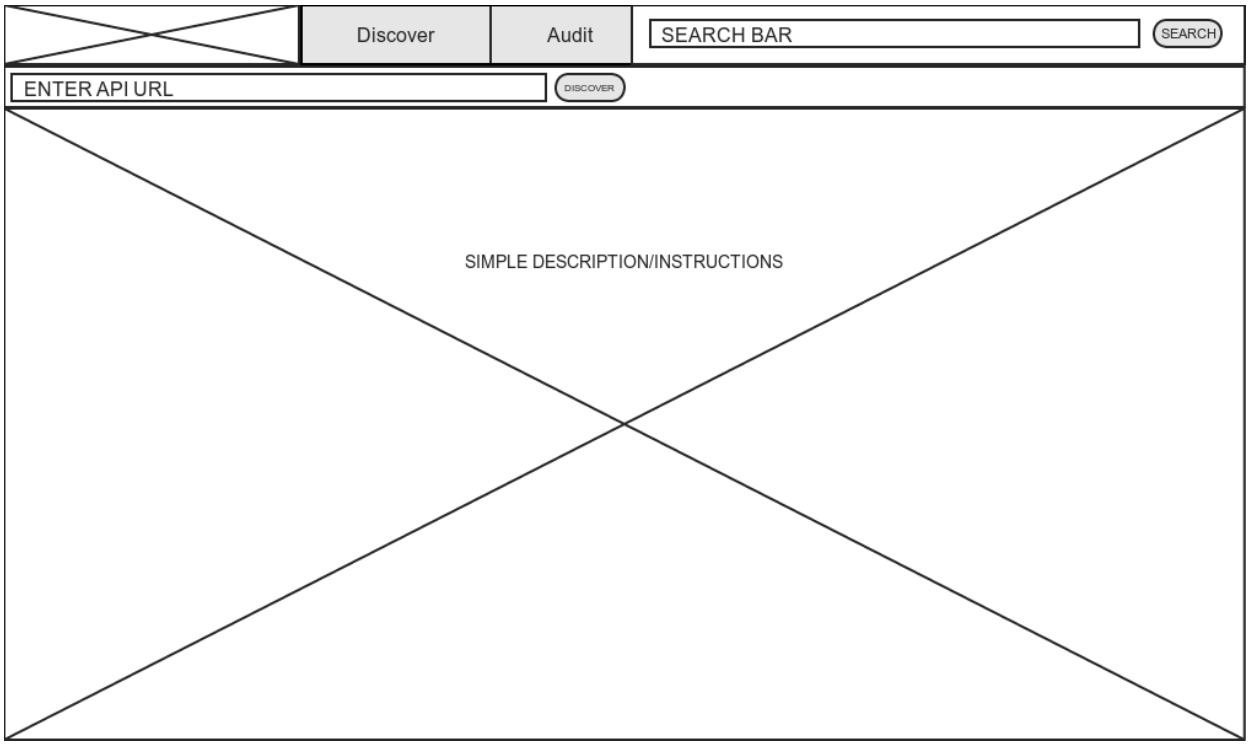
Role	**Members**
Project Manager	Eron Howard
Project Team Lead	Andrew Duong
QA Resources	David Parker, Renato Untalan
Developer Lead	Tong Wu
Developers	Andrew Duong, Jimmy Le, Mark Oparowski, Tong Wu
Scribe	Jimmy Le

7.1.2 Development Timeline

Component	Start Date	End Date	Risks and Assumptions	Owner
Set Up Team Environment and Learn the Tools	Feb.11.2013	Feb.15.2013	Risk of incompetent programmers.	All
Set Up Couchdb	Feb.18.2013	Feb.20.2013	Assume we will be using CouchDB	Tong
Code Planning	Feb.21.2013	Feb.22.2013	Assume the tools we are using will be unchanged	All
Priority 1 Tasks	Feb.25.2013	Mar.22.2013	Risk of P1 tasks not getting completed in time	All
Priority 2-3 Tasks	Mar.25.2013	Mar.26.2013	Risk of being road-blocked by P1 tasks	All
Testing and debug	Mar.25.2013	END	Assume we are completing P1 tasks on time	All
Complete all Tasks	Apr.29.2013	May.10.2013	Risk of excessive bugs in tasks	All
Implement Extra Components	May.13.2013	May.31.2013	Assume we will have extra components	All
Finalize project	Jun.3.2013	END	Risk of not finishing in time	Andy

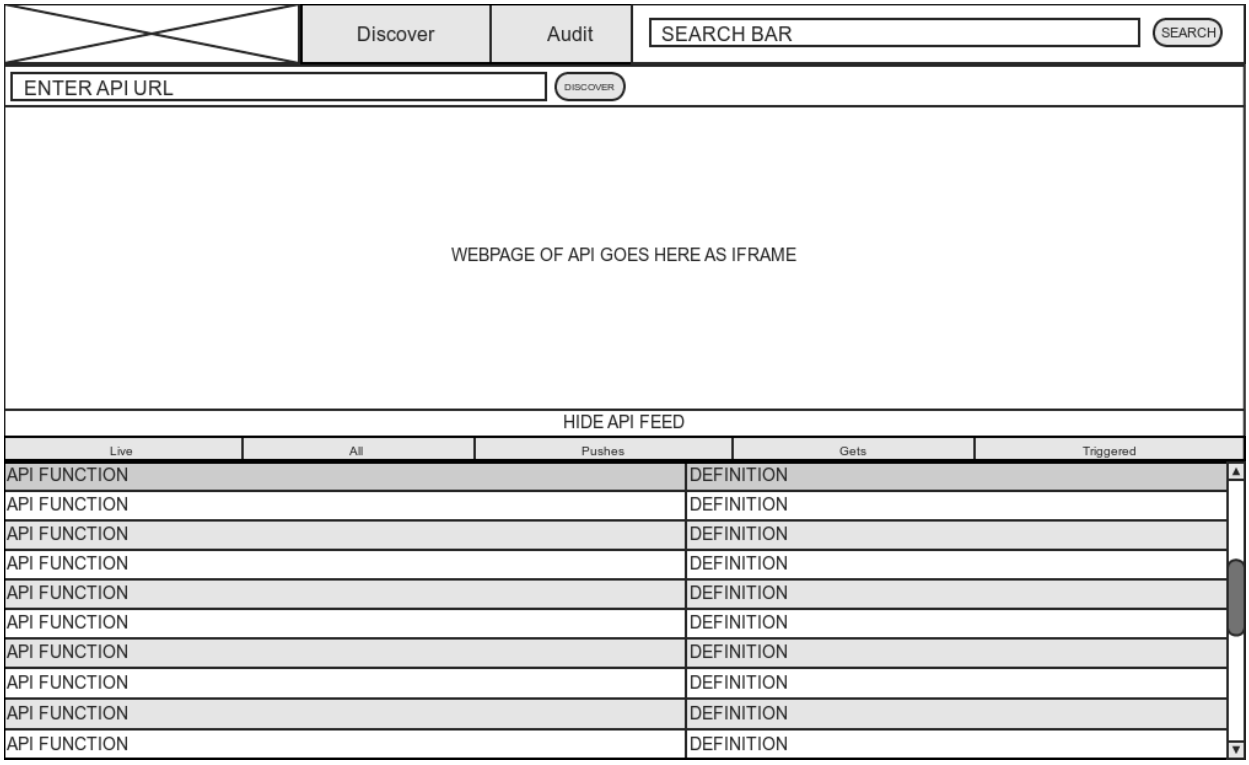
8. GUI Mock-ups

8.1 Landing Page



8.2 Discovery: Hidden API Feed

8.3 Discovery: Visible API Feed



8.4 API List from Search

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME API NAME

8.5 API Definition of Searched API

API FUNCTIONS FOR EXAMPLE API

Live | All | Pushes | Gets | Triggered

API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION
API FUNCTION	DEFINITION

I. Change Log

Date	Version	Author	Changes
2/12/2013	v0.1	NIBBLE	SRS Draft 1
3/7/2013	v0.2	NIBBLE	PRD revisions