RECEIPT KEEPER – IMPLEMENTATION OF RESTFUL API USING A LOOPBACK FRAMEWORK

C. Choi (cch.choi@gmail.com), H. Nahhal (nahhalhasan@gmail.com),

N.Collins (ncollins.pro@gmail.com), S. Kim (joe215.kim@gmail.com)

Supervisor: D. Dvorski

School of Engineering & Information Technology

Conestoga College

0. ABSTRACT

Keeping track of spending in daily life is the one of the tedious but important task. Many people are keeping financial ledger with hand writing or with the help of spread sheet program. However, it is very hard to keep track of spending and recognize overall trends with categories within a specific period. Moreover, if the receipt of item needs to be kept for the future reference such as a TAX report or refunding, it is hard to find the receipt where it was used.

Receipt Keeper is a solution to keep track of spending with mobile app and preserve the information by uploading the information to the server. Thus, users can keep track of their spending and check the statistics of their spending or search the receipt when needed. Moreover, for business users can collect or monitor their member’s receipt and spending by using of Group features.

This report will explore the solution developed with covering the major aspect of development, such as methodologies, architectural decision made, and technologies used. Finally, it shows how app is created for scalable RESTful app on Loopback Framework.

I. INTRODUCTION

Receipt Keeper will allow users to track how much money they spent in a certain category over a specific number of months, as well as start a group which others can be invited to join, and this will allow group admins to see reports for the group. Receipt Keeper is built to work on Android devices and also includes a web application. The mobile application will make it easier for people to manage categories, tags and reports while having the convenience of scanning receipts wherever they are. The web application provides the list of receipts and shows graphs to visualize users’ spending.

*A. Challenges and objectives*

The biggest challenge was to make a working product within limited time and limited resources. Thus, we try to start with implementing minimum viable product which has core

features; REST(*Representational State Transfer)* API supporting server, OCR (Optical Character Recognition).

*1) OCR engine to recognize numbers and characters on user’s receipt:* due to limitation of resource, open source products were considered, which were OpenCV, Tesseract, etc.

2) *Providing REST (Representational State Transfer) API for scalability: d*ifferent kind of devices should be connectable to the server such as web application, Android, iPhone, Tablet, even customized devices. Implementing a server which is connectable with API is another challenge.

3) *Android Application using REST API to communicate with Server:* To implement a working product within the limited time, the team is focusing on implementing core features, such as taking picture of receipt, recognizing characters on receipt, CRUD (Create, Update, Delete) of receipt, and uploading receipt to the server.

II. METHODOLOGIES

*A. Development Methodologies & Team Rules*

1. Agile Unified Process:

To implement working product with minimization of documentation, Agile Unified Process was used which has 2-week iteration period.

1. Coding rule

Coding rule was minimized for every member to follow the rule easily and was updated after members are accustomed to the rule. Coding rule also includes comment rule for Git, JavaScript, and Android.

*B. Source Control & Change Management*

1. GitHub

Three GitHub projects were created, one each for managing document, web server & application, android application.

1. Committing Rule

Direct commit to the server was discouraged, but making pull request was encouraged for peer review, and made a process for source code management.

1. Sharing Information

To share the information such as coding rule, link information, and minutes, Wiki page at GitHub was used.

*C. Issue Tracking*

GitHub’s issue tracking function was used to minimize managing or learning new tools. Although it has not many functionalities compared other issue tracking tools such as Jira, it has the benefit of seamlessly connectable to the GitHub.

*D. Backend as a Service (BAAS)*

LoopBack framework was used for implementing the server and will be stated in detail later.

*E. Deployment*

For supporting interoperation during the development, it was important to set up the server environment early so that every member can use it for development. HEROKU provides a free account for prototyping purpose. However, one of drawbacks of a free account is the server will go into sleep mode after 30 minutes of inactivity, so it needs to be activated before using the server. However, it is not a big problem for prototyping. Moreover, HEROKU supports buildpacks for deploying LoopBack app, so after uploading source code with Git to the HEROKU environment, it will be compiled and deployed automatically.

To avoid the problem of breaking the operation of the server, the pipeline features at HEROKU was used. FIG1. After setting up, every pull request at project’s GitHub will trigger a build at HEROKU, and it will pull from GitHub, and build automatically. This result can be verified at Dashboard of HEROKU. Normally, the code will be submitted to the staging server for testing purpose, and if the submitted source code was stable and decided to publish, it was promoted to the production stage manually. This

production server was a common server shared between members.

STAGING

PRODUCTION

Review

Fig. 1. HEROKU Pipeline

*F. Data as a Service (DAAS)*

Instead of setting up an internal customized environment using a container such as Docker, to reduce time setting up the environment, it was decided to use a cloud database service. Among the providers, mLab provides Database-as-a Service with various types of plans and features for MongoDB. Among the plans, the Sandbox plan provides 500MB of space free which is enough for development and prototyping, although it is slow compared to non-free plans.

*E. Sharing Issues and Communication*

Because team members worked in separate places, although one day per week was allocated for Capstone project lab, it was necessary to set up communication channels other than phone. The Slack is a famous messaging app for professional use. Channels were set up for the subject of discussion and have actively been used for the discussion of interoperation or sharing knowledge.

Additionally, to share the information such as meeting minutes, coding rules and documentation, GitHub’s Wiki was used.

III. ARCHITECTURAL DECISIONS

*A. Architectural Drivers*

1. Functional Requirements
2. User can input receipt easily by minimizing typing of information
3. Allow web application and mobile devices to connect to the server
4. User can search receipt or check the trend of spending with visualization
5. Quality Attributes
6. Scalability: Extensible to other kind of client
7. Modifiability: Easy to change API
8. Performance: Quick to search the information
9. Security
10. Constraints
11. The system should be delivered by Aug 10, 2016
12. Current available OCR engines are mostly for mobile
13. Architectural Decisions
14. REST API
15. NoSQL for the speed of search and Node.js
16. Express middleware modules for security such as Helmet
17. OCR processing is conducted at mobile

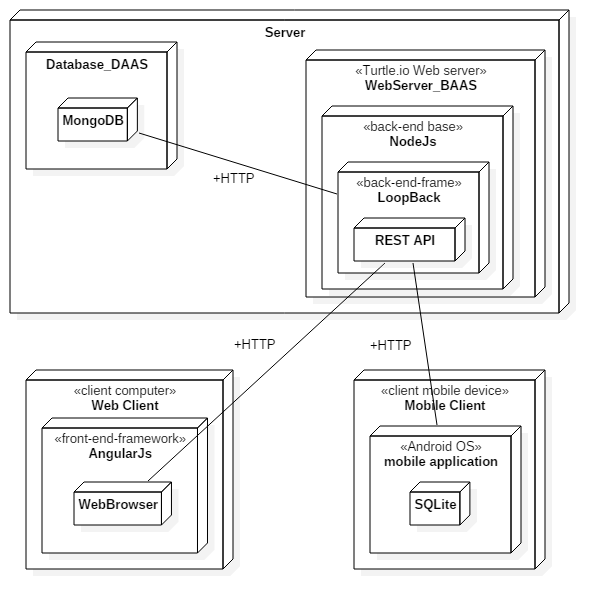


Fig. 2. Deployment Diagram

IV

. TECHNOLOGIES

Because one of the application’s main features is to help a user entering receipts by using Optical Character Recognition (OCR) technologies, OCR is main technology that is used.

1. Optical Character Recognition (OCR)

For implementing OCR within limited time, team decided to use open source for character recognition. OpenCV and Tesseract are the products considered. OpenCV is a famous computer vision open source product, and Tesseract is a famous OCR engine which was started in 2005 by HP and Google is a main contributor to the project. There are a lot of open source project subsidiaries based on Tesseract, and android-ocr is the project that was used for our project [1]. Although OpenCV was not used for this project, it can be used for pre-processing receipt images such as contrast enhancement, auto tilting of image to enhance recognition rate for OCR engine in the future project.

1. LoopBack

Another main technology is a server to support REST API. Various platforms are considered, but due to lack of team’s skills and to minimize time to implement the server, LoopBack is chosen finally. Table1 is the comparison of features between BAAS (Back end As a Service) solutions.

LoopBack is an open source Node application framework based on Express which is supported by IBM and StrongLoop (Acquired by IBM on September, 2015). It makes it easy to create REST API platforms and managing REST API with provided tools, such as creating new API or apply access control list (ACL). Moreover, it has Built-in API explorer to test API which is useful to test the server operation before implementing web client or mobile application.

LoopBack supports a Yeoman-based scaffolding tool for generating new project skeletons, so it takes a few minutes to make skeletons for the project.

LoopBack helps with that further by providing a model generator for building your models quickly. Entity relationships are also very well supported. The security model is also complex with user roles, principals and ACLs

Authentication and authorization are important tasks for implementing a server, but it was easy to setup authentication and authorization by using LoopBack’s built-in role based access controls. Open Standard for Authorization (OAuth) user and registration models is baked in the Loopback, so it is also easy to embed third party logins such as Google and Facebook.

StrongLoop Arc is a graphical tool for building, deploying, managing and monitoring LoopBack applications and APIs.

Developers have a choice of various database connectors for all major SQL databases and MongoDB. LoopBack helps with that further by providing a handy model generator for building your classes quickly. These can be validated with the built-in

validation methods. Entity relationships are very well supported. The security model is complex with user roles, principals and ACLs.

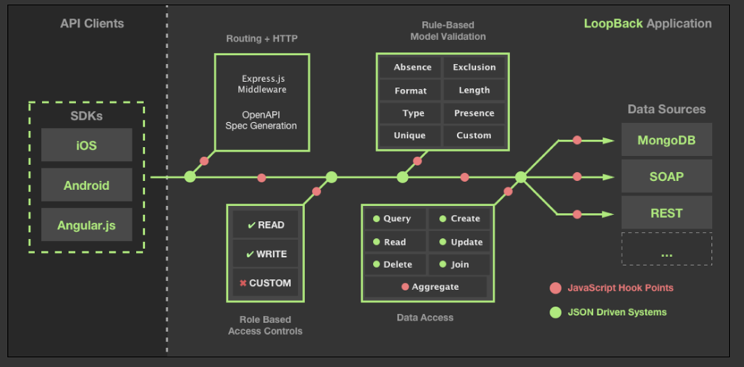


Fig. 2. LoopBack Application

TABLE I. COMPARISON OF BAAS (BACKEND AS A SERVICE)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **LoopBack** | **Express** | **Sails** | **Restify** | **Meteor** |
| **Type** | API framework | HTTP server library | Web MVC framework | REST HTTP library | Full-stack JavaScript app platform |
| **Top Features** | Enterprise connectivity, API Explorer, generators, client SDKs, websocket microservices | HTTP routing, middleware | Rails familiarity, MVC | Simplicity, REST routing | Universal JavaScript, reactive rendering, websocket microservices |
| **Suitable For** | Web apps, APIs | Simple web apps | Web apps, APIs | Simple REST APIs | Web apps |
| **Github Stars** | 5k | 19k | 10k | 3k | 28k |
| **Support** | StrongLoop | StrongLoop | N/A | N/A | Meteor Development Group |
| **Pure Node runtime** | Yes | Yes | Yes | Yes | No |
| **Client SDKs** | Angular, Browser, Node.js, iOS, Android, Xamarin | N/A | None | None | JavaScript, Cordova for iOS and Android, React, AngularJS |
| **Export API Definition** | Yes | With strong-remoting | None | None | With meteor-rest |
| **Tools** | Visual API composer, Explorer, CLI code generators | CLI app generator | Yeoman generator | Yeoman generator | CLI tool |
| **Visual API composition** | Yes | No | Yes | Yes | Yes |
| **StrongLoop Arc Build & Deploy, Monitoring, Profiling** | Yes | Yes |  |  | Proprietary package system and repository, npm |
| **Extensions** | Push, File Storage, Passport, OAuth 2.0, Express Middleware | Express / Connect Middleware | In-memory, File, PostgreSQL, MySQL, MongoDB | None | MongoDB, MySQL and PostgreSQL via 3rd-party packages |
| **Data sources** | In-memory/file, MongoDB, MySQL, Oracle, PostgreSQL, SQL Server, ATG, Email, REST, SOAP | None | No | No | Basic allow/deny |
| **ACLs** | Yes | No |  |  |  |

4. OPEN SOURCES

A lot of open sources are help to implement features within short time, but a lot of points should be considered beforehand such as License, sustainability, and securities.

1. Server

|  |  |  |
| --- | --- | --- |
| **Open Source** | **License** | **Function** |
| **bunyan: ^1.8.1,** | **Apache 2.0** |  |
| **compression: ^1.0.3,** |  |  |
| **cors: ^2.5.2,** |  |  |
| **errorhandler: ^1.1.1,** |  |  |
| **express-bunyan-logger: ^1.3.0,** |  |  |
| **helmet: ^1.3.0,** |  |  |
| **loopback: ^2.22.0,** |  |  |
| **loopback-boot: ^2.6.5,** |  |  |
| **loopback-component-explorer: ^2.4.0,** |  |  |
| **loopback-component-storage: ^1.5.0,** |  |  |
| **loopback-connector-mongodb: ^1.15.2,** |  |  |
| **loopback-datasource-juggler: ^2.39.0,** |  |  |
| **loopback-ds-timestamp-mixin: ^3.2.4,** |  |  |
| **morgan: ^1.7.0,** |  |  |
| **serve-favicon: ^2.0.1** |  |  |

1. Web Client

|  |  |  |
| --- | --- | --- |
| **json3: ~3.3.1,** |  |  |
| **font-awesome: 4.3.0,** |  |  |
| **oclazyload: ~0.5.2,** |  |  |
| **angular-loading-bar: ~0.7.0,** |  |  |
| **angular-chart.js: ~0.5.2,** |  |  |
| **angular-ui-router: ^0.3.0,** |  |  |
| **angular-resource: ^1.5.6,** |  |  |
| **bootstrap: ^3.3.6,** |  |  |
| **eonasdan-bootstrap-datetimepicker: ^4.17.37** |  |  |

1. Android Mobile App

|  |  |  |
| --- | --- | --- |
| **com.rmtheis:tess-two:5.4.0** |  |  |
| **com.strongloop:loopback-sdk-android:1.5.+** |  |  |
| **com.github.bumptech.**  **glide:glide:3.6.0** |  |  |
| **com.rmtheis:tess-two:5.4.0** |  |  |
| **Tesseract: https://github.com/tesseract-ocr/tesseract** |  |  |

V. APPLICATON PROGRAM INTERFACES (API)

Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive.”

*A. Application Layer*

1. *Routes*
2. *Middleware*
3. *Errors*
4. *Formatters*

*B. Domain Layer*

*C. Service Layer*

1. *Data Access*
2. *JWT (JSON Web Token)*
3. *Authentication*
4. *ACL(Access Control List)*

*D. Foundation Layer*

1. *Database*

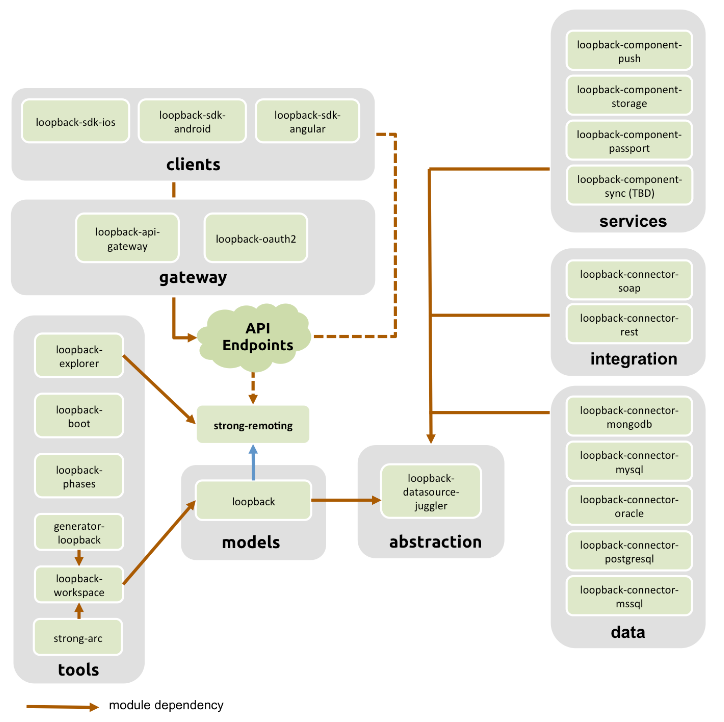


Fig. 3. LoopBack Modules

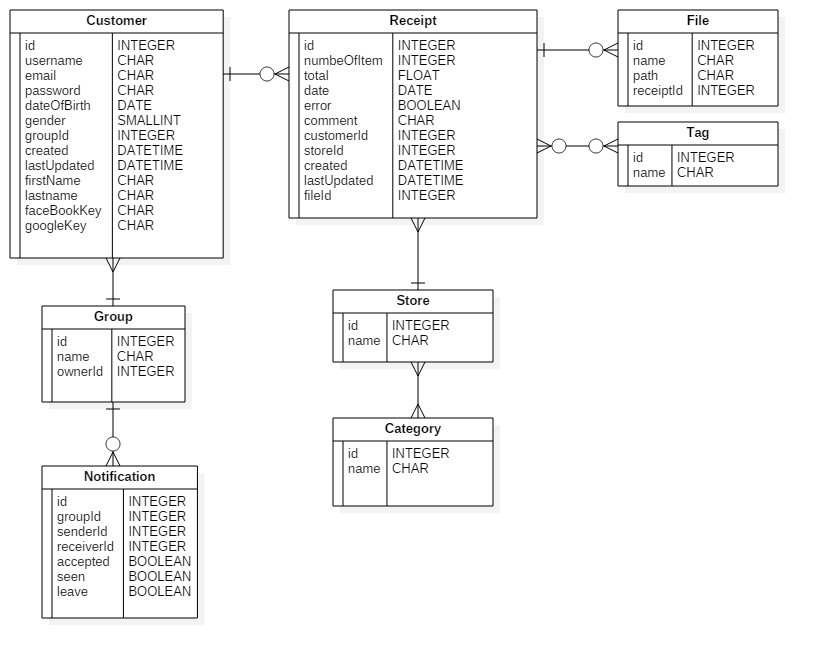


Fig. 4. Entity Diagram

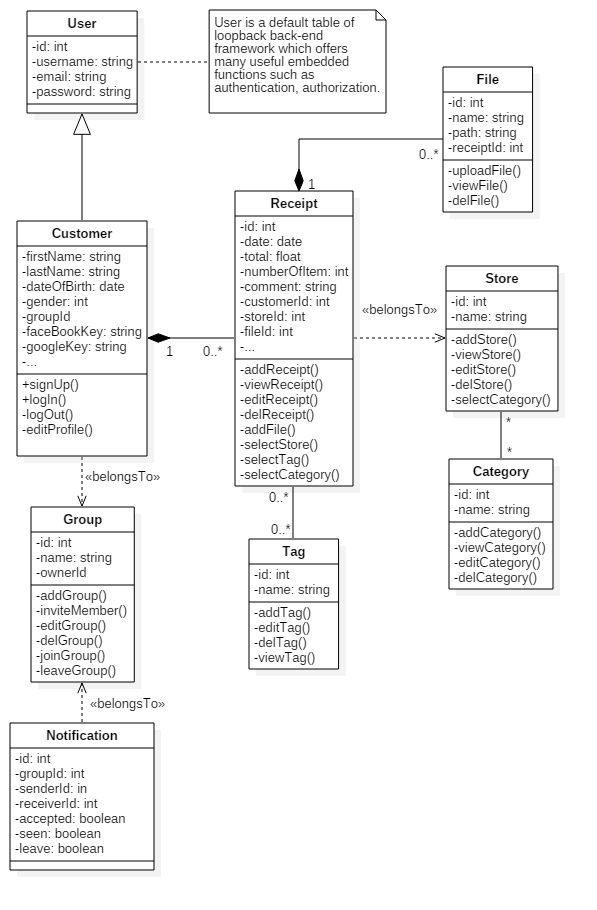


Fig. 5. Class Diagram

VI. CONCLUSIONS

By using the Loopback framework, it is easy and quick to implement REST API for scalability. It is also fast because is based on NodeJS and easy to implement a single page application with the help of AngularJS

REFERENCES

[1] android-ocr : https://github.com/rmtheis/android-ocr

[2] HEROKU, [*https://www.heroku.com/pricing*](https://www.heroku.com/pricing)

[3] LoopBack,

[*https://docs.strongloop.com/display/public/LB/LoopBack*](https://docs.strongloop.com/display/public/LB/LoopBack)

[4] OCR, *https://en.wikipedia.org/wiki/Optical\_character\_recognition*

[5] OpenCV: *https://en.wikipedia.org/wiki/OpenCV*

[6] Tesseract OCR: [*https://github.com/tesseract-ocr/tesseract*](https://github.com/tesseract-ocr/tesseract)

[7] Express, Production Best Practices: Security: *https://expressjs.com/en/advanced/best-practice-security.html*

[8] StrongLoop, Security considerations: *https://docs.strongloop.com/display/public/LB/Security+considerations*

[9] WIKIPEDIA: [*https://en.wikipedia.org/wiki/*](https://en.wikipedia.org/wiki/)

Glossary

***A***

ACL:

Access control list, a list associated with an object that identifies all the subjects that can access the object and their access rights. See Authentication, authorization, and permissions.

*B*

Build packs are scripts that are run when the app is deployed. They are used to install dependencies for the app and configure the environment.

*D*

Data source:

A data source connects with specific database or other back-end system using a connector.

*H*

Helmet

Helmet is node module and it helps secure a Express apps by setting various HTTP headers.

*O*

OCR (Optical character recognition):

Optical character recognition (optical character reader, OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast)

OpenCV:

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision, originally developed by Intel's research center in Nizhny Novgorod (Russia), later supported by Willow Garage and now maintained by Itseez.[1] The library is cross-platform and free for use under the open-source BSD license.

*R*

REST API:

representational state transfer (REST) is an architectural style of application programming interfaces consisting of a coordinated set of components, connectors, and data elements within a distributed hypermedia system, where the focus is on component roles and a specific set of interactions between data elements rather than implementation details. Its purpose is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability. REST is the software architectural style of the World Wide Web. To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems

typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers. REST systems interface with external systems as web resources identified by Uniform Resource Identifiers (URIs), for example /people/tom, which can be operated upon using standard verbs such as GET /people/tom.