*Florida International University*

*School of Computing and Information Sciences*

Software Engineering Focus

Final Deliverable

Project Title: GoDutch 1.0

**Team Members**: Zhenglin Pan, Jonathan Beltran

**Product Owner(s)**: Dr. Deng Pan

**Mentor(s)**:

**Instructor**: Masoud Sadjadi, Mohsen Taheri

The MIT License (MIT)

Copyright (c) *2016 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

***Abstract***

*The most annoying part of any trip with friends is figuring out how much each person needs to pay for the trip. GoDutch is a web application which seeks to remove the difficulty of payment by making sure each member of the group contributes equally to whatever they consumed. This is accomplished through allowing real-time collaboration amongst the trip members for reporting the expenses and GoDutch will calculate the rest.*

**Table of Contents**

**Introduction** ………………………………………………………………………………………………………………………… 6

Current System ……….………………………………………………………………………………………………………… 6

Purpose of New System …………………………………………………..………………………………………………… 6

**User Stories**

Implemented User Stories ………………………………………….……………………………………………………. 7

**Project Plan**

Hardware and Software Resources ……………………………………………………...…………………….… 17

Sprints Plan ……………………………………………………………..………………………………………………………. 19

*Sprint 1*  …………………………………………………………………………………………………………………………... 19

*Sprint 2*  …………………………………………………………………………………………………………………………... 19

*Sprint 3*  …………………………………………………………………………………………………………………………... 19

*Sprint 4*  …………………………………………………………………………………………………………………………... 20

*Sprint 5*  …………………………………………………………………………………………………………………………... 20

*Sprint 6*  …………………………………………………………………………………………………………………………... 21

*Sprint 7*  …………………………………………………………………………………………………………………………... 21

**System Design**

Architectural Patterns ………………………………………………………………………………………………….. 22

System and Subsystem Decomposition …………………………………………………………………………….… 23

Deployment Diagram …………………………………………………………………………………………………….…... 25

Design Patterns ………………………………………………………………………………………………………….….... 25

**System Validation**  ……………………………………………………………………………………………………………….27

**Glossary**  ……………………………………………………………………………………………………………………………….32

**Appendix**  ……………………………………………………………………………………………………………………………….33

Appendix A - UML Diagrams ………………………………………………………………………………………………. 33

*Static UML Diagrams*

*Dynamic UML Diagrams*

Appendix B - User Interface Design …………………………………………………………………………….…... 63

Appendix C - Sprint Review Reports …………………………………………………………………………...…… 65

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents …………………………………………………………………….…………… 69

# 

# 

# Introduction

GoDutch 1.0 is a web application that assists a group of travellers with the distribution of payments that happen during their trip. It firstly computes the share a user needs to pay individually for their trip based on their own purchase and consumption activities. Then it generates a list of transactions among the participants, so that each user can end up paying their part by following the list, while the total number of transactions within the group is minimized. GoDutch can be found very helpful for different kinds of trip groups, especially those where multiple purchasers each pays for a subgroup of participants, i.e. payments cannot be simply evenly divided.

# Current System

When looking at other applications which attempt to accomplish the goal of group expenditure calculation, you will mostly find single user applications. Not only is this method inefficient, it can lead to disputes over the fairness of the contribution. Moreover these applications treat everyone in the group as “consumers” when there are situations which one group member may not participate in all the expenses.

Additionally, because payment is involved for these types of applications, security must be a main concern. In GoDutch any sensitive material is automatically encrypted before it is saved. The lines of communication between each user is also protected and encrypted by the use of JWT (JsonWebTokens). This prevents sessions from being hijacked, which is a common attack when session information is used to maintain the state of an application.

# Purpose of New System

As a group tool, GoDutch allows multiple users from the same trip editing at the same time and seeing each other’s changes mostly in real time. It not only can serve as an payment calculator, but also can be used as a purchase plan of items that will be needed for the trip. All participants have the access to add items and modify their purchase and consumption.

GoDutch 1.0 leverages Model-View-Controller (MVC) architectural pattern, where views are constructed using HTML and Less.js, controllers are implemented in Angular.js and JavaScript, and models are stored in MongoDB. The project is hosted on Node.js and routing is handled by Socket.io. The transaction result is generated using Minimum-Cost-Max-Flow algorithm (MCMF).

# User Stories

The following section provides the detailed user stories that were implemented in this iteration of the GoDutch project. These user stories served as the basis for the implementation of the project’s features. This section also shows the user stories that are to be considered for future development. The numbering system used for the user story is what was assigned by mingle, it holds no significance.

## Implemented User Stories

## Sprint 1, 2

(01/24/2017 - 02/7/2017)

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/117)100 Set up host with Gulp.js and Node.js

### Description:

As a developer, I would like to set up the host using Gulp.js and Node.js to ease development process.

### Acceptance Criteria:

1. Gulp should watch all the changes that happen during development and update public code so that developers will only need to refresh the page to see changes.
2. The server should be able to host the page on localhost using Node.js.

### UML diagrams:

* See diagrams in Appendix A; Figures A1, A2

# 

# 

# 

# [#117](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/117) Implement MCMF in JS

### Description:

As a user, I would like the application to take care of transaction optimization using an efficient algorithm.

### Acceptance Criteria:

1. Given input (the total amount spent by the group) and usage per person, the algorithm should return minimum number of transactions so that everyone is paying the correct amount.
2. The computation should be done within 0.1 second for any input that has less than 100000 users.

### UML diagrams:

* See diagrams in Appendix A; Figures A3, A4

# 

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/124)124 Implementing Login

### Description:

As a user, I want to be able to login into the GoDutch Application

### Acceptance Criteria:

1. Given that a user has an account created that user should be able to successfully login
2. When a user logins successfully, that user should be redirected to the homepage

### Related Task:

1. User story #125 Implementing Signup
2. User story #126 Secure Login Information

### UML diagrams:

* See diagrams in Appendix A; Figures A5 - A7

# 

# 

# 

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/125)125 Implementing SignUp

### Description:

As a user, I want to be able to signup and use the GoDutch Application

### Acceptance Criteria:

1. Given a user’s desired Username and Password, a user should be able to create an account and have it saved in the database
2. A user should be able to login on a created account
3. A username/email is required to be unique and any attempt to create an account with an existing username/email should be rejected

### Related Task:

1. User story #124 Implementing Login
2. User story #126 Secure Login Information

### UML diagrams:

* See diagrams in Appendix A; Figures A8 - A10

# 

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/126)126 Secure Login Information

### Description:

As a user, I want my login information to be encrypted

### Acceptance Criteria:

1. Given that a user is attempting to create an account, the password of the user should be encrypted before saving
2. Given a query from the database for a user, the information retrieved should show encryption where appropriate and not clear text

### Related Task:

1. User story #124 Implementing Login
2. User story #125 Implementing Signup

## 

## 

## Sprint 3

(02/08/2017 - 02/21/2017)

# [#127](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/127) Implement Navigation UI

### Description:

As a user, I would like to have a well designed navigation bar where I can easily go to the homepage, my trips, sign in/sign out and contact information, etc

### Acceptance Criteria:

1. Navigation bar should be on top, from left to right and be consistent with the page in both color and style.
2. Navigation bar should at least include Homepage, MyTrip, ContactUs, SignIn and SignOut tabs or buttons.
3. Switching among navigation tabs should not take more than 0.1 second loading time.

### Related Task:

1. User story #128 Login/Logout/Signup UI design
2. User story #129 Implement MyTrip Page UI

### UML diagrams:

* See diagrams in Appendix A; Figures A11-A13

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/128)128 Login/Logout/Signup UI design

### Description:

As a user, I want to quickly and easily create, access, and exit my account

### Acceptance Criteria:

1. A user may login
2. A user may signup
3. A confirmation is performed whenever login/logout/signup
4. An appropriate message is displayed

### Related Task:

1. User story #124 Implementing Login
2. User story #125 Implementing Signup
3. User story #126 Secure Login Information

### UML diagrams:

* See diagrams in Appendix A; Figures A14-A16

# 

# [#12](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/129)9 Implement MyTrip Page UI

### Description:

As a user, I would like to be able to browse my past trip list and view the details of them. The details should include the purchasers, items and consumers of the selected trip.

### Acceptance Criteria:

1. MyTrip page must have sections such as past trips, purchasers, items and consumers.
2. MyTrip page must have consistent color and style with the rest content.

### Related Task:

1. User story #127 Navigation UI

### UML diagrams:

* See diagrams in Appendix A; Figures A17-A19

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/130)130 Adding Real-Time requirements

### Description:

As a user, I want to see other people’s additions/edits in real time

### Acceptance Criteria:

1. When two users are on the system in the same time, they may communicate with each other
2. When two users share the same trip, they can see the changes in real time
3. When two users do not share the same trip, they are not affected by each other

### Related Task:

1. User story #129 Implement MyTrip Page UI

### UML diagrams:

* See diagrams in Appendix A; Figures A20-A22

## 

## 

## Sprint 4

(02/22/2017-3/07/2017)

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/145)145 Implement Routes using Socket.IO

### Description:

As a developer, I want to handle all my events through sockets instead of RESTful API routes

### Acceptance Criteria:

1. All site functions which worked previously must also function after change
2. Sessions are now kept tracked by the sockets

### Related Task:

1. User story #130 Adding Real-Time requirements

### UML diagrams:

* See diagrams in Appendix A; Figures A24, A25

# 

# [#1](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/146)46 Implement MyTrip Page Frontend Logic

### Description:

As a user, I would like to use MyTrip page to create, view and edit my trips. For each trip, I want to be able to see all the participants and items purchased for that trip as well as the relationship among items and participants.

### Acceptance Criteria:

1. MyTrip page must at least have create, view and edit trip functionalities.
2. MyTrip page must highlight the item/participant selected by user and highlight related items/participants as well.
3. All operations must be done within 0.1 second.

### Related Task:

1. User story #129 MyTrip Tab UI Implementation

### UML diagrams:

* See diagrams in Appendix A; Figures A26-A28

# 

# 

# [#1](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/147)47 Implement Routes for Transaction Calculation Functionality

### Description:

As a user, I would like to see the minimized transaction list by clicking on a “calculation” button.

### Acceptance Criteria:

1. There should be a button for user to click on to run the MCMF algorithm.
2. There should be route connecting the button to the back end algorithm.
3. The calculation result should be shown on the page.
4. The process must be done in 0.1 second.

### Related Task:

1. User story #145 Refactoring RESTful API to sockets
2. User story #117 Implement MCMF in javascript

### UML diagrams:

* See diagrams in Appendix A; Figures A29-A31

## Sprint 5

(03/07/2017-3/24/2017)

# 

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/157)157 Implement Emailing Service

### Description:

As a user, I would like to invite my friends and family to my trip via email

### Acceptance Criteria:

1. SMTP service successfully sends email to any account
2. When a user clicks on that link, they are added to the trip

### Related Task:

1. User story #146 Implement MyTrip Page Frontend Logic

### UML diagrams:

* See diagrams in Appendix A; Figures A32-A34

# 

# 

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/158)158 Content update for Home/About/Contact

### Description:

As a user, I want to hit the home page when I login, be able to contact the admins of the site, and know what the purpose of the site is

### Acceptance Criteria:

1. Navigation to other pages are possible and the content can be displayed if available

### Related Task:

1. User story #127 Navigation UI implementation

### UML diagrams:

* See diagrams in Appendix A; Figures A35

# 

# [#1](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/159)59 Refactor object storage to enable saving and retrieving data

### Description:

As a user, I would like to have access to my past trips that are stored in database and to save my trips to database whenever a change is made.

### Acceptance Criteria:

1. Data must be retrieved and saved concurrently safely.
2. Changes should be saved in real time.
3. Users can see their past trips saved in database through MyTrip trip list.
4. The saving and retrieving process must be done in 0.1 second.

### Related Task:

1. User story #146 Implement MyTrip Page Frontend Logic
2. User story #130 Real-Time requirement

### UML diagrams:

* See diagrams in Appendix A; Figures A36-A38

## Sprint 6

(03/24/2017-4/04/2017)

# 

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/166)166 Reduce the Saving Frequency

### Description:

As a user, I want the system to save my changes after I stop typing or making changes.

### Acceptance Criteria:

1. The system should save user changes automatically after it detects the user has stopped typing.

### Related Task:

1. User story #146 Implement MyTrip Page Frontend Logic
2. User story #130 Adding Real-Time Requirements

### UML diagrams:

* See diagrams in Appendix A; Figures A39-A41

# [#](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/167)167 Broadcast Changes of a Trip to All Its Participants

### Description:

As a user, I would like to have other people see the updated version of the trip soon after I finish changing it and vise versa.

### Acceptance Criteria:

1. New changes should be saved to database and be broadcasted to all users within the same trip.
2. The broadcast should be concurrently safe.

### Related Task:

1. User story #146 Implement MyTrip Page Frontend Logic
2. User story #159 Refactor object storage to enable saving and retriving data
3. User story #130 Real-Time requirement

### UML diagrams:

* See diagrams in Appendix A; Figures A42-A44

# 

# 

# [#1](https://fiu-scis-seniorproject.mingle.thoughtworks.com/projects/godutch/cards/173)73 Refactoring Socket Auth and Functions

### Description:

As a developer, I want to handle my own form of authentication in order to have consistency in login throughout the site.

### Acceptance Criteria:

1. All site functions which work previously work after the change as well
2. Unauthorized access is prevented

### Related Task:

1. User story #124 Implementing Login
2. User story #130 Adding Real-Time Requirements
3. User story #145 Refactoring RESTful API routes to sockets

# Project Plan

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

## Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Project Management Tools

|  |  |
| --- | --- |
| Google Drive | Online file storage and edit tool that helps the team to share resources. |
| Mingle | Agile board that helps manage the project in terms of stories and tasks within sprints. |
| Github | Version control tool that eases team collaboration so that each team member can work separately. |

Hardware

|  |  |
| --- | --- |
| DELL Latitude E7470 | OS: Windows 7  CPU: Intel Core i7  RAM: 8GB |
| MSI Phantom Pro | OS: Ubuntu 14.0.4  CPU: Intel Core i7  Ram: 5GB |

Software

|  |  |
| --- | --- |
| VirtualBox | Virtual machine generator that creates Ubuntu environment for the team to test on multiple operation systems |
| Google Chrome | Developer friendly browser where GoDutch will be released on. |
| GitBash | Git tool for windows that allows windows users use git commands as wells as basic linux commands in console. |
| Robomongo | Visualization tool for MongoDB that provides developers with easy access to view and edit documents stored in MongoDB. |
| Visual Studio Code | Low weight IDE with clean git version control tool built in that helps developers code and resolve merge conflicts easy and efficiently. |

## 

## 

## Sprints Plan

### Sprint 1, 2

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 3:00pm, 01/31/2017

End time: 3:45pm, 01/31/2017

After discussion, the velocity of the team were estimated to be on pace.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 117
* User Story 100
* User Story 123

The team members indicated their willingness to work on the following user stories.

* Zhenglin Pan
* User Story 117
* User Story 100
* User Story 123

### Sprint 3

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 3:00pm, 02/14/2017

End time: 3:45pm, 02/14/2017

After discussion, the velocity of the team were estimated to be on pace.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 127
* User Story 128
* User Story 129
* User Story 130

The team members indicated their willingness to work on the following user stories.

* Jonathan Beltran
* User Story 128
* User Story 130
* Zhenglin Pan
* User Story 127
* User Story 129

### Sprint 4

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 3:00pm, 02/28/2017

End time: 3:45pm, 02/28/2017

After discussion, the velocity of the team were estimated to be on pace.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 146
* User Story 145
* User Story 147

The team members indicated their willingness to work on the following user stories.

* Jonathan Beltran
* User Story 145
* Zhenglin Pan
* User Story 146
* User Story 147

### Sprint 5

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 3:00pm, 03/21/2017

End time: 3:45pm, 03/21/2017

After discussion, the velocity of the team were estimated a little slow.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 157
* User Story 158
* User Story 159

The team members indicated their willingness to work on the following user stories.

* Jonathan Beltran
* User Story 157
* User Story 158
* Zhenglin Pan
* User Story 159

### Sprint 6

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 3:00pm, 03/28/2017

End time: 3:45pm, 03/28/2017

After discussion, the velocity of the team were estimated a little slow.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 166
* User Story 167
* User Story 173
* User Story 174

The team members indicated their willingness to work on the following user stories.

* Jonathan Beltran
* User Story 173
* User Story 174
* Zhenglin Pan
* User Story 166
* User Story 167

# 

# 

# System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

## Architectural Patterns

GoDutch is essentially leveraging Model-View-View Model (MVVM) architectural pattern (Figure 1). The users will be directly interacting with the view models i.e. trip model, item model and person model. During the interaction, once the views are changed, they will change the view models that are data-bound to them, then the latter will send signals to database and perform corresponding changes to the models. Similarly, when the view models are modified by commands, the views will be notified so that users can see the changes reflected on UI in real time.

On top of MVVM, GoDutch is also taking advantage of Client/Server (C/S) architectural pattern. The service that minimizes total number of transactions within a trip group is implemented on server. After users finish editing their trips, they can click on the calculate button, which will send a service request to the server, where MCMF algorithm will be run to generate the calculation result. Then GoDutch will send the result back to the client side and display it to the users to finish the service. By using C/S architectural pattern, we can keep GoDutch core functionality implementation private and reduce the computational power requirement of user devices.

System Design.png

Figure 1. GoDutch utilizes both MVVM and C/S architectural patterns.

## System and Subsystem Decomposition

GoDutch has four layers, each has its own subsystem (Figure 2).

* View layer is implemented in Html and CSS. It presents the page and contents that users request to view. The pages include Home, MyTrip, ContactUs, About, SignUp, etc. When users are interacting with the page, CSS is changed according to frontend controllers implemented in Angular.js.
* Front End layer is controlled by Angular.js. It binds variables to the page so that user changes can be captured and processed in real time. Similarly, when frontend logic modifies values that are bound to views, the changes will be pushed to the pages simultaneously.
* Routing layer is handled by both Socket.io and HTTP request. User login routes go through HTTP request while functionality related routes use sockets to communicate.
* In Back End layer, we use Node.js as the server host. JavaScript is used to deal with requests and perform corresponding interactions with MongoDB and Min-Cost-Max-Flow service.

Subsystem.png

Figure 2. GoDutch has View, Front End, Routing and Back End layers.

## 

## Deployment Diagram

We use Gulp.js to gather all JavaScript, Html and CSS code and put them into distributable directory, where App.js is generated. GoDutch server hosts the website using App.js on the port that is specified in config file. Meanwhile, GoDutch database server is hosted on another port. Once the page is hosted, users are able to access the application through GoDuth UI from their devices (Figure 3).

Deployment Diagram.png

Figure 3. GoDutch Deployment Diagram.

## Design Patterns

GoDutch uses several design patterns including Front Controller, Observer and Singleton.

The Front Controller design pattern is very popular in web applications because of the way it handles requests using one centralized interface. In GoDutch, we use the Route class to handle all requests from the client side and send back responses. Socket.io does most of the work in the implementation while HTTP requests are only used for user login routes. With Front Controller design pattern, the GoDutch routing system is well organized and easy to maintain (Figure 4).

In the frontend implementation, we utilize Observer design pattern to build the link between View and View Model. All variables used for presenting views to the users are observed by the view models through data binding. Consequently, changes made by users can be captured and processed by frontend logic in real time and the users can observe new modifications done by front end logic simultaneously as well.

Additionally, each user can only have one socket during their session. This is an implementation of Singleton design pattern. One socket per user not only makes the authentication more secure, but also reduces the cost needed to keep user sessions.

# Object Design.png

Figure 4. GoDutch leverages Front Controller, Observer and Singleton design patterns.

# 

# 

# System Validation

For this iteration of GoDutch development, we do not have automated testing enabled. However, we conducted numerous manual tests including unit test, integration test, functional test and performance test to ensure GoDutch is working properly during the fast paced development cycles. The naming system used corresponds to the test’s user story ID number.

## Unit Test

### Test case ID: #117 Testing MCMF in JS

* Description/Summary of Test: Tested example tripModal against the algorithm
* Pre-condition: A trip which whose cost has not been calculated
* Expected Results: A list of relationships showing users and how much they owe in payment
* Actual Result: Min-max flow returned a satisfied network whose links represented the amount owed among the different users.
* Status (Fail/Pass): Pass

### Test case ID: #124\_1 Testing Successful Login

* Description/Summary of Test: A user was manually created in the database, and we attempted to login with those credentials
* Pre-condition: The desired user already exist in the database
* Expected Results: The server will return a successful message and create a cookie for that user
* Actual Result: The server returned the successful message, and when the page was refreshed the system still recognized that the user was logged in
* Status (Fail/Pass): Pass

### Test case ID: #124\_2 Testing Unsuccessful Login

* Description/Summary of Test: Two test were performed here, one with random entry (with no entry in the database) and one with incorrect password (testing uniqueness)
* Pre-condition:The database does not contain the user who will login
* Expected Results: The user will be rejected by the server, and in the case of incorrect password, they will be rejected by an appropriate message
* Actual Result: The server rejected their login, in the case of random login they were rejected with unknown user; in the case of incorrect password they were rejected with incorrect password
* Status (Fail/Pass): Pass

### Test case ID: #125\_1 Testing Successful Signup

* Description/Summary of Test: We made sure to attempt to signup with a user who didn’t already exist in the database
* Pre-condition:The database does not contain the user who will signup
* Expected Results: The server will return a success message along with the database now containing that user
* Actual Result: The server has no issues in saving the user and the user was now contained in the database
* Status (Fail/Pass): Pass

### Test case ID: #125\_2 Testing Unsuccessful Signup

* Description/Summary of Test: We next attempt to signup with those same credentials from the previous test case, Test case #125\_1
* Pre-condition:The database contains at least one matching username or email
* Expected Results: The database will reject the account creation, and the database will return an error of non-unique signup
* Actual Result: When attempting to signup with a duplicate email, we were rejected with a duplicate email message; When we attempted to signup with a duplicate username, we were rejected with a duplicated username message.
* Status (Fail/Pass): Pass

### Test case ID: #126 Testing Secure Login Info

* Description/Summary of Test: In the database, when making a query for a user the results should come out as plain text, however with the sensitive fields being encrypted by a SHA-512 hash
* Pre-condition: A query is made to the database for any/all users
* Expected Results: The query will produce the user and any non-sensitive info will show up as plain text, with the sensitive fields being hid behind a hash.
* Actual Result: The user retrieved from the database had plain text fields apart from the password which was a hash
* Status (Fail/Pass): Pass

### Test case ID: #129 Testing MyTrip Page UI

* Description/Summary of Test: Manual test to make sure the story works properly
* Pre-condition: The MyTrip page has been loaded
* Expected Results: When the user clicks on a trip, trip detail should be shown on the right side of the page. When the user hovers over an icon, all related icons should be highlighted.
* Actual Result: When the user clicks on a trip, trip detail is shown on the right side of the page. When the user hovers over an icon, all related icons is highlighted.
* Status (Fail/Pass): Pass

### Test case ID: #130 Adding Sockets.io

* Description/Summary of Test: We attempted to send a message from the client to the server and then receive a response from the server which echoed our message
* Pre-condition: The client and server both have sockets.io included
* Expected Results: The server will receive the message and echo the message so that it is visible to the client
* Actual Result: Once the message was sent from the client to the server, that message was again displayed in the console of the client
* Status (Fail/Pass): Pass

### Test case ID: #146 MyTrip Page Frontend-logic Test

* Description/Summary of Test: Manual test to make sure the story works properly
* Pre-condition: The MyTrip page has been loaded
* Expected Results: When editing an item, the corresponding change should be shown on page.
* Actual Result: When editing an item, the corresponding change is shown on page.
* Status (Fail/Pass): Pass

### Test case ID: #147 Testing Transaction Calculation Function

* Description/Summary of Test: Manual test to make sure the story works properly
* Pre-condition: The trip has been edited.
* Expected Results: When a user clicks on calculate button, the result should be shown on right side of the screen.
* Actual Result: When a user clicks on calculate button, the result is shown on right side of the screen
* Status (Fail/Pass): Pass

### Test case ID: #157 Sending Emails

* Description/Summary of Test: Given an email address the server will send a message, plaintext or html, to the client
* Pre-condition: A desired email to receive the message
* Expected Results: The user will receive an email from a GoDutch account, with the text/html coming from the server
* Actual Result: The email which was entered received the message from the server, whether it was text or html
* Status (Fail/Pass): Pass

### 

### 

### Test case ID: #159 Testing the Saving and Retrieving Data

* Description/Summary of Test: Manual test to make sure the story works properly
* Pre-condition: The MyTrip page has been loaded
* Expected Results: When editing an item, the corresponding change should be saved into database.
* Actual Result: When editing an item, the corresponding change is saved into database.
* Status (Fail/Pass): Pass

### Test case ID: #167 Testing Real-Time Collaboration

* Description/Summary of Test: Manual test to make sure the story works properly
* Pre-condition: The user has made changes to trips.
* Expected Results: All other users in trip should see the changes.
* Actual Result: All other users in trip see the changes.
* Status (Fail/Pass): Pass

## Integration Test

### Test case ID: #130\_I Testing Sockets IO interaction with angular

* Description/Summary of Test: We wanted to make sure the synchronous property of the sockets did not conflict with the asynchronous property from nodejs/angularJS
* Pre-condition: A user will perform their day-to-day activities and not run into any error which did not occur
* Expected Results: The sockets will not conflict with any of the server or client function
* Actual Result: The application has no conflicts with previous server function
* Status (Fail/Pass): Pass

### Test case ID: #145\_I Testing Sockets IO interaction with angular

* Description/Summary of Test: Testing server/client interaction with the removal of API routes
* Pre-condition: A user will perform their day-to-day activities and not run into any error which did not occur
* Expected Results: When a user performs Login/Logout/Sign In/Signout, or any of the trip functions, the server will be able to interpret the socket end points and perform as well as the restful API implementation
* Actual Result: All previous server functions have no issue, promises were implemented to overcome the issue of asynchronous vs synchronous processes.
* Status (Fail/Pass): Pass

### 

### 

### Test case ID: #159\_I Testing object storage refactor

* Description/Summary of Test: We wanted the front-end model of the page to be the same as the one it would be storing/retrieving. We also wanted it to be lightweight because this model would be sent to all clients when it was changed
* Pre-condition: Previously users could not interact with each other, because in order to describe the change one user made to a trip we would retrieve the individual changes. Now changes are sent as a whole model which has a description of the page
* Expected Results: When a user joins/creates a trip, this trip will be saved to their profile. Along with that any changes one user makes to a trip can be seen changed in real-time for any other user who is apart of the trip
* Actual Result: The page did update in real-time, however has an issue when the speed of changes come in milliseconds. The front page UI still works despite the change to the Angular model used in the frontend
* Status (Fail/Pass): Pass

## Functional Test

### Test case ID: #166\_F Reducing the Save Frequency

* Description/Summary of Test: Without proper implementation of Operational Transformation, there are times when changes that are made to fast can be lost. To counter this we add a purposeful delay.
* Expected Results: A user will no longer lose progress when type changes, and changes will be broadcasted once it has been first saved in the database
* Actual Result: True collaboration is now possible and server stress is now lower when multiple users are connected
* Status (Fail/Pass): Pass

## Performance Test

### Test case ID: #166\_P Testing Ideal Save Frequency

* Description/Summary of Test: With a purposeful delay, some applications begin to feel like they lag (when in fact they aren’t). We tested to see what save frequency was most natural while typing.
* Expected Results: When a user starts typing, they can see the delay (as is visible in software like Google Docs) however it is not enough in order for the application to feel less usable
* Actual Result: There is little to no difference from previous experience in editing trips, and saves to the server are now more uniform
* Status (Fail/Pass): Pass

# 

# Glossary

*Client/Server pattern*: A software architectural pattern that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

*Front Controller pattern*: A software design pattern in which a controller handles all requests for a website. It is a useful structure for Web application developers to achieve the flexibility and reuse without code redundancy.

*Min-Cost-Max-Flow*: A maximum flow with the smallest possible cost. This problem combines maximum flow (getting as much flow as possible from the source to the sink) with shortest path (reaching from the source to the sink with minimum cost).

*Model-View-ViewModel pattern*: A software architectural pattern which facilitates a separation of development of the graphical user interface – be it via a markup language or GUI code – from development of the business logic or back-end logic (the data model).

*Observer pattern*: A software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

*Singleton pattern*: A software design pattern that restricts the instantiation of a class to one object.

*Software architectural pattern*: A general, reusable solution to a commonly occurring problem in software architecture within a given context.

*Software design pattern*: A general reusable solution to a commonly occurring problem within a given context in software design.

# Appendix

## Appendix A - UML Diagrams

### Sprint 1, 2

**User Story #100 Setup the Host with Gulp and Node.js**

|  |
| --- |
| 100.png |
| Figure A1. Use case diagram |

|  |
| --- |
| 100.png |
| Figure A2. Sequence Diagram |

#### User Story #117 Implement MCMF in javascript

|  |
| --- |
| Untitled Diagram.png |
| Figure A3. Use case diagram |

|  |
| --- |
| Untitled Diagram.png |
| Figure A4. Sequence diagram |

#### User Story #124 Implementing Login

|  |
| --- |
| LOGIN.png |
| Figure A5. Use Case Diagram |

|  |
| --- |
| LOGIN_SEQ.png |
| Figure A6. Sequence Diagram |

|  |
| --- |
|  |
| Figure A7. Class Diagram LOGIN_CLASS.png |

#### 

#### 

#### User Story #125 Implementing Signup

|  |
| --- |
| SIGNUP.png |
| Figure A8. Use Case Diagram |

|  |
| --- |
| Untitled Diagram.png |
| Figure A9. Sequence Diagram |

|  |
| --- |
|  |
| Figure A10. Class DiagramSIGNUP_CLASS.png |

### 

### 

### Sprint 3

#### User Story #127 Navigation UI Implementation

|  |
| --- |
| 127 use case.png |
| Figure 11. Use Case Diagram |

|  |
| --- |
| 127 sequence diagram.png |
| Figure A12. Sequence Diagram |

|  |
| --- |
| 127 class diagram.png |
| Figure A13. Minimal Class Diagram |

#### 

#### 

#### User Story #128 Login/Logout/Signup UI Design

|  |
| --- |
| SIGNUP.png |
| Figure A14. Use Case Diagram |

|  |
| --- |
| Untitled Diagram.png |
| Figure A15. Sequence Diagram |

#### 

|  |
| --- |
| SIGNUP_CLASS.png |
| Figure A16. Class Diagram |

#### 

#### 

#### User Story #129 Login/Logout/Signup UI Design

|  |
| --- |
| 129 use case.png |
| Figure A17. Use Case Diagram |

#### 

|  |
| --- |
| 129 sequence diagram.png |
| Figure A18. Sequence Diagram |

|  |
| --- |
| 129 class diagram.png |
| Figure A19. Minimal Class Diagram |

#### User Story #130 Adding Real-Time Requirements

|  |
| --- |
| Real-Time.png |
| Figure A20. Class Diagram |

|  |
| --- |
| Real-Time.png |
| Figure A21. Class Diagram |

|  |
| --- |
| Real-Time-Class.png |
| Figure A22. Class Diagram |

### 

### 

### Sprint 4

#### User Story #145 Refactoring RESTful API to Sockets

|  |
| --- |
| NEW_LOGIN.png |
| Figure A23. Use Case Diagram |

|  |
| --- |
| TokenBased.png |
| Figure A24. Sequence Diagram |

|  |
| --- |
| Real-Time-Class.png |
| Figure A25. Class Diagram |

#### 

#### 

#### User Story #146 Implement MyTrip Page Frontend Logic

|  |
| --- |
| 146.png |
| Figure A26. Use Case Diagram |

|  |
| --- |
| 146 sequence.png |
| Figure A27. Sequence Diagram |

|  |
| --- |
| 146 class.png |
| Figure A28. Class Diagram |

#### 

#### 

#### User Story #147 Implement Routes for Transaction Calculation Functionality

|  |
| --- |
| 147 use case.png |
| Figure A29. Use Case Diagram |

|  |
| --- |
| 147 sequence.png |
| Figure A30. Sequence Diagram |

|  |
| --- |
| 147 class.png |
| Figure A31. Class Diagram |

### 

### 

### Sprint 5

#### User Story #157 Emailing Service

|  |
| --- |
| EMAIL.png |
| Figure A32. Class Diagram |

|  |
| --- |
| Email-Sequence.png |
| Figure A33. Sequence Diagram |

|  |
| --- |
| Emailing-Class.png |
| Figure A34. Sequence Diagram |

#### 

#### 

#### User Story #158 Content update for Home/About/Contact page

#### 

|  |
| --- |
| CONTENT.png |
| Figure A35. Use Case Diagram |

#### 

#### 

#### User Story #159 Refactor object storage to enable saving and retrieving data

|  |
| --- |
| 146.png |
| Figure A36. Use Case Diagram |

|  |
| --- |
| 159 sequence.png |
| Figure A37. Sequence Diagram |

|  |
| --- |
| 146 class.png |
| Figure A38. Sequence Diagram |

#### 

### 

### Sprint 6

#### User Story #166 Reduce the Saving Frequency

|  |
| --- |
| 146.png |
| Figure A39. Use Case Diagram |

|  |
| --- |
| 166 sequence.png |
| Figure A40. Sequence Diagram |

|  |
| --- |
| 166 class.png |
| Figure A41. Class Diagram |

#### User Story #167 Broadcast Changes of a Trip to All Its Users

|  |
| --- |
| 167 usecase.png |
| Figure A42. Use Case Diagram |

|  |
| --- |
| 167 sequence.png |
| Figure A43. Sequence Diagram |

|  |
| --- |
| 167 class.png |
| Figure A44. Class Diagram |

#### 

#### 

#### 

## Appendix B - User Interface Design

|  |
| --- |
| Login.png |
| Figure B1. Initial Landing Page |

|  |
| --- |
| D5.png |
| Figure B2. Logged In Landing Page |

|  |
| --- |
| contact.png |
| Figure B3. Contact Us Page |

|  |
| --- |
| email.png |
| Figure B4. Example Email Invite Link |

|  |
| --- |
| D6.png |
| Figure B5. Initial Trip View |

|  |
| --- |
| Trip.png |
| Figure B6. Trip with example values |

## 

## 

## 

## Appendix C - Sprint Review Reports

### ***Sprint 1, 2***

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 2:45pm, 02/14/2017

End time: 3:00pm, 02/14/2017

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story <117 Implement MCMF in JS>
* User Story <100 Setup host using Gulp and Node.js>
* User Story <123 Research in UI Design>
* User Story <124 Login>
* User Story <125 Sign up>
* User Story <126 Secure Login Information>

### Sprint 3

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 2:45pm, 02/28/2017

End time: 3:00pm, 02/28/2017

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story <127>
* User Story <128>
* User Story <129>
* User Story <130>

### Sprint 4

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 2:45pm, 03/12/2017

End time: 3:00pm, 03/12/2017

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story <145>
* User Story <146>
* User Story <147>

### Sprint 5

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 2:45pm, 03/27/2017

End time: 3:00pm, 03/27/2017

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story <157>
* User Story <158>
* User Story <159>

### Sprint 6

Attendees: Zhenglin Pan, Jonathan Beltran, Deng Pan

Start time: 2:45pm, 04/11/2017

End time: 3:00pm, 0411/2017

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story <166>
* User Story <167>
* User Story <173>
* User Story <174>

## 

## 

## Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

# User Manual: Using the basic functions of the site

## Signing Up

When first arriving to the GoDutch site you will see this screen (Figure D1).

|  |
| --- |
| Login.png |
| Figure D1. Landing Page |

Enter your desired username (Figure D2).

|  |
| --- |
| D1.png |
| Figure D2. Username Field |

Reasons for why your username may be rejected can be because someone else has already chosen that username.

Enter your desired Email (Figure D3).

|  |
| --- |
| D2.png |
| Figure D3. Email Field |

Reasons for why your email may be rejected can be because someone else has already registered for the site using that email

Enter your password (Figure D4) and then confirm your password.

|  |
| --- |
| D3.png |
| Figure D4. Password and Re-enter Password Field |

If you successfully signup, you will automatically be logged in and be taken to this page (Figure D5) other you will receive the errors mentioned before.

|  |
| --- |
| D5.png |
| Figure D5. Logged HomePage |

## 

## 

## Creating a Trip

When you are signed in to GoDutch this will be your main view (Figure D6), go to the MyTrip tab. If this is your first time here you will have an empty page, to create a trip click on the highlighted button (Figure D7).

|  |
| --- |
| D6.png |
| Figure D6. Empty Trip View |

|  |
| --- |
| D7.png |
| Figure D7. Create a trip |

You can also give your trip a name (Figure D8).

|  |
| --- |
| D8.png |
| Figure D8. Trip Name Field |

## 

## 

## Inviting Friends

If you have a trip created, you can invite your friends to it. Simply select the trip (Figure D9) and type in your friend’s email and send it off (Figure D10). They do not need to have an account on the site in order to be invited.

|  |
| --- |
| D9.png |
| Figure D9. Select a trip |

|  |
| --- |
| D10.png |
| Figure D10. Send Email Field |

## Viewing Trips

The trips are located on the left hand side of the page (Figure D11). By clicking on the photo of the trip you want to view, you are selecting the trip.

|  |
| --- |
| D11.png |
| Figure D11. Change Trips |

## 

## 

## Editing Trips

To add an item to your trip click on the button in the item section (Figure D12). You can add a name to the item, how much it cost, and your friends can put whether they used the item or not (Figure D13).

|  |
| --- |
| D12.png |
| Figure D12. Add Item button |

|  |
| --- |
| D13.png |
| Figure D13. Username Field |

More importantly the cost of the item is one way, that is your friends won’t see the price that you put on the item. Rather the price you put is how much you contributed to the purchase of the item.

## Calculating Trips

To calculate the price of the trip, and what everyone should be paying, click on the calculate button (Figure D14). There you can find who owes whom money (Figure D15).

|  |
| --- |
| D14.png |
| Figure D14. Calculate Button |

|  |
| --- |
| D15.png |
| Figure D15. Total Transaction Window |

# 

# 

# Installation

For this application the software that was used was NodeJS, MongoDB, and GitHub. In the next sections there will be a tutorial on how to install each of the essential software and Windows, Mac, and Linux.

**Windows**

We will begin by installing GitHub for windows because it is required to use both NodeJS and MongoDB.

Begin by installing GitHub for windows from the official github site <https://desktop.github.com/>. After you have installed it use Windows PowerShell, which can be accessed by pressing the Windows key or Windows search bar and typing PowerShell.

To ensure it installed correctly open up PowerShell and type git.

Now we are going to install NodeJS. Download the NodeJS installer from the official NodeJS site <https://nodejs.org/en/> and download the build which is recommended for most users.

After downloading the installer, run the installer and keep all the default settings it suggest. NodeJS will require a restart to complete its installation.

Once the computer is restarted, open PowerShell and type node -v followed by npm -v. If both of these return without error than NodeJS has successfully been installed on the computer.

Finally we are going to install MongoDB. Download the MongoDB installer from the official site <https://www.mongodb.com/download-center#community> any community server version will work.

Once MongoDB is installed, open PowerShell and move to where MongoDB was installed

* When you open powershell the default directory is C:\Users\Username
* Change directory back twice, type cd .. twice
* Create a directory called data, and then a directory called db by typing mkdir ‘data\db’
* Now change directory in Program Files, cd ‘Program Files’
* Change directory into MongoDB’s bin folder, cd ‘MongoDB\Server\<server version>\bin’
* Execute ‘mongodb --install --logpath=”C:\data\db\log.txt” dbpath=”C:\data\db” ‘ in the directory

Once you have executed in this manner, the next time you want to start the MongoDB server all you will need to type is net start mongodb, of course if this doesn’t work you could always start mongodb using the final step but only typing mongod.exe

Now you have all the required software for this project.

**Mac**

We will begin by installing GitHub for windows because it is required to use both NodeJS and MongoDB.

Before beginning we will need to install XCode which can be found in the App store. After you have installed XCode, open up the terminal and type in

* /usr/bin/ruby -e "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install>)"

This will install Homebrew to your terminal.

Begin by installing GitHub for Mac from the official github site <https://desktop.github.com/>. After you have installed it open the application and click on the advance tab>Install Command Line Tools.

To ensure it installed correctly open up the terminal and type git, it should return options

Now we are going to install NodeJS. Simply type brew install node. Type node -v followed by npm -v. If both of these return without error than NodeJS has successfully been installed on the computer.

Finally we are going to install MongoDB. Simply type brew install mongodb. You will need to create a /data/db folder in your default directory, do this by typing mkdir ~/data/db. You will also need to give mongodb permission to use this folder, do so by typing sudo chown -R `id -un` /data/db. Your terminal will prompt you for your password.

To ensure MongoDB is install type mongod, to keep the server on type mongod &.

Now you have all the required software for this project.

**Linux**

We will begin by installing GitHub for windows because it is required to use both NodeJS and MongoDB. Open up the terminal on ubuntu and do the following:

To install GitHub type sudo apt-get install git

To install NodeJS type

curl -sL https://deb.nodesource.com/setup\_6.x | sudo -E bash -  
sudo apt-get install -y nodejs

sudo apt-get install -y build-essential

To install MongoDB type

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927

(this will install the mongo directory)

sudo apt-get update

sudo apt-get install -y mongodb-org

Now you have all the required software for this project.

## Installing GoDutch

Go to your terminal (Powershell, Mac shell, etc.) and clone our repository:

Git clone <https://github.com/FIU-SCIS-Senior-Projects/GoDutch-1.0>

Next execute npm install or npm update, this will install the dependencies from the package.json file. If you wish to install any new add-ons/libraries type npm install <library> --save and it will automatically be saved into package.json.

You will need to set up a config.js file in the server/config/ directory before you can run this application. The config.js file should look like this (Figure 18).

You will also need to execute npm install gulp -g to install the gulp command globally.

Under server/config/ create a file named config.js that has the following content:

//setup for mongoDB

var config = {

port: 27017,

db: 'mongodb://localhost/default',//replace default for db name

host: '**<computer name>**:8080',

jwtSecret: 'devSechsecret',

sessionSecret: 'devSech',

expire: 60\*60\*24\*60

};

module.exports = config;

Note: Replace **<computer name>** with the DNS name for the server (if you are testing in a LAN, you can use your own computer name instead).

Now open three terminals, one to run the node server, one to run the gulp server, and finally one to run the mongo server.

In the first terminal type gulp quick. This is a setting that is written in the gulp file in the repository. Gulp will watch the server files for any changes to front-end code.

In the second terminal type mongod and leave the server running.

In the third terminal type node server, the server will begin on localhost:8080, if you go to <http://localhost:8080> you will find the webpage.

# 

# 

# ShortComings/Wish List:

In the current system, we have the following shortcomings:

* Defective real-time collaboration

The current system uses artificial delay to stagger user requests to reduce the risk of concurrency issues

* Unresponsive user interface

The current system is not giving sufficient user feedback when users interact with the page.

* Lack of page contents

The current system focuses on the main functionalities of MyTrip page, while the other pages such as Home, ContactUs and About are dummy pages that have no functionalities implemented.

For future versions of GoDutch, we would like to include the following functionalities:

* True collaboration implemented with Operational Transformation, this can be implemented with a library called ShareJS
* Making the trip view more robust
* Allow chat messaging between members of the site
* Implement Changelog on homepage, similar to GitHub tracking of commits
* Create a mobile app for easy access
* Link to payment system such as Venmo or Paypal
* Make the site secure
* Allow for scanning of receipts for easy item creation