

IS213 Enterprise Solution Development

Team Members:

Deborah Lee

Lee Rui Peng

Rachel Lui

Pooja Bharath Kumar

Neo Shi Shiong

Sherman Lee

# 

NAP Contents Page

[Introduction 1](#_Toc36993791)

[Business and User main Scenario 1: User registers for a new account 1](#_Toc36993792)

[Business and User main Scenario 2: User creates a namecard 1](#_Toc36993793)

[Business and User main scenario 3: User selects email recipients and sends emails 1](#_Toc36993794)

[Business and User main Scenario 4: User finds jobs using Job Scraper 1](#_Toc36993795)

[Technical Overview Diagram 1](#_Toc36993796)

[User Scenarios 2](#_Toc36993797)

[Main User scenario 1: User registers for a new account 2](#_Toc36993798)

[Sub User scenario 1.1: User logs in to account 2](#_Toc36993800)

[Main User scenario 2: User creates a new namecard 3](#_Toc36993803)

[Sub User scenario 2.1: User retrieves namecard details 3](#_Toc36993804)

[Main User scenario 3: User chooses email recipients 4](#_Toc36993806)

[Sub User scenario 3.1: User sends email to desired recipients 4](#_Toc36993808)

[Beyond the Labs used for scenario 3 5](#_Toc36993811)

[Main User scenario 4: User finds jobs using job scraper 6](#_Toc36993812)

[Beyond the Labs used for scenario 4 6](#_Toc36993815)

[Remaining Beyond the Labs not covered above 6](#_Toc36993816)

# 

# Introduction

NAP, short for Networking Assistance Platform is a centralised website to assist users in registering and viewing multiple name cards that they have, instead of keeping multiple physical copies of them. NAP also uses techniques like *data scraping* to display the latest internships and jobs from various websites.

# Business and User main Scenario 1: User registers for a new account

The user inputs personal details as required by the NAP UI when signing up for an account. These details are then registered into the **User database** when the user clicks “Signup”.

# Business and User main Scenario 2: User creates a namecard

The user will input the name card details as required on the “Add Namecard” page. These details are then registered into the **Namecard database** when the user clicks “Create Namecard”. The phone number field needs to be integers.

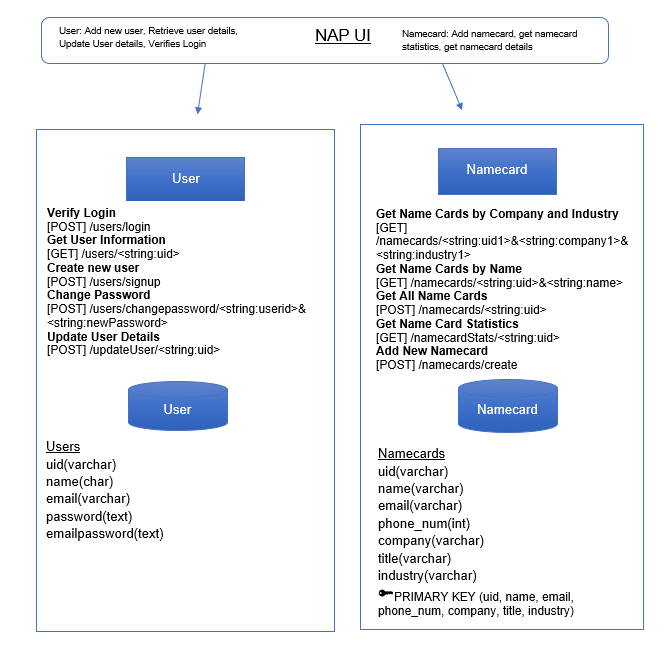
# Business and User main scenario 3: User selects email recipients and sends emails

The UI will display a drop-down list of all the user’s namecards in the **Namecard database**. The user can filter the namecards by industry or company and select desired contacts to send the email to. The user then crafts an email, which the application sends out to each recipient with their name in the salutation. (e.g. “Dear Michael”)

# Business and User main Scenario 4: User finds jobs using Job Scraper

The user looks for jobs on NAP by keying in a specific keyword (e.g. Analytics Internship). The job scraper then scrapes Indeed and Internjobs, returning a list of the job titles and links for the user to choose from.

Technical Overview Diagram



# 

# User Scenarios

## Main User scenario 1: User registers for a new account

## 

1. Create an account with personal details

{uid,name,email,password,emailpassword}

HTTP POST

User Microservice

NAP UI

2. Returns notification of account creation

## Sub User scenario 1.1: User logs in to account

## 

1. Log in with personal details

{email, password }

HTTP POST

User Microservice

NAP UI

2. Returns login status/ redirects to home page

**Detailed Steps for this scenario:**

1. User registers for a new account by clicking on “Sign up here” button, he is redirected to sign up page and inputs all the details required by the sign-up page: **uid, name, email, password, email password**. Then, user clicks the “Sign up” button and the UI invokes the user microservice via HTTP POST to add the new user.
2. Upon receiving the UI request, user microservice hashes the password and adds received user details into User database to create a new user. The microservice returns status of the user creation to the UI.
3. If the user account has been successfully created, the user inputs his email address and password on the login page to log in. invokes the user microservice via HTTP POST to verify the user.
4. Upon receiving the UI request, user microservice checks if username(uid) exists and correctness of password. Status of the login is returned to the UI.
5. If successful login, user is redirected to NAP homepage and the uid is stored in a session to be used again. Else, user will be notified and prompted to try again.

### 

### Microservice Used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service Name | Operational information | Description of the functionality | Input | Output |
| User | **Verify Login**  [POST] /users/login | Not applicable as other details show functionalities clearly | {uid,name,email,password,  emailpassword} | Status of account creation |
| **Create new user**  [POST] /users/signup | {email, password} | Status of login |

## 

## Main User scenario 2: User creates a new namecard

1. Enter new name card information

{uid, Name, company, phone\_num, email, title,industry}

HTTP POST

Namecard Microservice

NAP UI

2. Display name card created and notification of successful name card addition

## Sub User scenario 2.1: User retrieves namecard details

1. Send namecard filter details and uid

{uid, name}

HTTP POST

Namecard Microservice

NAP UI

2. Return namecard details with matching uid and name

{uid, name, [phone\_num](http://localhost/phpMyAdmin/sql.php), company ,title, industry, email}

**Detailed Steps for this scenario:**

1. User creates a new namecard by clicking on the “Add Namecard” button on the Namecards page.
2. User inputs all the required fields as required on the add namecard page: **name, phone\_num, company, title, industry, email**
3. The NAP UI then invokes the namecard microservice via HTTP POST to add the new namecard.
4. Upon receiving UI request, namecard microservice adds all received namecard details into the namecard database and returns a status of namecard creation to the UI.
5. The user can find a specific namecard from all his namecards by searching the name in the search box.
6. The UI then invokes the namecard microservice via HTTP GET to display the searched namecard.
7. Upon receiving UI request, namecard microservice returns matching all matching namecard(s).
8. UI displays list of matching namecard(s)

### 

### Microservice used:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service Name | Operational information | Description of the functionality | Input | Output |
| Namecard | **Get Name Cards by Name**  [GET] /namecards/<string:uid>&  <string:name> | Not applicable as operational information shows functionalities clearly | {uid, name} | {name, [phone\_num](http://localhost/phpMyAdmin/sql.php), company ,title, industry, email} |
| **Get All Name Cards**  [GET] /namecards/<string:uid> | { uid } | {name, [phone\_num](http://localhost/phpMyAdmin/sql.php), company ,title, industry, email} |
| **Add New Namecard**  [POST] /namecards/create | {uid, name, [phone\_num](http://localhost/phpMyAdmin/sql.php), company ,title, industry, email} | {status code of namecard creation} |

## 

## Main User scenario 3: User chooses email recipients

## 

1. Send namecard filter details and uid

{uid, company, industry}

HTTP GET

Namecard Microservice

NAP UI

2. Return namecard details with matching uid, industry, and company

{uid, name, [phone\_num](http://localhost/phpMyAdmin/sql.php), company ,title, industry, email}

## Sub User scenario 3.1: User sends email to desired recipients

1. Send email message to desired recipients

{ uid, name, emailcheck, emailname, emailsubject, email\_message }

NAP UI

## 

User Microservice

2. Query user email and email password {uid}

HTTP GET

HTTP GET

3. Returns user email and email password

{Useremail, emailpassword}

5. Returns emails sent notification

Emailing Microservice

AMQP

4. Send emails

{Useremail, emailpassword, emailcheck, emailname, emailsubject, email\_message }

Email Sender Microservice

**Detailed Steps for this scenario:**

1. The user chooses email recipients by filtering the namecards based on the company and industry. The UI then invokes the namecard microservice via HTTP GET which returns the matching namecard details.
2. Upon selection of recipients, the user crafts the email and sends it out, triggering the UI to invoke the emailing microservice via HTTP GET to send the email messages to each recipient.
3. Upon receiving UI request, emailing microservice invokes the user microservice via HTTP GET to get the user’s email and email password.
4. The emailing microservice then loops through all email recipients to assemble an email with each recipients’ name in the salutations of each email. Each email message is published to the exchange with the same routing key.
5. The routing key reaches 1 queue and multiple instances of EmailSender microservice listen to this queue for load balancing. Each EmailSender microservice invokes Gmail SMTP server to send an email to the recipient.
6. After the emails are sent, the UI notifies the user of email sent confirmation.

### Microservice Used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service Name | Operational information | Description of the functionality | Input | Output |
| Namecard | **Get Name Cards by Company and Industry**  [GET] /namecards/<string:uid1>&<string:company1>  &<string:industry1> | Gets list of namecards with matching company and industry | {uid, company, industry} | {uid, name, [phone\_num](http://localhost/phpMyAdmin/sql.php), company ,title, industry, email} |
| Emailing | **Send Email**  [GET] /emailing/<string:emailcheck>  &<string:emailname>  &<string:emailsubject>  &<string:emailmessage>  &<string: uid> | Assembles each email message with recipient’s name in salutation. publishes each email message to the exchange with the same routing key | {emailcheck, emailname, emailsubject, emailmessage, uid} | {notification of emails sent} |
| User | **Get User Information**  [GET] /users/<string:uid> | Gets the user email and password with matching uid | {uid} | {Useremail, emailpassword} |
| Email Sender | Direct Exchange with  (BKEY) gmail.email | Retrieve email message and sends each email to recipient email via Gmail SMTP server | {Useremail, emailpassword, name, email, email\_message} | Nil |

## Beyond the Labs

1. Using SMTP server in this scenario is a unique feature used in this scenario. The user doesn’t have to navigate to his own personal email account to send multiple emails. Instead, the SMTP server is invoked when the user clicks “Send”.

2. Emailing is a composite microservice that orchestrates User microservice and EmailSender microservice.

## Main User scenario 4: User finds jobs using job scraper

## 

Intern Jobs

2. Send request to internjobs.com

{keyword}

1. Get job/intern roles and links

{keyword}

NAP UI

HTTP GET

HTTP GET

3. Returns top search results

{Job Title, URL}

4. Display top results

{Job Title, URL}

HTTP GET

Job Microservice

2. Send request to indeed.com

{keyword}

Indeed

3. Return top results

{Job Title, URL}

**Detailed Steps for this scenario:**

1. The user inputs keywords like “Data Analytics” into the search field, invoking job microservice via HTTP GET.
2. Upon receiving the UI request, job microservice uses the keyword to send HTTP GET requests to internjobs.com and indeed.com to get their top job openings on their results page.
3. The Job microservice retrieves all the text from the two sites’ result page and uses python package BeautifulSoup to extract the top job titles and their URL links by searching for the relevant HTML tags. It compiles them and sends them back to the UI to display.
4. The user can click on any URL link to be redirected to the job posting to find out more or apply.

### Microservice Used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service Name | Operational information | Description of the functionality | Input | Output |
| Job | **Search for jobs**  [GET]  /jobs/<string:keyword> | Scrapes job boards from *internjobs.com* and *indeed.com* with keyword provided by user and returns list of top results | {keyword} | {Job Title, URL} |

## Beyond the Labs used for scenario 4

Finding Jobs by **scraping** job boards is a unique feature used in this scenario. The job microservice takes the keywords that the user inputs and queries websites like internjob.com and indeed.com. Job microservice then extracts the links and job titles from the result pages using a python package called BeautifulSoup.

## Remaining Beyond the Labs not covered above

Separately deployed docker containers were used for most of the microservices i.e. User, Namecard and Job. These docker containers enabled easy sharing of the microservices and their dependencies with others, enabling them to run the microservices on their machines too.