4.1 Requirements

*API Test Tool* is a web application inspired by *Postman*, which is an API and network test tool. The main function is to send different types of request and all the request response information can be displayed. The detailed functions are as follows.

4.1.1 TEST API

There are four kinds of authorization types, namely *Normal*, *Basic Auth*, *Digest Auth* and *OAuth 1.0*. Depending on the type of authorization, the inputs are different. For all the four types, the input URL, interface request method (GET, POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK and PURGE), URL params and Headers are essential.

Besides, for the other three types except *Normal*, the authorization headers are needed and it will be generated and added as a custom header. Specifically, for *Basic Auth*, the authorization headers are Username and Password; for *Digest Auth*, the authorization headers are Username, Realm, Password, Nonce, Algorithm, qop, Nonce count, Client nonce and Opaque; for *OAuth 1.0*, the authorization headers are Consumer key, Consumer secret, Token, Token secret, Signature Method, Timestamp, Nonce, Version and Realm.

4.1.2 HISTORY

HISTORY is a storage of the request history using the form of “request + URL”. It stores all the request information in *4.1.1 TEST API*, and each entry in it can be deleted individually or completely.

4.1.3 COLLECTIONS

In COLLECTIONS, new collections can be created and multiple requests can be categorized and stored in different collections. Also, collections can be import from disk or URL. The existing collections can be edited, deleted and shared as a link or downloaded as a JSON file.

4.1.4 INFORMATION

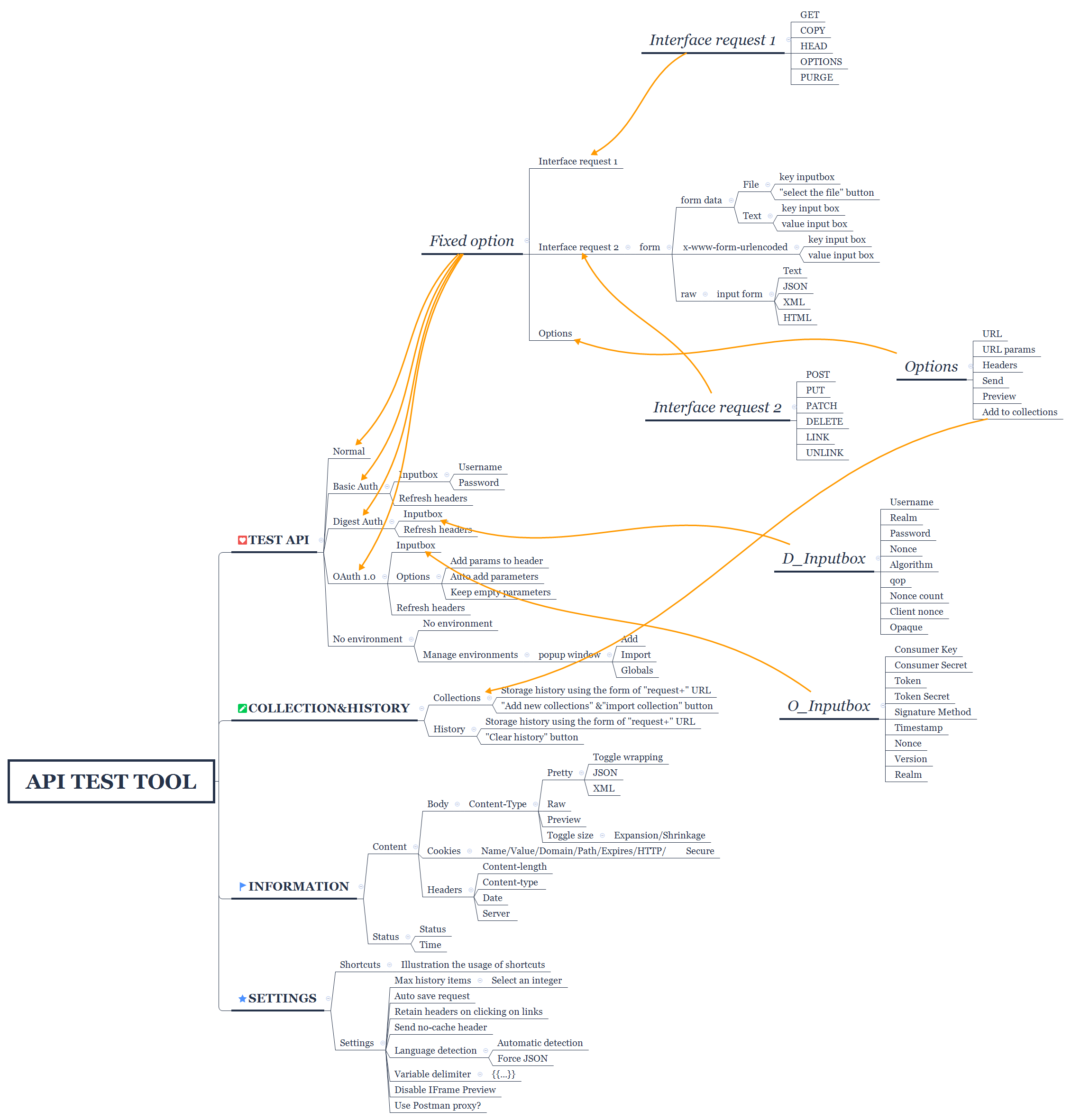
INFORMATION shows the response to the request in *4.1.1 TEST API*, which includes Body, Cookies, Headers and Status. In Body, the response information can be shown in different content-type, including Pretty (JSON or XML), Raw and Preview. The cookies information and the headers information in response are displayed in Cookies and Headers, respectively. In Status section, there is the response time and the response status in the form of HTTP status code.

4.1.5 SETTINGS

In SETTINGS section, the shortcuts can be used and the basic preferences (such as *Max history items*, *Auto save request*, *Send no-cache header*, Language detection and so on) can be set and modified.

4.1.6 Testcase

The testcase of the above requirements is shown in Figure 4.1.



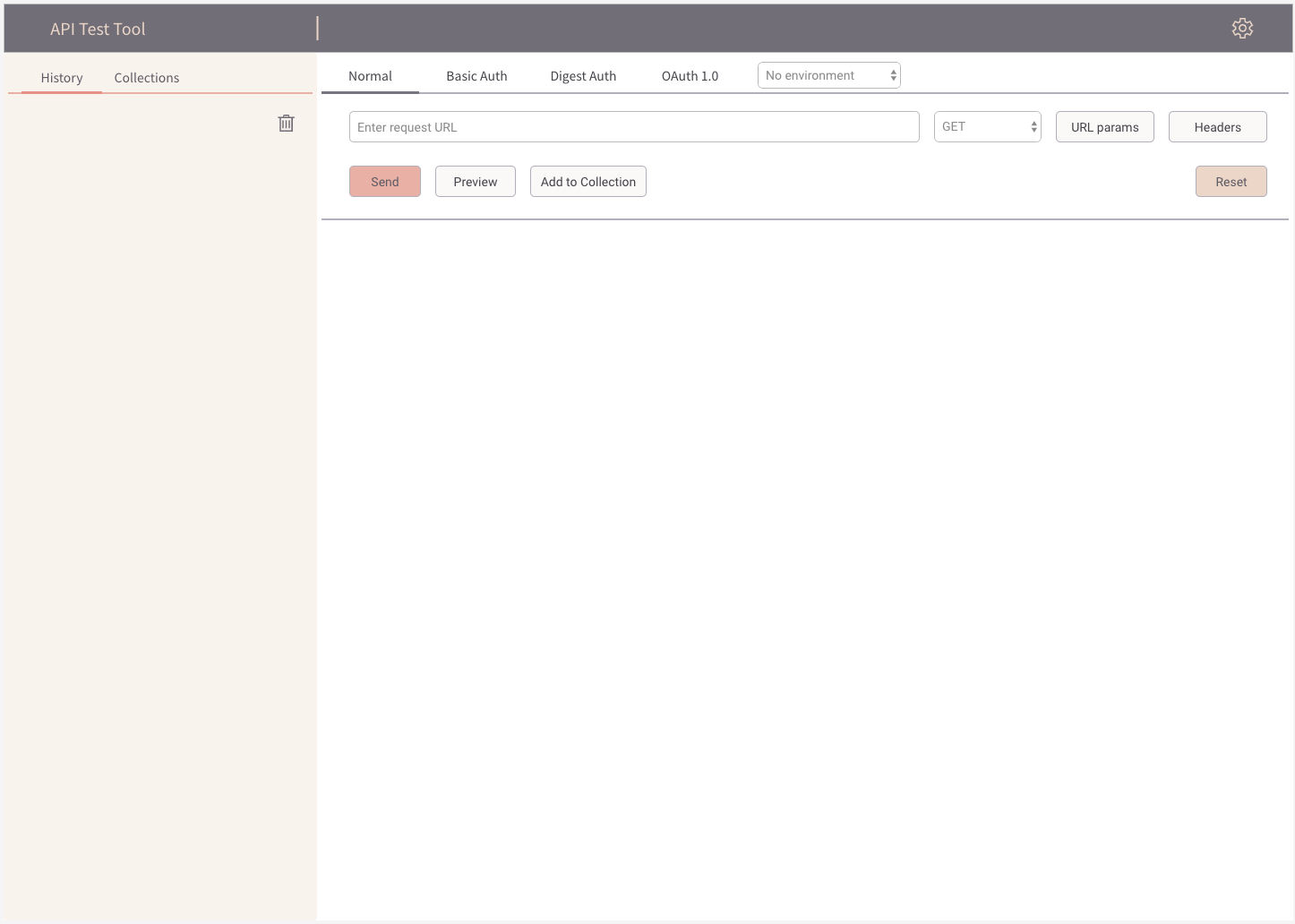
**Figure 4.1** Testcase

4.2 Design

In accordance with requirements, the prototyping, workflow, use case diagram, state diagram, sequence diagram, activity diagram and database designed are shown in detail in this section.

4.2.1 Prototyping

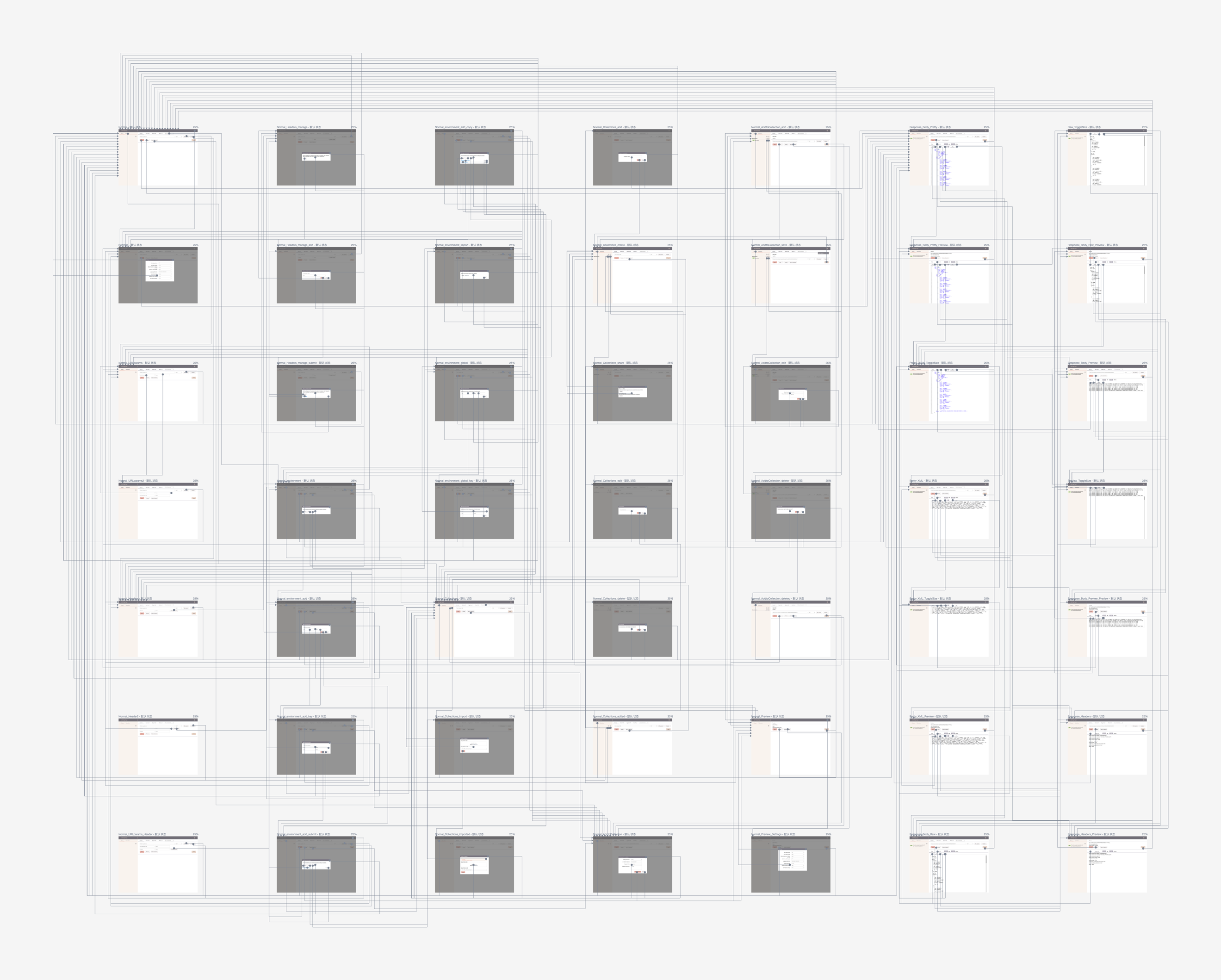
It is a web application inspired by *Postman*. An example of the prototype is shown in Figure 4.2., and the detailed prototype is in our Git repository.



**Figure 4.2** Prototyping

4.2.2 Workflow

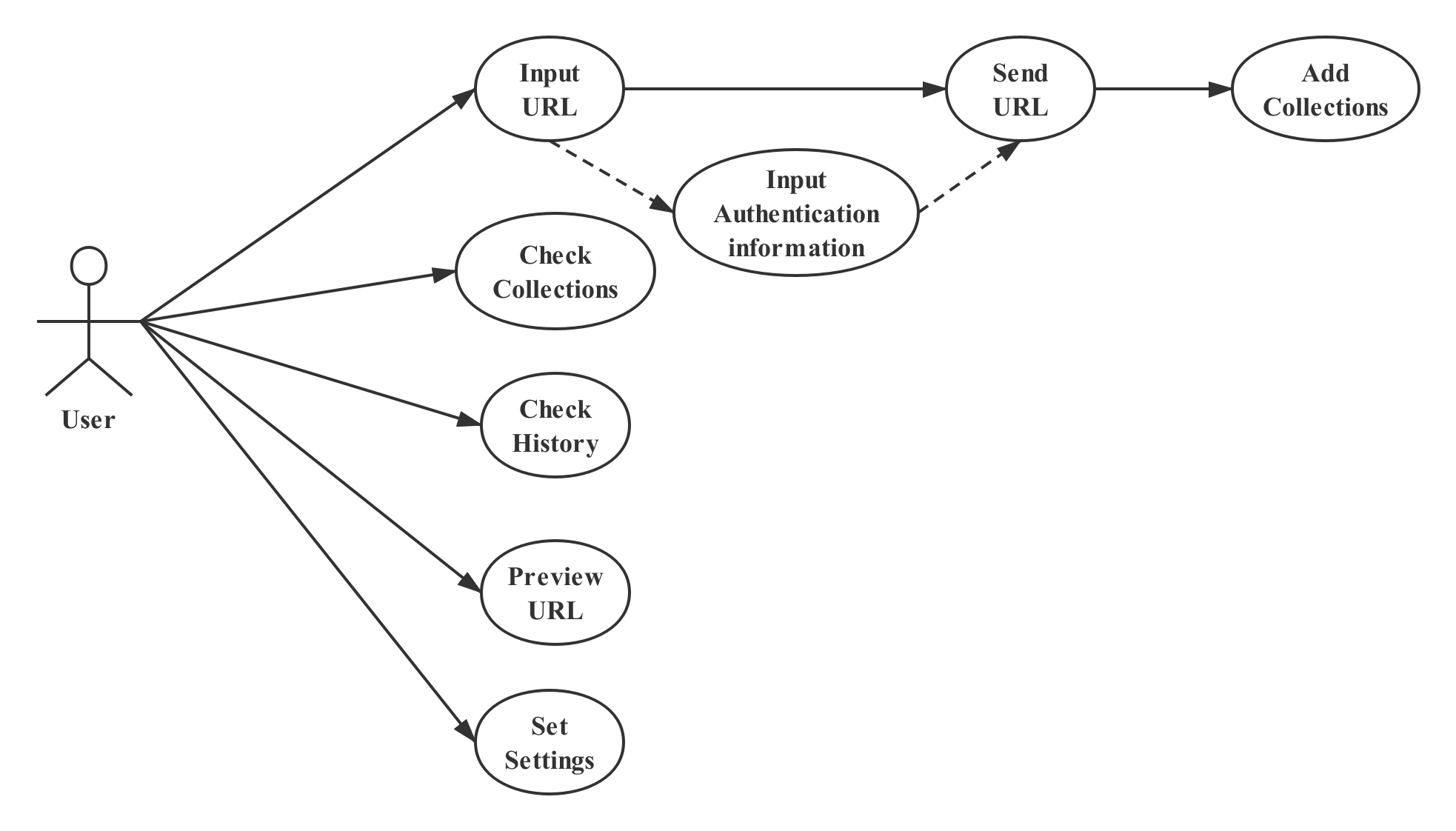
The workflow generated from the prototype design is shown in Figure 4.3.



**Figure 4.3** Workflow

4.2.3 Use case diagram

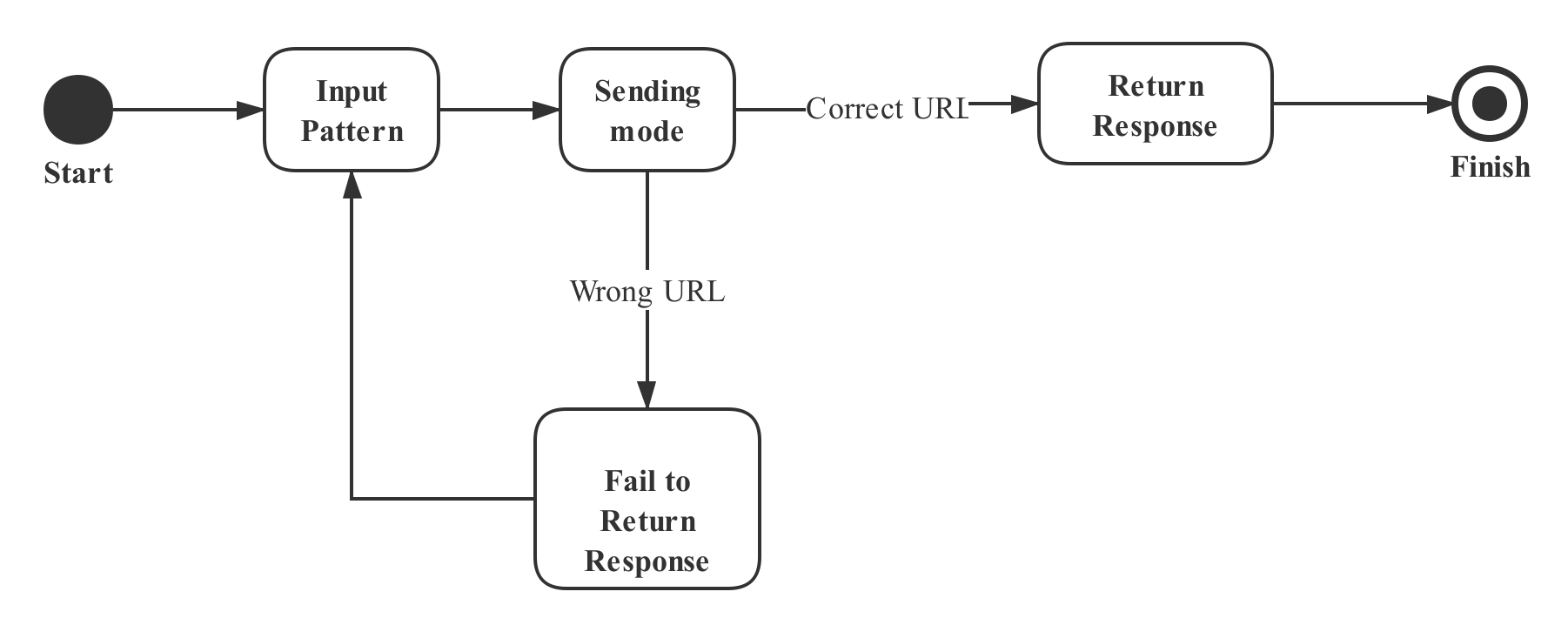
The use case diagram is shown in Figure 4.4. As the use case diagram shown, there is only one participant, the User, and there is the following use case. Firstly, the user *Input URL* to choose the request method, and input the request URL and other request information (such as authentication information) to finish the whole request information. And then user *Send URL* to send the request to get the response information. Meanwhile, the user can also *Add Collections* to add the request to a collection. Secondly, user can *Check Collections* to do the operations (such as edit, deletion, import and share) on the existing or new collections. Thirdly, user can *Check History* to view the history list. Lastly, user *Set Settings* to set the shortcuts and basic preferences.



**Figure 4.4** Use case diagram

4.2.4 State diagram

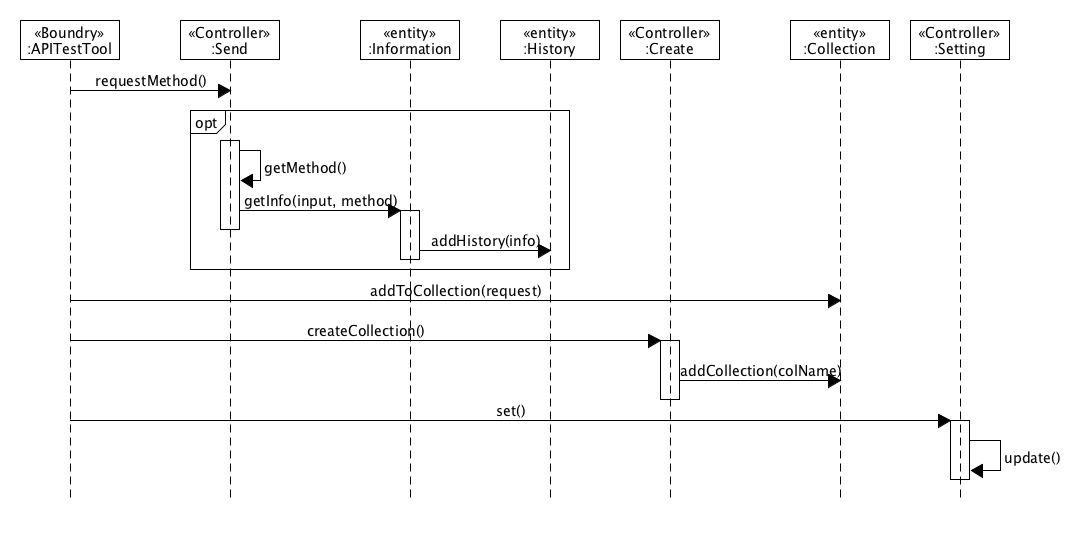
The state diagram is shown in Figure 4.5. As the state diagram shown, the initial state is *Input Pattern*, and the system waits for user input in this state. After the system obtains the user’s input, it changes to *Sending mode*, and it waits for response from the server. If the user’s input is a wrong URL, the system *Fail to Return Response* and return to *Input Pattern* state. Otherwise, if the user’s input is correct, the state changes to *Return Response*, and the whole process finishes.



**Figure 4.5** State diagram

4.2.5 Sequence diagram

The sequence diagram is shown in Figure 4.6. As the sequence diagram shown, *APITestTool* sends the request message to the controller *Send*, and it calls *getMethod()* function to get the optional request method, and calls *getInfo(input, method)* function to get the response information, and the information is added to the history. Besides, *APITestTool* can calls *addToCollection(request)* function to add the request to an existing collection, or calls *createCollection()* to create a new collection. To set the shortcuts and basic preferences, system calls *set()* function, and the controller calls *update()* function to modify the current settings.



**Figure 4.6** Sequence diagram

4.2.6 Activity diagram

4.2.7 Database design

The History table design, Collection table design and Method table design in database are shown in Table 4.1, Table 4.2 and Table 4.3, respectively.

**Table 4.1** History Table Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table Name** | **history** | | | |
| **Attribute** | **Type** | **Null/Non-Null** | **constraint condition** | **Description** |
| **id** | int(11) | Non-Null | Primary key | The id of history  auto-increment |
| **method** | varchar(10) | Non-Null |  | method of request |
| **request** | varchar(200) | Non-Null |  | The URL of request |
| **body** | json | Null |  | content of response |
| **cookie** | json | Null |  | Cookie of response from server |
| **header** | json | Null |  | header of response from server |
| **date** | datetime | Null |  | The date of request |
| **time** | int(11) | Null |  | The time of response |

**Table 4.2** Collection Table Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table Name** | **collection** | | | |
| **Attribute** | **Type** | **Null/Non-Null** | **constraint condition** | **Description** |
| **id** | int(11) | Non-Null | Primary key | The id of collection  auto-increment |
| **method** | varchar(10) | Non-Null |  | method of request |
| **request** | varchar(200) | Non-Null |  | The URL of request |

**Table 4.3** Method Table Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table Name** | **method** | | | |
| **Attribute** | **Type** | **Null/Non-Null** | **constraint condition** | **Description** |
| **id** | int(11) | Non-Null | Primary key | The id of method |
| **method** | varchar(10) | Non-Null |  | method of request |

4.3 Implementation

4.3.1 Content

4.3.2 Server