

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе
Дисциплина: Разработка сетевых приложений
Тема: IMAP-клиент

Выполнили студенты гр. 43501/3

Мальцев М.С.

Преподаватель

Зозуля А.В

«_____» _____ 2019 г.

Санкт-Петербург
2019

1 Цель работы.

Разработка IMAP-клиента.

2 Ход работы

2.1 Архитектура приложения

При разработке приложения было решено выделить четыре сущности:

1. **View** - часть приложения, основная задача которой - это взаимодействие с пользователем. Для написания этого компонента было решено использовать HTML, CSS и JS (чистый).
2. **Http-Server** - выполняет задачу промежуточного слоя между компонентом View, с которым работают пользователи и компонентом IMAP-Client, который реализует основную функциональность приложения. Для создания этой части приложения был выбран язык Golang и библиотека для работы сетью "net". Она позволяет создать готовый http сервер, с помощью её пакета "net/http".
3. **Database** - простой компонент, задача которого сохранить введенные пользователем данные аутентификации. Написан на языке Golang.
4. **IMAP-Client** - основная часть приложения, задача которой установка соединения с IMAP сервером и получение от него почтовых данных, с последующей передачей их компоненту Http-server, с целью вывода их пользователю. Написан на языке Golang. Используются пакеты "crypto/tls" - для установки соединения, пакет "mime" и несколько декодеров, для перевода текста письма в читаемый человеком вид.

Также в приложении использовался пакет "github.com/logrusorgru/aurora" для дифференциации логов по цветам.

На рисунке 2.1 представлены связи между выделенными компонентами.

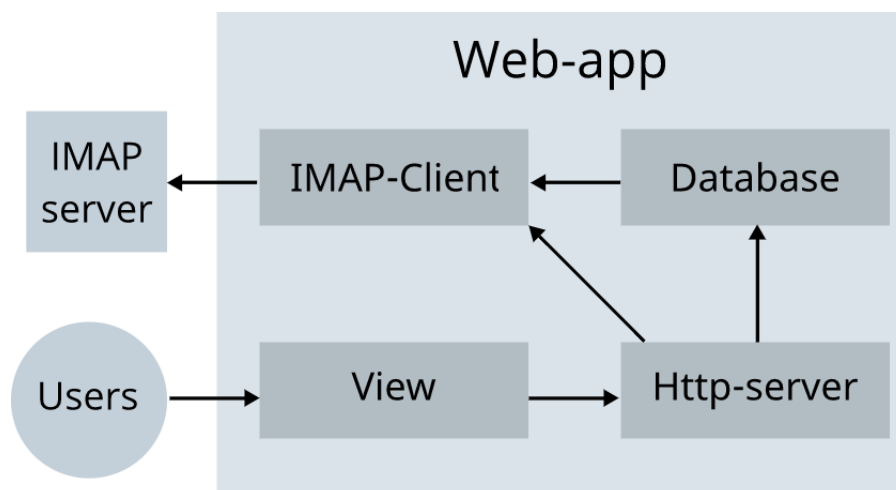


Рис. 2.1: Компоненты приложения и связи между ними.

Таким образом, было решено создать веб-приложение, которое позволяет подключиться с помощью браузера и получить почтовые данные из указанного IMAP сервера.

2.2 Поддерживаемые команды IMAP протокола

Реализованный IMAP клиент поддерживает следующие команды, соответствующие протоколу IMAP (RFC 3501):

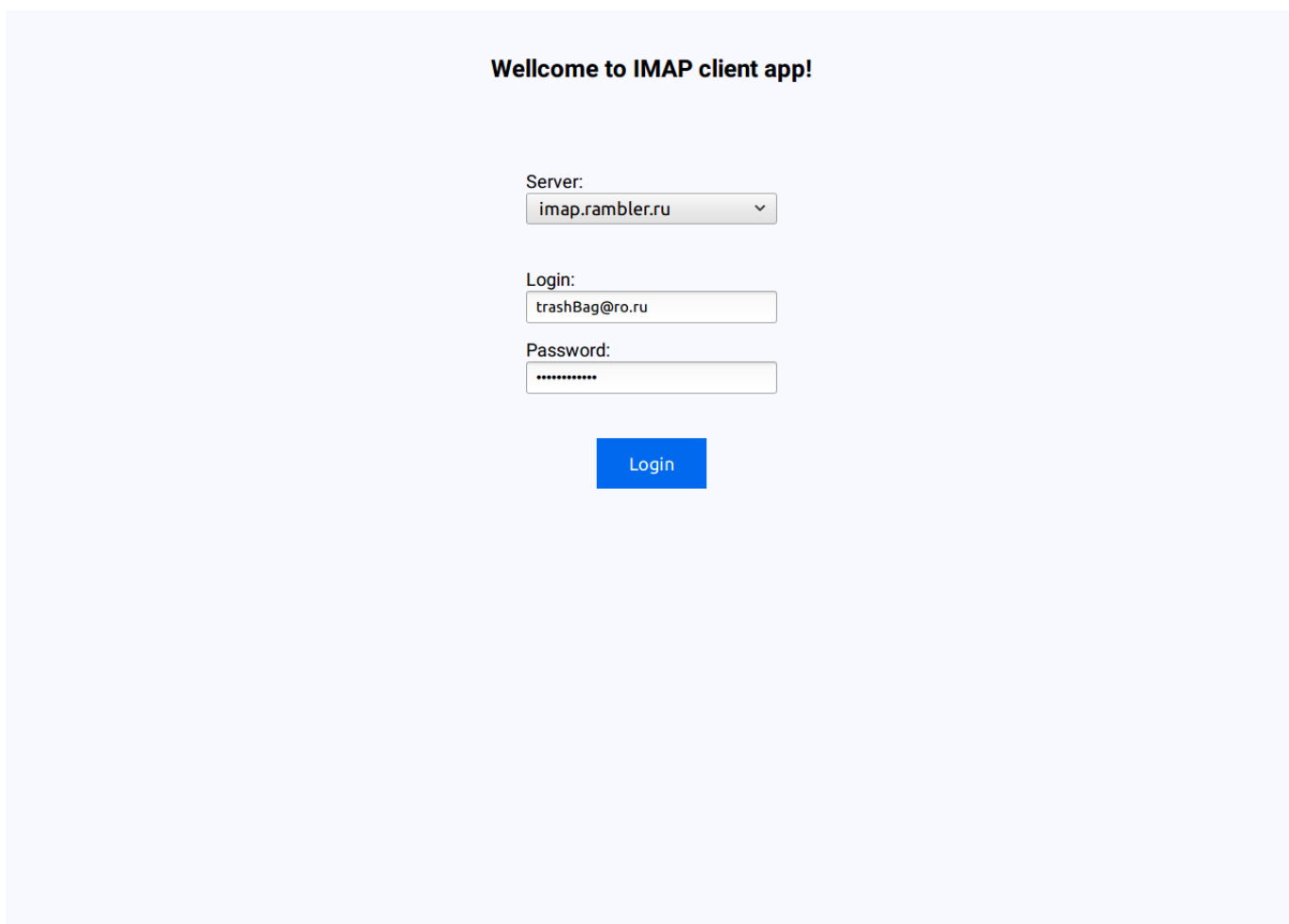
- **login** - передает IMAP-серверу аутентификационные данные пользователя. Используется при аутентификации пользователя в веб приложении, с целью сообщить ему в случае некорректности введенных данных.
- **examine** - команда, позволяющая выбрать почтовый ящик, с доступом только на чтение.
- **select** - команда, аналогичная **examine**, за исключением того, что ящик открывается и на чтение, и на запись. При использовании этой команды просмотренные письма автоматически помечаются флагом **SEEN**.
- **fetch** - команда, с помощью которой из почтового ящика извлекается необходимая информация. Используется, с целью получить список, находящихся в ящике, писем с краткой информацией о них. Также используется при получении полного текста письма.
- **search** - позволяет проводить операцию поиск. Возвращает номера писем, соответствующих условию поиска. Используется для обнаружения непочитанных писем и для поиска письма с заданным **UID**.

- list - команда, выводящая список папок, расположенных на IMAP-сервере и соответствующих условиям поиска. Используется для получения всех папок сервера.
- logout - завершает обмен данными между IMAP-клиентом и IMAP-сервером.

С целью избежать зависания программы, в начале сессии обмена данными устанавливается таймер, который через две секунды обрывает соединение.

2.3 Графический интерфейс приложения

На рисунках 2.2, 2.3, 2.4 представлен графический интерфейс приложения.



Wellcome to IMAP client app!

Server:
imap.rambler.ru

Login:
trashBag@ro.ru

Password:

Login

Рис. 2.2: Графический интерфейс приложения. Окно логина.

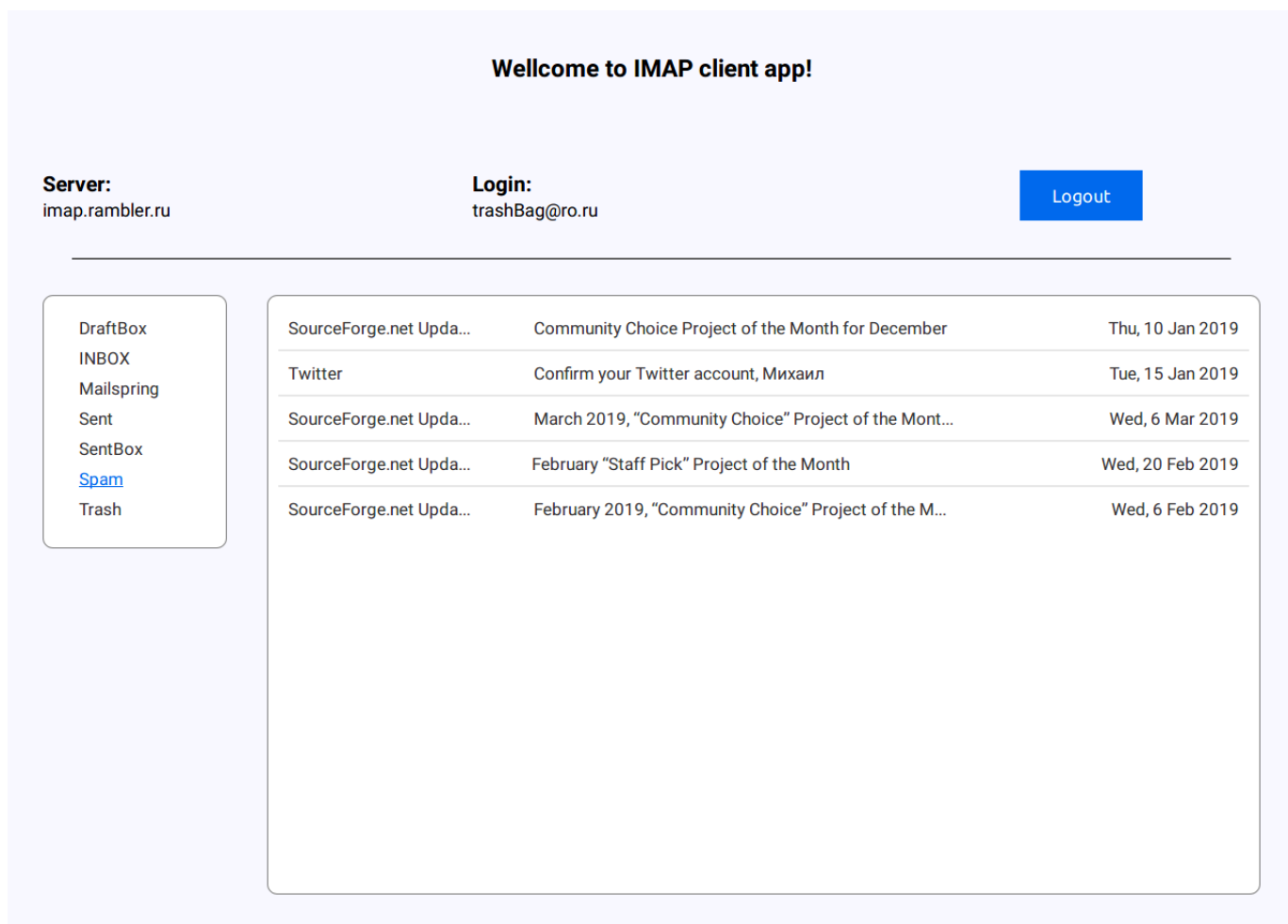


Рис. 2.3: Графический интерфейс приложения. Открыта папка Spam.

Date: Sun, 17 Mar 2019 21:02:50 +0300
 Subject: Test subject
 From: Михаил Мальцев<mikle9997@mail.ru>
 To: trashBag@ro.ru
 Text: <div dir='auto'>Test text<div dir="auto">Second line</div><div dir="auto">
</div><div dir="auto">Yolo</div></div>

Рис. 2.4: Графический интерфейс приложения. Отображение письма.

Разработанный интерфейс отображает используемую, на данный момент, почтовую папку и выделяет неп прочитанные сообщения. Окно логина уведомляет в случае, если пользователь забыл ввести данные в одну из форм или он ввёл некорректные данные. При открытие письмо выглядит максимально минималистично. На отдельные строчки вынесены такие поля, как дата, тема письма, от кого оно, кому предназначалось и собственно сам текст письма.

2.4 Тестирование

Для тестирования был выбран действующий почтовый сервер - `imap.rambler.ru`. Приложение подключалось к серверу `imap.rambler.ru` по порту 993. С сервера извлекались почтовые данные, которые успешно отображались приложением. Также был проведен эксперимент с посылкой нового письма на сервер, с помощью стороннего сервиса, и отображением его в разработанном приложении. Все проверки были успешно пройдены.

3 Вывод

В результате выполнения работы был создан ИМАР-клиент. В процессе создания приложения был подробно изучен протокол ИМАР и были закреплены знания, принципов функционирования почтовых сервисов. Также, в связи с тем, что данные приходящие от почтового сервиса требовалось в понятном виде продемонстрировать пользователю, были улучшены навыки работы с регулярными выражениями. Разработанное приложение было протестировано в работе с реальным ИМАР-сервером, тестирование подтвердило работоспособность.

4 Листинги

```
1 package main
2
3 import (
4     "fmt"
5     "html/template"
6     "net/http"
7     "regexp"
8
9     db "./db"
10    imap "./imap"
11    . "github.com/logrusorgru/aurora"
12 )
13
14 const (
15     port = ":3000"
16 )
17
18 type Data struct {
19     Auth      db.AuthData
20     Envelope [] imap.Envelope
21     Boxes     imap.Boxes
22 }
23
24 var funcMap = template.FuncMap{
25     "trunc": func(c int, s string) string {
26         runes := []rune(s)
27         if len(runes) <= c {
28             return s
29         }
30         return string(runes[:c]) + "..."
31     },
32     "dec": func(i int) int {
33         return i - 1
34     },
35 }
36
37 func isLocalhost(remoteAddr string) bool {
38     var localhost = regexp.MustCompile(`127.0.0.1:`)
39     return localhost.MatchString(remoteAddr)
40     // return true
41 }
42
43 // Index(roote) route handler
44 // read auth data from auth.json and show login/logout info
45 // also view result of request to imap server
46 func indexHandler(res http.ResponseWriter, req *http.Request) {
47     if !isLocalhost(req.RemoteAddr) {
48         http.Redirect(res, req, "https://hyperborea-theatre.ru/",
49             http.StatusSeeOther)
50         return
51     }
52     fmt.Println(req.Method, req.URL)
53
54     templates := template.Must(template.New("main").Funcs(funcMap).ParseGlob("./templates/*"))
55
56     authData := db.GetAuthData()
```

```

56
57     box := req.FormValue("box")
58     var data Data
59     if authData.Login != "" && authData.Password != "" && authData.Server != ""
60     {
61         envelopeData, listOfBoxes := imap.GetListOfMails(box)
62
63         data.Auth = authData
64         data.Envelope = envelopeData
65         data.Boxes = listOfBoxes
66     }
67     templates.ExecuteTemplate(res, "index", data)
68 }
69 func letterHandler(res http.ResponseWriter, req *http.Request) {
70     if !isLocalhost(req.RemoteAddr) {
71         http.Redirect(res, req, "https://hyperborea-theatre.ru/",
72             http.StatusSeeOther)
73         return
74     }
75     uid := req.FormValue("uid")
76     box := req.FormValue("box")
77     fmt.Println(req.Method, req.URL)
78
79     templates := template.Must(template.New("main").Funcs(funcMap).ParseGlob("./
80     templates/*"))
81
82     letter := imap.GetLetter(uid, box)
83
84     templates.ExecuteTemplate(res, "letter", letter)
85 }
86 // Auth route hanlder
87 // write data from Post request to auth.json
88 // and send redirect to index(root)
89 func authHandler(res http.ResponseWriter, req *http.Request) {
90     if !isLocalhost(req.RemoteAddr) {
91         http.Redirect(res, req, "https://hyperborea-theatre.ru/",
92             http.StatusSeeOther)
93         return
94     }
95     fmt.Println(req.Method, req.URL)
96     err := req.ParseForm()
97     if err != nil {
98         panic(err)
99     }
100     data := db.AuthData{
101         Login:    req.FormValue("login"),
102         Password: req.FormValue("password"),
103         Server:   req.FormValue("server")}
104
105     db.SaveAuthData(data)
106
107     if req.Method == "POST" && data.Login != "" && data.Password != "" &&
108     data.Server != "" {
109         fmt.Fprintf(res, "%t", imap.TryToLogin())
110     } else {
111         http.Redirect(res, req, "/", http.StatusSeeOther)
112     }
113 }

```



```

111
112 func faviconHandler(res http.ResponseWriter, req *http.Request) {
113     fmt.Fprint(res, "lol")
114 }
115
116 func main() {
117     http.Handle("/public/", http.StripPrefix("/public/", http.FileServer(
118         http.Dir("./public/"))))
119     http.HandleFunc("/favicon.ico", faviconHandler)
120     http.HandleFunc("/", indexHandler)
121     http.HandleFunc("/auth", authHandler)
122     http.HandleFunc("/letter", letterHandler)
123
124     fmt.Println("Listening on port", Brown(port))
125     err := http.ListenAndServe(port, nil)
126     fmt.Println("Error creating http server:", Red(err))
127 }

```

```

1 package db
2
3 import (
4     "encoding/json"
5     "os"
6 )
7
8 const (
9     fileName = "auth.json"
10 )
11
12 // AuthData - structure which describe authentication data which user entered
13 // Login : login of user
14 // Password : password of user
15 // Server : IMAP server which was chosen by user
16 type AuthData struct {
17     Login    string `json:"login"`
18     Password string `json:"password"`
19     Server   string `json:"server"`
20 }
21
22 // GetAuthData open file 'auth.json' if exist
23 // and return data from it
24 // if not exist create it and put void auth data
25 func GetAuthData() AuthData {
26     fi, err := os.Open(fileName)
27     if err != nil {
28         createFileWithVoidAuth()
29         ad := AuthData{}
30         return ad
31     }
32     defer func() {
33         if err := fi.Close(); err != nil {
34             panic(err)
35         }
36     }()
37
38     var data AuthData
39     jsonParser := json.NewDecoder(fi)
40
41     if err = jsonParser.Decode(&data); err != nil {
42         panic(err.Error())
43     }
44 }

```

```

43     }
44     return data
45 }
46
47 func createFileWithVoidAuth() {
48     voidAuth := map[string]string{
49         "login":    "",
50         "password": "",
51         "server":    ""}
52
53     fo, err := os.Create("auth.json")
54     if err != nil {
55         panic(err)
56     }
57     defer func() {
58         if err := fo.Close(); err != nil {
59             panic(err)
60         }
61     }()
62
63     enc := json.NewEncoder(fo)
64     enc.Encode(voidAuth)
65 }
66
67 // SaveAuthData – save authentication data to auth.json file
68 // data : input value
69 func SaveAuthData(data AuthData) {
70     fo, err := os.Create("auth.json")
71     if err != nil {
72         panic(err)
73     }
74     defer func() {
75         if err := fo.Close(); err != nil {
76             panic(err)
77         }
78     }()
79
80     enc := json.NewEncoder(fo)
81     enc.Encode(data)
82 }

```

```

1 package imap
2
3 import (
4     "crypto/tls"
5     "fmt"
6     "time"
7
8     db " ../db"
9     . "github.com/logrusorg/aurora"
10 )
11
12 const (
13     port = ":993"
14 )
15
16 // Envelope – struct for data from envelope of mails
17 type Envelope struct {
18     Date      string
19     Subject   string

```

```

20     Sender    string
21     Email     string
22     Seen      bool
23     UID       int
24 }
25
26 // Letter – struct which describe content of email
27 type Letter struct {
28     Date,
29     Subject,
30     From,
31     To,
32     Body string
33 }
34
35 // Boxes – name of boxes and one selected
36 type Boxes struct {
37     BoxesNames []string
38     SelectedBox int
39 }
40
41 // TryToLogin – return result of login
42 func TryToLogin() (successful bool) {
43     authData := db.GetAuthData()
44     conn := createConn(authData.Server)
45     defer func() {
46         conn.Close()
47         fmt.Println("Connection successful close")
48     }()
49     conn.SetReadDeadline(time.Now().Add(2 * time.Second))
50
51     successful = false
52     str := login(conn, authData.Login, authData.Password)
53     if len(str) > 0 {
54         successful = isOK(str[len(str)-1])
55     } else {
56         fmt.Println("Error->", Red("NO completed"))
57     }
58     logout(conn)
59     return
60 }
61
62 // GetListOfMails – return array of string
63 func GetListOfMails(selectedBox string) (envelopes []Envelope, listofBoxes Boxes) {
64     authData := db.GetAuthData()
65     conn := createConn(authData.Server)
66     defer func() {
67         conn.Close()
68         fmt.Println("Connection successful close")
69     }()
70     conn.SetReadDeadline(time.Now().Add(2 * time.Second))
71
72     login(conn, authData.Login, authData.Password)
73     examineBox(conn, selectedBox)
74     responseFetch := fetchHeader(conn)
75     responseSearch := searchUnseen(conn)
76     if len(responseFetch) > 0 && len(responseSearch) > 0 {
77         messages := mergeStringsToMessages(responseFetch)
78         unseen := parseSearch(responseSearch)

```

```

79     envelopes = extractUsefulData(messages, unseen)
80 } else {
81     fmt.Println("Error->", Red("NO completed"))
82 }
83 listOfBoxes.BoxesNames = parseBoxList(getListOfBoxes(conn))
84 for i, name := range listOfBoxes.BoxesNames {
85     if name == selectedBox {
86         listOfBoxes.SelectedBox = i
87     }
88 }
89
90 logout(conn)
91 return
92 }
93
94 // GetLetter - find by UID letter and return it
95 func GetLetter(uid string, selectedBox string) (letter Letter) {
96     authData := db.GetAuthData()
97     conn := createConn(authData.Server)
98     defer func() {
99         conn.Close()
100         fmt.Println("Connection successful close")
101     }()
102     conn.SetReadDeadline(time.Now().Add(2 * time.Second))
103
104     login(conn, authData.Login, authData.Password)
105     selectInbox(conn, selectedBox)
106
107     letterNum := parseSearch(findLetter(conn, uid))[0]
108     letterText := fetchLetter(conn, letterNum)
109
110     letter.Date, letter.Subject, letter.From, letter.To, letter.Body =
111     parseLetter(letterText)
112
113     logout(conn)
114     return
115 }
116
117 // Set up tls connection
118 func createConn(server string) *tls.Conn {
119     conn, err := tls.Dial("tcp", server+port, &tls.Config{})
120     if err != nil {
121         fmt.Println(Red("Connection failed"))
122         panic(err)
123     }
124     fmt.Println("Connection successful open")
125     return conn
126 }

```

```

1 package imap
2
3 import (
4     "crypto/tls"
5     "fmt"
6     "strconv"
7
8     . "github.com/logrusorgru/aurora"
9 )
10
11 func login(conn *tls.Conn, login string, pass string) []string {

```

```

12     prefix := "a0001"
13     fmt.Println("Client->", Green(prefix+" login "+login+" *****"))
14     fmt.Fprintf(conn, prefix+" login "+login+" "+pass+"\n")
15
16     return readBeforePrefixLine(conn, prefix)
17 }
18
19 func examineBox(conn *tls.Conn, selectedBox string) []string {
20     prefix := "a0002"
21     fmt.Println("Client->", Green(prefix+" examine "+selectedBox))
22     fmt.Fprintf(conn, "%s examine %s\n", prefix, selectedBox)
23
24     return readBeforePrefixLine(conn, prefix)
25 }
26
27 func selectInbox(conn *tls.Conn, selectedBox string) []string {
28     prefix := "a0003"
29     fmt.Println("Client->", Green(prefix+" select "+selectedBox))
30     fmt.Fprintf(conn, "%s select %s\n", prefix, selectedBox)
31
32     return readBeforePrefixLine(conn, prefix)
33 }
34
35 func fetchHeader(conn *tls.Conn) []string {
36     prefix := "a0004"
37     fmt.Println("Client->", Green(prefix+" fetch 1:* (ENVELOPE UID) "))
38     fmt.Fprintf(conn, prefix+" fetch 1:* (ENVELOPE UID) \n")
39
40     return readBeforePrefixLine(conn, prefix)
41 }
42
43 func searchUnseen(conn *tls.Conn) []string {
44     prefix := "a0005"
45     fmt.Println("Client->", Green(prefix+" search unseen"))
46     fmt.Fprintf(conn, prefix+" search unseen\n")
47
48     return readBeforePrefixLine(conn, prefix)
49 }
50
51 func findLetter(conn *tls.Conn, uid string) []string {
52     prefix := "a0006"
53     fmt.Println("Client->", Green(prefix+" search UID "+uid))
54     fmt.Fprintf(conn, "%s search UID %s\n", prefix, uid)
55
56     return readBeforePrefixLine(conn, prefix)
57 }
58
59 func fetchLetter(conn *tls.Conn, num int) []string {
60     prefix := "a0007"
61     fmt.Println("Client->", Green(prefix+" fetch "+strconv.Itoa(num)+" (body []) "
62     ))
63     fmt.Fprintf(conn, "%s fetch %d (body []) \n", prefix, num)
64
65     return readBeforePrefixLine(conn, prefix)
66 }
67
68 func getListOfBoxes(conn *tls.Conn) []string {
69     prefix := "a0008"
70     fmt.Println("Client->", Green(prefix+" LIST "" "" '"))
71     fmt.Fprintf(conn, "%s LIST \"\" \"\"%\" \n", prefix)

```

```

71
72     return readBeforPrefixLine(conn, prefix)
73 }
74
75 func logout(conn *tls.Conn) []string {
76     prefix := "a00010"
77     fmt.Println("Client->", Green(prefix+" logout"))
78     fmt.Fprintf(conn, prefix+" logout\n")
79
80     return readBeforPrefixLine(conn, prefix)
81 }

```

```

1 package imap
2
3 import (
4     "bytes"
5     "encoding/base64"
6     "fmt"
7     "io/ioutil"
8     "mime"
9     "mime/quotedprintable"
10    "net/mail"
11    "regexp"
12    "strconv"
13    "strings"
14
15    . "github.com/logrusorg/gru/aurora"
16 )
17
18 func isOK(response string) (successful bool) {
19     var valid = regexp.MustCompile(`^.{5} OK `)
20     successful = valid.MatchString(response)
21     return
22 }
23
24 // convert strings to messages
25 // some messages can contain from many strings
26 func mergeStringsToMessages(response []string) (result []string) {
27     var valid = regexp.MustCompile(`^\s* ([0-9]+) FETCH `)
28     var lenght = len(response)
29     for i, line := range response {
30         if valid.MatchString(line) {
31             result = append(result, line)
32         } else {
33             if i != lenght-1 {
34                 result[len(result)-1] += line
35             }
36         }
37     }
38     return
39 }
40
41 func extractUsefulData(response []string, unseen []int) (result []Envelope) {
42     var dateRegexp = regexp.MustCompile(`[a-zA-Z]{3}, ?(\d+) [a-zA-Z]{3} \d{4}`)
43     var subjectRegexp = regexp.MustCompile(`" "(.)*" \(\('`
44     var contactRegexp = regexp.MustCompile(`" \(\(("[w!&=/, (\-)(\|)(\.) (\?)`
45     var uidRegexp = regexp.MustCompile(`UID \d+`)

```

```

46     for i, line := range response {
47         date := dateRegexp.FindString(line)
48
49         subject := ""
50         matchesSubject := subjectRegexp.FindString(line)
51         a1 := []rune(matchesSubject)
52         if len(matchesSubject) > 7 {
53             subject = utf8Decoder(string(a1[3 : len(matchesSubject)-4]))
54         }
55
56         matchesContact := contactRegexp.FindString(line)
57         sender := ""
58         email := ""
59         a2 := []rune(matchesContact)
60         if len(matchesContact) > 6 {
61             sender, email = contactParser(string(a2[4 : len(matchesContact)-2]))
62         }
63
64         uid, err := strconv.Atoi(strings.Split(uidRegexp.FindString(line), " ")[1])
65         if err != nil {
66             fmt.Println("UID Atoi :", Red(err))
67         }
68
69         envelope := Envelope{date, subject, sender, email, !contains(unseen, i
+1), uid}
70         result = append(result, envelope)
71     }
72     return
73 }
74
75 func contactParser(contact string) (sender, email string) {
76     parts := strings.Split(contact, "\"")
77     var codeNil = regexp.MustCompile('NIL')
78     var partOfContact []string
79     for _, part := range parts {
80         if !codeNil.MatchString(part) && part != "" && part != " " {
81             partOfContact = append(partOfContact, utf8Decoder(part))
82         }
83     }
84     switch len(partOfContact) {
85     case 1:
86         {
87             sender = fmt.Sprintf("%s", partOfContact[0])
88         }
89     case 2:
90         {
91             sender = fmt.Sprintf("%s", partOfContact[0])
92             email = fmt.Sprintf("<%s@%s>", partOfContact[0], partOfContact[1])
93         }
94     case 3:
95         {
96             sender = fmt.Sprintf("%s", partOfContact[0])
97             email = fmt.Sprintf("<%s@%s>", partOfContact[1], partOfContact[2])
98         }
99     case 4:
100        {
101            sender = fmt.Sprintf("%s %s", partOfContact[0], partOfContact[1])
102            email = fmt.Sprintf("<%s@%s>", partOfContact[2], partOfContact[3])
103        }

```

```

104     }
105     return
106 }
107
108 func utf8Decoder(text string) (result string) {
109     var codeUTF8 = regexp.MustCompile(`=\?(utf-8|UTF-8)\?[\\w!&=/, (\-)(\|)(\.)
110     (\?)(\+):#@]*\?=`)
111     dec := new(mime.WordDecoder)
112     if codeUTF8.MatchString(text) {
113         var b strings.Builder
114         parts := strings.Split(text, " ")
115         for _, part := range parts {
116             if part != "" && part != " " {
117                 if codeUTF8.MatchString(part) {
118                     decodedText, _ := dec.Decode(part)
119                     b.WriteString(decodedText)
120                 } else {
121                     b.WriteString(part)
122                 }
123             }
124         }
125         result = b.String()
126     } else {
127         result = text
128     }
129     return
130 }
131
132 func parseSearch(response []string) (result []int) {
133     if len(response) == 2 {
134         var num = regexp.MustCompile(`\d+`)
135         numArr := num.FindAllString(response[0], -1)
136         for _, part := range numArr {
137             if newNum, err := strconv.Atoi(part); err == nil {
138                 result = append(result, newNum)
139             }
140         }
141     }
142     return
143 }
144
145 func parseLetter(letter []string) (date, subject, from, to, body string) {
146     var b strings.Builder
147     numOfStr := len(letter)
148     for i, line := range letter {
149         if i > 0 && i < numOfStr-2 {
150             b.WriteString(line)
151         }
152     }
153     strLetter := b.String()
154
155     msg, err := mail.ReadMessage(bytes.NewBuffer([]byte(strLetter)))
156     if err != nil {
157         panic(err)
158     }
159
160     date = msg.Header.Get("Date")
161     subject = utf8Decoder(msg.Header.Get("Subject"))
162     from = utf8Decoder(msg.Header.Get("From"))
163     to = utf8Decoder(msg.Header.Get("To"))

```



```

163
164     encoding := msg.Header.Get("Content-Transfer-Encoding")
165
166     if encoding == "base64" {
167         buf := new(bytes.Buffer)
168         buf.ReadFrom(msg.Body)
169         s := buf.String()
170         decodedB64, err := base64.StdEncoding.DecodeString(s)
171         if err != nil {
172             fmt.Println("decodeB64 error:", err)
173             return
174         }
175         body = string(decodedB64)
176     } else {
177         r := quotedprintable.NewReader(msg.Body)
178         decodedQI, err := ioutil.ReadAll(r)
179         if err != nil {
180             fmt.Println("decodeQI error:", err)
181             return
182         }
183         body = string(decodedQI)
184     }
185
186     return
187 }
188
189 func parseBoxList(response []string) (nameOfBoxes []string) {
190     var b strings.Builder
191     for _, line := range response {
192         b.WriteString(line)
193     }
194     strResponse := b.String()
195
196     var boxNameRegexp = regexp.MustCompile(`/"/ [\w]+`)
197     boxNames := boxNameRegexp.FindAllString(strResponse, -1)
198
199     for _, nameInText := range boxNames {
200         nameOfBoxes = append(nameOfBoxes, nameInText[4:])
201     }
202
203     return
204 }

```

```

1 package imap
2
3 import (
4     "bufio"
5     "crypto/tls"
6     "fmt"
7     "io"
8     "strings"
9
10    . "github.com/logrusorgru/aurora"
11 )
12
13 // readStrings - read the specified number of lines
14 // conn : reading stream
15 // num : number of lines which should be reading
16 // text : result of reading
17 func readStrings(conn io.Reader, num int) (text []string) {

```

```

18     reader := bufio.NewReader(conn)
19     for index := 0; index < num; index++ {
20         line, err := reader.ReadString('\n')
21         if err != nil {
22             fmt.Println("Error->", Red(err))
23             break
24         }
25         text = append(text, line)
26         fmt.Printf("Server-> %s", Cyan(line))
27     }
28     return
29 }
30
31 // 2.3.4. [RFC-2822] Size Message Attribute
32 // The number of octets in the message, as expressed in [RFC-2822]
33 // format.
34 func readBytes() {
35
36 }
37
38 func readBeforePrefixLine(conn *tls.Conn, prefix string) (response []string) {
39     reader := bufio.NewReader(conn)
40     for {
41         line, err := reader.ReadString('\n')
42         if err != nil {
43             fmt.Println("Error->", Red(err))
44             break
45         }
46         response = append(response, line)
47         fmt.Printf("Server-> %s", Cyan(line))
48         if strings.HasPrefix(line, prefix) {
49             break
50         }
51     }
52     return
53 }

```

```

1 package imap
2
3 func contains(s []int, e int) bool {
4     for _, a := range s {
5         if a == e {
6             return true
7         }
8     }
9     return false
10 }

```

```

1 {{ define "content" }}
2
3     <div class="content">
4         {{ $selected := .Boxes.SelectedBox }}
5         {{ $ArrayOfBoxes := .Boxes.BoxesNames }}
6
7         <div class="left-block">
8             <div class="box-list">
9                 {{ range $key, $value := $ArrayOfBoxes }}
10                 <a href="/?box={{ $value }}" class="box-item {{ if eq $key
$selected }}selected{{ end }}">
11                     {{ $value }}

```

```

12         </a>
13     {{ end }}
14 </div>
15 </div>
16
17 <div class="message-block">
18     {{ range $key, $value := .Envelope }}
19         {{ if not (eq $key 0) }} <hr noshade> {{ end }}
20         <a href="/letter?box={{ index $ArrayOfBoxes $selected }}&uid={{
21 $value.UID }}" class="message {{ if not .Seen }}unseen{{ end }}">
22             <div class="contact" title="{{ .Sender }}" {{.Email }}">
23                 <span>{{ trunc 20 $value.Sender }}</span> <span class="
24 funny-ball">&#9679;</span>
25             </div>
26             <div class="subject" title="{{ .Subject }}">
27                 {{ trunc 50 $value.Subject }}
28             </div>
29             <div class="date">
30                 {{ .Date }}
31             </div>
32         </a>
33     {{ end }}
34 </div>
35
36 {{ end }}

```

```

1 {{ define "footer" }}
2
3     <script src="/public/main.js"></script>
4
5 </body>
6 </html>
7
8 {{ end }}

```

```

1 {{ define "header" }}
2
3 <!DOCTYPE html>
4 <html>
5 <head>
6     <meta charset="utf-8">
7     <meta http-equiv="X-UA-Compatible" content="IE=edge">
8     <title>IMAP client</title>
9     <meta name="viewport" content="width=device-width, initial-scale=1">
10    <link rel="stylesheet" type="text/css" media="screen" href="/public/main.css
11    ">
12 </head>
13 <body>
14 {{ end }}

```

```

1 {{ define "index" }}
2 {{ template "header" }}
3
4     <div class="main">
5         <div class="title">
6             <h2>Wellcome to IMAP client app!</h2>
7         </div>

```

```

8
9      {{ if and .Auth.Login .Auth.Password .Auth.Server }}
10
11      <div class="config-data">
12          <div class="choose-server">
13              <h3>Server:</h3>
14              <p>{{ .Auth.Server }}</p>
15          </div>
16
17          <div class="login-block" id="login-block">
18              <h3>Login:</h3>
19              <p>{{ .Auth.Login }}</p>
20          </div>
21
22          <div class="login-button-block">
23              <input type="submit" class="blue-button" value="Logout" id="
logout-button" />
24          </div>
25      </div>
26
27      <hr/>
28
29      <div class="output">
30          {{ template "content" . }}
31      </div>
32
33      {{ else }}
34
35      <form class="config-form" action="/auth">
36          <div class="choose-server">
37              <label for="server">Server:</label>
38              <select name="server" id="server">
39                  <option>imap.rambler.ru</option>
40              </select>
41          </div>
42
43          <div class="login-block" id="login-block">
44              <label for="login" id="label-login">Login:</label>
45              <input type="text" id="login" placeholder="login" name="
login" autocomplete="off" />
46
47              <label for="password" id="label-password">Password:</label>
48              <input type="password" id="password" placeholder="*****"
name="password" autocomplete="off" />
49          </div>
50
51          <div class="login-button-block">
52              <input type="submit" class="blue-button" value="Login" id="
login-button" />
53          </div>
54      </form>
55
56      {{ end }}
57
58  </div>
59
60  {{ template "footer" }}
61  {{ end }}

```

```

1  {{ define "letter" }}

```

```

2  {{ template "header" }}
3
4      <div class="letter">
5          <div class="date">
6              Date: {{ .Date }}
7          </div>
8          <div class="subject">
9              Subject: {{ .Subject }}
10         </div>
11         <div class="from">
12             From: {{ .From }}
13         </div>
14         <div class="to">
15             To: {{ .To }}
16         </div>
17         <div class="body-text">
18             Text: {{ .Body }}
19         </div>
20     </div>
21
22 {{ template "footer" }}
23 {{ end }}

```

```

1  * {
2      margin: 0;
3      padding: 0;
4      box-sizing: border-box;
5  }
6
7  body {
8      font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "
9      Oxygen",
10      "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
11      sans-serif;
12      -webkit-font-smoothing: antialiased;
13      -moz-osx-font-smoothing: grayscale;
14  }
15  input:active, input:focus {
16      outline: none;
17  }
18
19  button::-moz-focus-inner {
20      border: 0;
21  }
22
23  .main {
24      min-height: 100vh;
25      display: flex;
26      flex-direction: column;
27      align-items: center;
28      justify-content: center;
29      background-color: ghostwhite;
30  }
31
32  .title {
33      display: flex;
34      flex-direction: column;
35      justify-content: center;
36      flex: 0;

```

```

37     margin-top: 40px;
38     margin-bottom: 50px;
39 }
40
41 .config-form {
42     flex: 1;
43     font-size: 18px;
44 }
45
46 .config-data {
47     display: flex;
48     flex-direction: row;
49     justify-content: space-between;
50     width: 100%;
51     flex: 0;
52     flex-wrap: wrap;
53     font-size: 18px;
54 }
55
56 .choose-server {
57     display: flex;
58     flex-direction: column;
59     justify-content: center;
60     align-items: left;
61     margin: 35px;
62     flex: 1;
63 }
64 .choose-server select {
65     font-size: 18px;
66 }
67
68 .login-block label {
69     margin-top: 7px;
70 }
71
72 .login-block input {
73     margin-bottom: 7px;
74     padding: 2px;
75 }
76
77 .login-block {
78     display: flex;
79     flex-direction: column;
80     justify-content: center;
81     margin: 35px;
82     flex: 1;
83 }
84
85 .login-button-block {
86     display: flex;
87     flex-direction: column;
88     justify-content: center;
89     align-items: center;
90     margin: 35px;
91     flex: 1;
92 }
93
94 hr {
95     min-width: 90%;
96 }

```

```

97
98 /* Buttons styles start */
99 input.blue-button {
100     display: inline-block;
101     border: none;
102     padding: 1rem 2rem;
103     margin: 2px;
104     text-decoration: none;
105     background: #0069ed;
106     color: #ffffff;
107     font-size: calc(2px + 1rem);
108     line-height: 1;
109     cursor: pointer;
110     text-align: center;
111     transition: background 250ms ease-in-out, transform 150ms ease;
112     -webkit-appearance: none;
113     -moz-appearance: none;
114 }
115 input.blue-button:hover,
116 input.blue-button:focus {
117     background: #0053ba;
118 }
119 input.blue-button:active {
120     transform: scale(0.95);
121 }
122 /* Button styles end */
123
124 input.error {
125     background-color: #fce4e4;
126     border-color: #cc0033;
127     border-style: solid;
128     outline: none;
129 }
130 label.error {
131     color: #cc0033;
132 }
133
134 .output {
135     flex: 1;
136     width: 100%;
137     display: flex;
138     flex-direction: column;
139     justify-content: start;
140 }
141
142 .content {
143     flex: 2;
144     display: flex;
145     flex-direction: row;
146     align-items: center;
147     justify-content: center;
148     margin: 35px;
149     font-size: 17px;
150 }
151
152 .left-block {
153     flex: 1;
154     height: 100%;
155     margin-right: 20px;
156 }

```

```

157
158 .box-list {
159     display: flex;
160     justify-content: start;
161     align-items: flex-start;
162     flex-direction: column;
163     background-color: #FFFFFF;
164     border-radius: 10px;
165     border: 1px solid #757575;
166     padding: 20px;
167     margin-right: 10px;
168 }
169
170 .box-item {
171     margin-bottom: 7px;
172     margin-left: 15px;
173     color: #333032;
174     text-decoration: none;
175 }
176
177 .box-item.selected {
178     color: #0069ed;
179     text-decoration-line: underline;
180 }
181
182 .box-item:hover {
183     color: #0069ed;
184 }
185
186 .message-block {
187     flex: 5;
188     height: 100%;
189     display: flex;
190     flex-direction: column-reverse;
191     justify-content: start;
192     background-color: #FFFFFF;
193     border-radius: 10px;
194     border: 1px solid #757575;
195     padding: 10px;
196     margin-left: 10px;
197 }
198
199 .message {
200     color: #333032;
201     text-decoration: none;
202     display: flex;
203     flex-direction: row;
204     justify-content: space-between;
205     padding: 10px;
206     flex-wrap: wrap;
207 }
208
209 .message:hover {
210     background-color: #fdf25c65;
211 }
212
213 .message:hover .funny-ball {
214     display: block;
215 }
216

```



```

217 .unseen {
218     background-color: #a3a8a82a;
219 }
220
221 .unseen .funny-ball{
222     display: block;
223     color: #0069ed;
224 }
225
226 .contact {
227     flex: 2;
228     display: flex;
229     flex-direction: row;
230     justify-content: space-between;
231     align-items: center;
232     margin-right: 15px;
233 }
234 .funny-ball {
235     display: none;
236     color: #cfcfcf;
237 }
238 .unseen .contact {
239     font-weight: bold;
240 }
241 .subject {
242     flex: 5;
243     display: flex;
244     flex-direction: column;
245     justify-content: start;
246     align-items: left;
247 }
248 .unseen .subject {
249     font-weight: bold;
250 }
251 .date {
252     display: flex;
253     flex-direction: column;
254     align-items: right;
255 }
256 .message-block hr {
257     color: #e7e7e7;
258 }

```

```

1
2 function removeErrors() {
3     document.getElementById("login").classList.remove('error')
4     document.getElementById("label-login").classList.remove('error')
5     document.getElementById("password").classList.remove('error')
6     document.getElementById("label-password").classList.remove('error')
7 }
8
9 document.getElementById('login-block').addEventListener("click", event => {
10     removeErrors();
11 });
12
13 if (document.getElementById('logout-button')) {
14     document.getElementById('logout-button').addEventListener("click", event =>
15     {
16         event.preventDefault();
17         window.location.replace("/auth");
18     }
19 );

```

```

17     });
18 }
19
20 if (document.getElementById('login-button')) {
21     document.getElementById('login-button').addEventListener("click", event => {
22         event.preventDefault();
23         const server = document.getElementById("server").value,
24             login = document.getElementById("login").value,
25             password = document.getElementById("password").value;
26
27         if (!server || !login || !password) {
28             if (!login) {
29                 document.getElementById("login").classList.add('error')
30                 document.getElementById("label-login").classList.add('error')
31             }
32             if (!password) {
33                 document.getElementById("password").classList.add('error')
34                 document.getElementById("label-password").classList.add('error')
35             }
36             return;
37         }
38
39         var xhr = new XMLHttpRequest();
40         xhr.onreadystatechange = () => {
41             if (xhr.readyState === XMLHttpRequest.DONE) {
42                 if (xhr.responseText === "true") {
43                     window.location.reload();
44                 } else {
45                     document.getElementById("login").classList.add('error')
46                     document.getElementById("label-login").classList.add('error')
47
48                     document.getElementById("password").classList.add('error')
49                     document.getElementById("label-password").classList.add('
50 error')
51                 }
52             }
53         }
54
55         const body = 'server=' + encodeURIComponent(server) +
56             '&login=' + encodeURIComponent(login) +
57             '&password=' + encodeURIComponent(password);
58
59         xhr.open("POST", '/auth', true);
60         xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
61         xhr.send(body);
62     });
63 }
64
65 if (document.getElementsByClassName('message')) {
66     const list = document.getElementsByClassName('message');
67     for (let item of list) {
68         item.addEventListener("click", event => {
69             item.classList.remove("unseen");
70         });
71     }
72 }

```