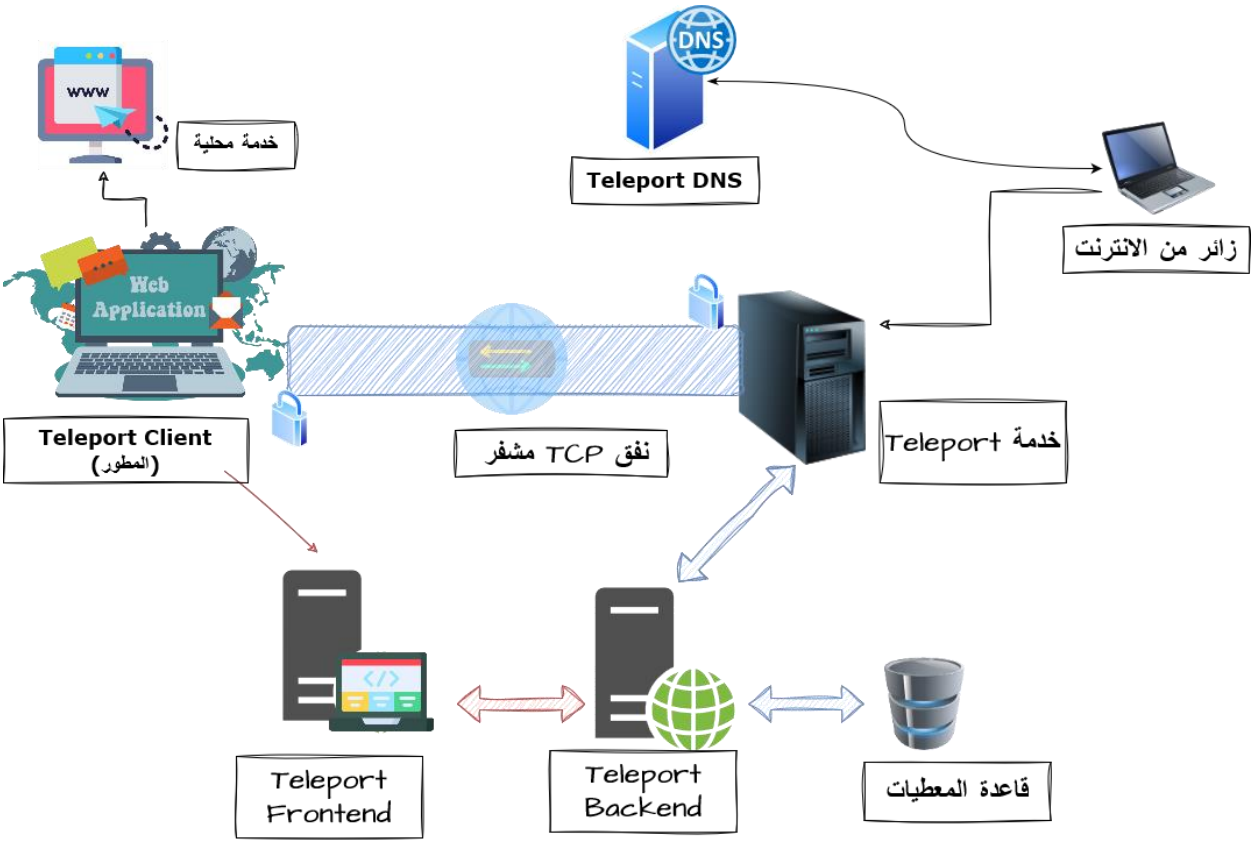


هذا الجزء ليس عنصرا ظاهرا في النظام وانما هو عبارة عن process يقوم تطبيق سطح المكتب Teleport_Desktop_application في طرف المطور (Teleport client) بتشغيلها لكنها لا تقل أهمية عن أي عنصر اخر.

هذا هو الجز العملي الخاص به مع تطبيق سطح المكتب

في التقرير 😈

الفقرة **4.7- Teleport client application**

Teleport client application

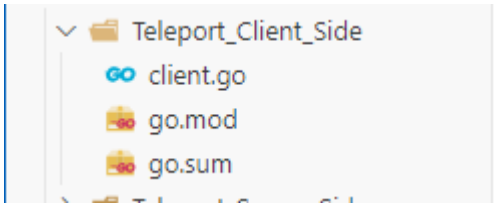
نعرض في هذا الجزء تفاصيل تنجيز تطبيق سطح المكتب الذي سيعمل في طرف الزبون (المطور)، يجب التذكر أولا انا هذا التطبيق ليس مجرد واجهة عرض للمعلومات بل هو تغليف لخدمة قائمة بحد ذاتها مهمتها التخاطب مع خدمة Teleport في طرف المخدم وتبادل معلومات معه للوصول في النهاية لفتح نفق مشفر بين Teleport clinet و Teleport service كما هو موضح في الشكل 1.6 .

تم بناء التطبيق باستخدام إطار عمل Flutter أما الخدمة بحد ذاتها تم تنجيزها باستخدام لغة Go لنتحدث أولا عن تنجيز الخدمة بلغة Go.

Teleport Client Go section -1.4.7

هنا الأمور اقل تعقيداً مما هي عليه في طرف المخدم. تقتصر مهمة هذه الخدمة على:

- ارسال اسم المستخدم وكلمة المرور والمنفذ الذي يريد المطور مشاركته.
- تطبيق خوارزمية diffie hellman بالتزامن مع طرف المخدم للوصول لمفتاح مشترك ومن ثم فتح نفق اتصال مشفر.
- توجيه حركة المرور من وإلى الخدمة المحلية.



الشكل 1 .7 Teleport client side

مجرد ملف واحد client.go مقسم وظيفيا لجزيئين:

- الجزء الأول أسميته جزء ping pong في هذا الجزء يتم تبادل الرسائل مع طرف المخدم والاتفاق على المفتاح المشترك وحصول الزبون على الرابط URL الذي يمكنه من مشاركة بيئته المحلية.
- الجز الثاني يتم من خلاله فتح النفق والتحكم بتوجيه حركة المرور الواردة من النفق الى الخدمة المحلية وبالعكس.

```
    }
    //   ************************************  Deffi hellman and authintaction
    // this part ping pong with the server 🏓🏓
    curve := elliptic.P256()
    privKey, x, y, err := elliptic.GenerateKey(curve, rand.Reader)
    fatal(err)
    pubKey := elliptic.Marshal(curve, x, y)
    clientPubKeyHex := hex.EncodeToString(pubKey)

    conn, err := net.Dial("tcp", net.JoinHostPort(*host, *port))
    fatal(err)
    client := httputil.NewClientConn(conn, bufio.NewReader(conn))
    req, err := http.NewRequest("GET", "/", nil)
    fatal(err)

    req.Header.Set("X-Username", *username)
    req.Header.Set("X-Password", *password)
    req.Header.Set("X-Client-Public-Key", clientPubKeyHex)
    req.Host = net.JoinHostPort(*host, *port)
    log.Println("Sending request with username, password, and public key")
    client.Write(req)

    resp, _ := client.Read(req)

    serverPubKeyHex := resp.Header.Get("X-Server-Public-Key")
    if serverPubKeyHex == "" {
        log.Fatal("Server public key not received")
    }
    serverPubKey, err := hex.DecodeString(serverPubKeyHex)
    fatal(err)

    serverX, serverY := elliptic.Unmarshal(curve, serverPubKey)
    if serverX == nil || serverY == nil {
        log.Fatal("Failed to unmarshal server public key")
    }
```

ping pong code section 2 .7 الشكل

```go
//   ************************************  handleConnection ********************************

func handleSecureConnection(ch, conn io.ReadWriteCloser, aesGCM cipher.AEAD) {
    defer ch.Close()
    defer conn.Close()

    buffer := make([]byte, 4096)

    //here let's handle sending encrypted data from client to server .....
    go func() {
        defer conn.Close() // ensure that the connection is closed when done
        for {
            n, err := conn.Read(buffer)
            if err != nil {
                if err != io.EOF {
                    log.Println("Connection read error:", err)
                }
                return
            }

            nonce := make([]byte, aesGCM.NonceSize())
            if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
                log.Println("Error generating nonce:", err)
                return
            }
            log.Printf("Client nonce: %x\n", nonce)

            encryptedData := aesGCM.Seal(nil, nonce, buffer[:n], nil)
            log.Printf("Client encrypted data: %x\n", encryptedData)

            if _, err := ch.Write(nonce); err != nil {
                log.Println("Error writing nonce to channel:", err)
                return
```

الشكل 7. 3 التحكم بالنفق وتوجيه حركة المرور

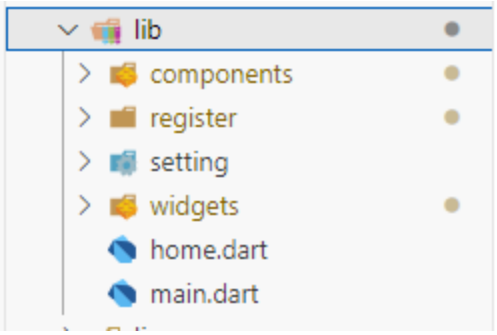
Teleport Client Flutter section -1.4.7

في هذا القسم نسلط الضوء على بناء تطبيق سطح المكتب الخاص بالزبون المتطلبات الوظيفية الخاصة بهذا التطبيق ان يسمح للمطور بما يلي:

- تسجيل الدخول.
- مشاركة بيئة محلية تعمل على منفذ يمكن تحديدهز
- استعراض معلومات إحصائية وسجل الروابط.

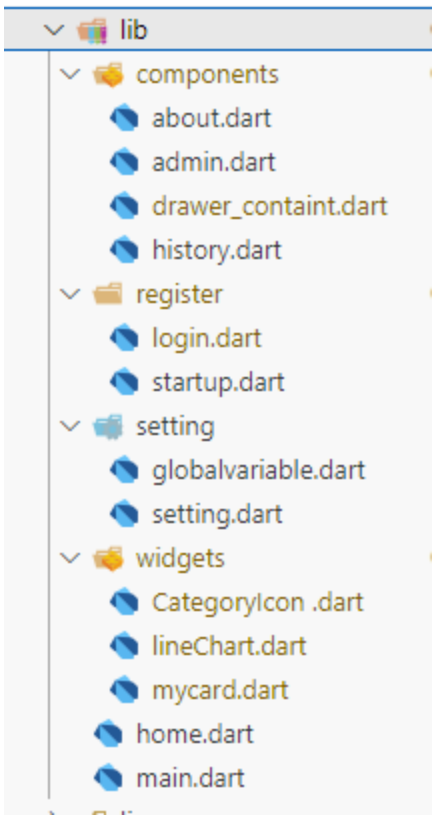
تم تصميم مجلدات التطبيق لتكون اقرب ما يمكن للمعمارية النظيفية Clean architecture مما يسهل لاحقا الصيانة والتطوير.

يظهر الشكل 14.7 بنية مجلدات التي تم انشائها لتنظيم العمل.



الشكل 7. 4 بنية مجلدات تطبيق سطح المكتب

لمعرفة تفاصيل هذه المجلدات لنرى المستوى الثاني في الشكل 15.7



الشكل 7. 5 المستوى الثاني من مجلدات تطبيق سطح المكتب

نلاحظ انه تم الفرز وظيفيا في مجلدات للمساعدة لزيادة الترتيب وتسهيل الصيانة.

يحتوي التطبيق واجهات تخص الزبون (المطور) تقدم له ما يلزم لمشاركة بيئته المحلية والحصول على الاحصائيات وسجل المشاركات.

وأيضا واجهات تخص مدير النظام لعرض احصائيات المستخدمين.

الحزم المستخدمة:

```yaml
dependencies:
  flutter:
    sdk: flutter
  process_run: ^1.0.1

  cupertino_icons: ^1.0.2
  provider: ^6.1.2
  curved_labeled_navigation_bar: ^2.0.2
  shared_preferences: ^2.2.2
  percent_indicator: ^4.2.3
  window_manager: ^0.4.0
  http: ^1.1.0
  url_launcher: ^6.1.14
  fl_chart: ^0.40.0
  system_info2: ^4.0.0

dev_dependencies:
```

الشكل 7. 6 الحزم المستخدمة في تطبيق سطح المكتب

لن نذكر تفاصيل كود لانها عبارة عن تصميم واجهات سيتم عرضها في قسم الواجهات لاحقا.