**<שומר סיסמאות>**

מסמך עיצוב

<עדי בלייאר>

<גרסה 1.2>

<23.5.19>

**היסטוריית גרסאות המסמך**

|  |  |  |
| --- | --- | --- |
| **תאריך** | גרסה | **תקציר השינויים** |
| 20.1.19 | 0.3 | פרק 2 ן 3 מעודכנים לתקופה |
| 5.4.19 | 0.4 | עדכון פרק 2 |
| 19.4.19 | 0.5 | עדכון כל הפרקים, גרסה 1 של סרבר בלבד. |
| 20.4.19 | 0.6 | הוספה של כל התרשימים המתוקנים(סרבר בלבד) |
| 5.5.19 | 0.7 | סידור כללי של החלק על צד שרת. |
| 6-13.5.19 | 0.8 | כתיבת צד לקוח, ללא פרק ממשק משתמש |
| 14.5.19 | 0.9 | שינוי מבנה צד שרת. ארגון והוספת פסקת פתיחה. |
| 15.5.19 | 1 | שינוי מבנה צד לקוח. הוספת פרק דיון בנושא העיצוב הנבחר ופסקת פתיחה |
| 21.5.19 | 1.1 | סידור צד שרת ולקוח, הורדת תיאור קבצים בצד שרת. שינוי דיאגרמות לפי דרישה. |
| 23.5.19 | 1.2 | הוספת נספח חקר והשלמת פרק ממשק משתמש, גרסה מלאה ראשונה. |



**1. הקדמה**

1.1 מטרה

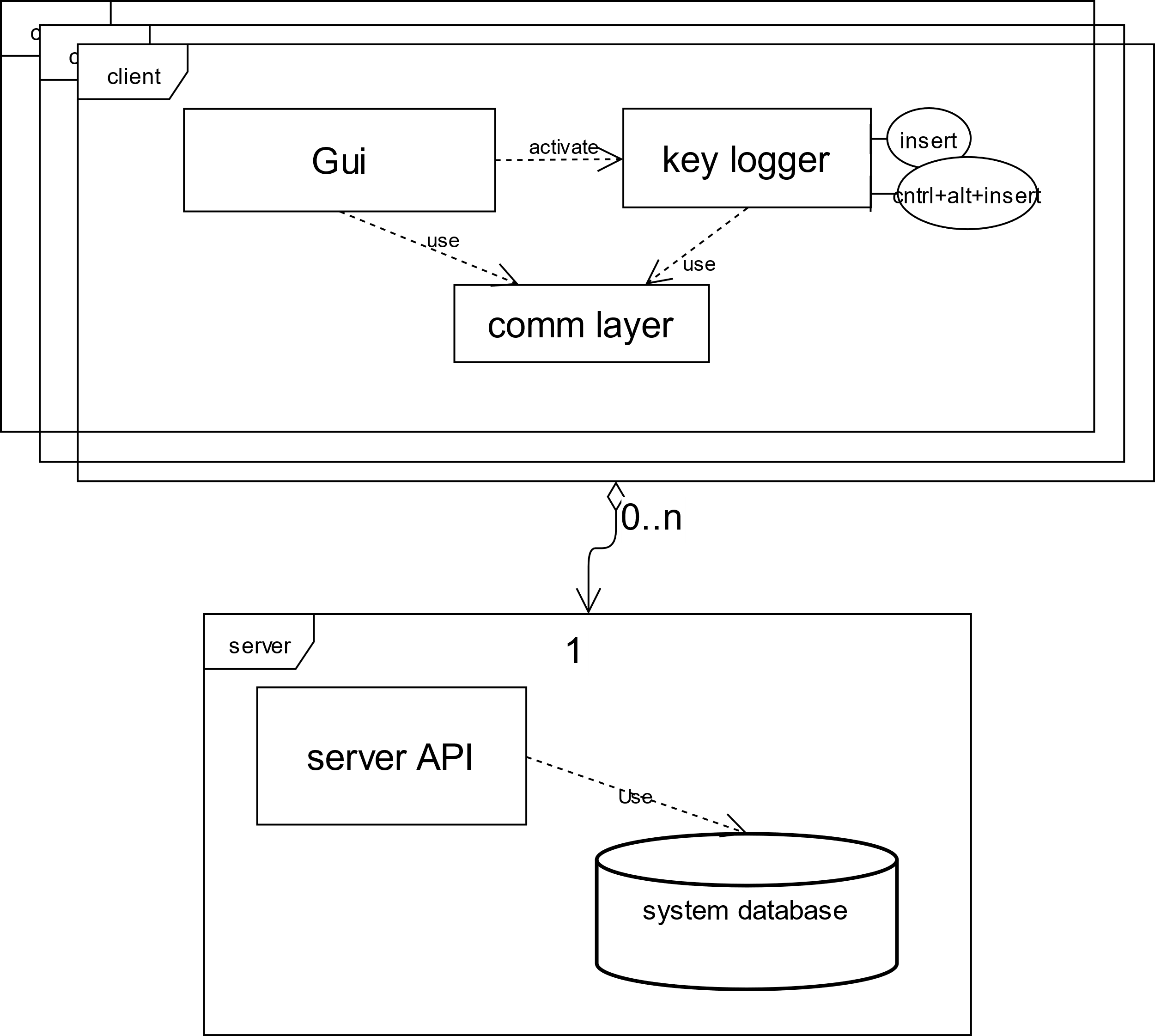
*מסמך שמסביר את עיצוב תוכנות הלקוח והשרת.*

1.2 המוצר

*המוצר הינו שומר סיסמאות. המשתמש מזדהה עם שם משתמש וסיסמא יחידים והתוכנה שומרת לו את שמות המשתמש והסיסמאות לכל אתרי האינטרנט בהם הוא משתמש.*

1. ארכיטקטורת מערכת

2.1 מבט על



**צד הלקוח מכיל שלושה חלקים:**

1. Gui **– ממשק משתמש בסיס של הרשמה התחברות וכולי.**
2. Key logger **– משמש להזרקת סיסמאות והוספת אתרים חדשים, עובד על** chrome **בלבד.**
3. Comm layer **– מטפל בתקשורת מול השרת, נמצא בשימוש בידי שני החלקים האחרים.**

**השרת מספק** api **בסיס מבוסס על פרוטוקול** https **שמאפשר שמירת סיסמאות וניהול מידע המשתמש.**

**השרת הוא** stateless **– מטפל בכל הבקשות באותה צורה, ולא שומר מידע בזמן תקשורת. כל המידע הרלוונטי נשמר על גבי הטוקן שנשלח ללקוח בהתחברות הראשונית.**

**מנגנון ההזדהות של המשתמשים הוא שם משתמש וסיסמא או** JWT**.**

JWT **– טוקן הזדהות שמכיל את נתוני המשתמש, נשלח אליו לאחר הזדהות עם שם משתמש וסיסמא ומשמש להזדהות עתידית. בתוכנה זו הטוקן הוא מסוג** JWE**, זאת אומרת כולו מוצפן ולא ניתן לקריאה על ידי הלקוח.**

2.2 ארכיטקטורת לקוח

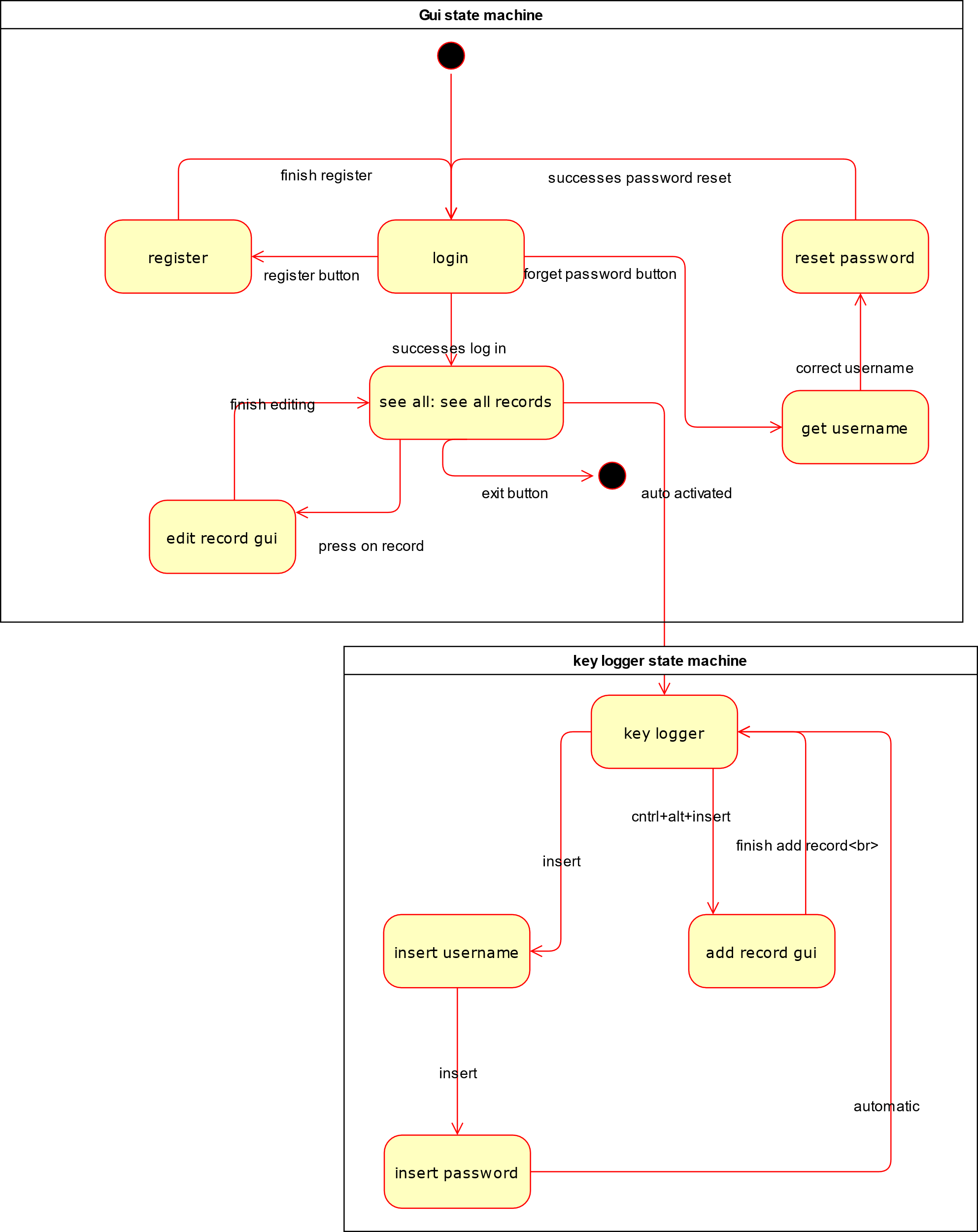
**תוכנת הלקוח מורכבת משני מכונות מצבים.**

**מכונת מצבים ממשק משתמש-אחראית להרשמה, התחברות וכל הממשק הנראה.**

**מכונת מצבים קיצורי מקלדת -אחראית לכל ממשקי המשתמש הקשורים לקיצורי מקלדת, מתחילה את פעולתה לאחר התחברות המשתמש.**

**שתי המכונות רצות במקביל.**

**כל פונקציונאליות של ממשק הלוקח הוא מצב באחת משתי המכונות.**

****

**להזיז או למחוק??**

**תיקיות מרכזיות:**

Gui **– מכיל את כל המשקים הגרפים לכל המצבים**

State **– מכיל את כל המצבים למכונה הראשונה. כל מצב יורש מהממשק הגראפי שלו ומרחיב אותו.**

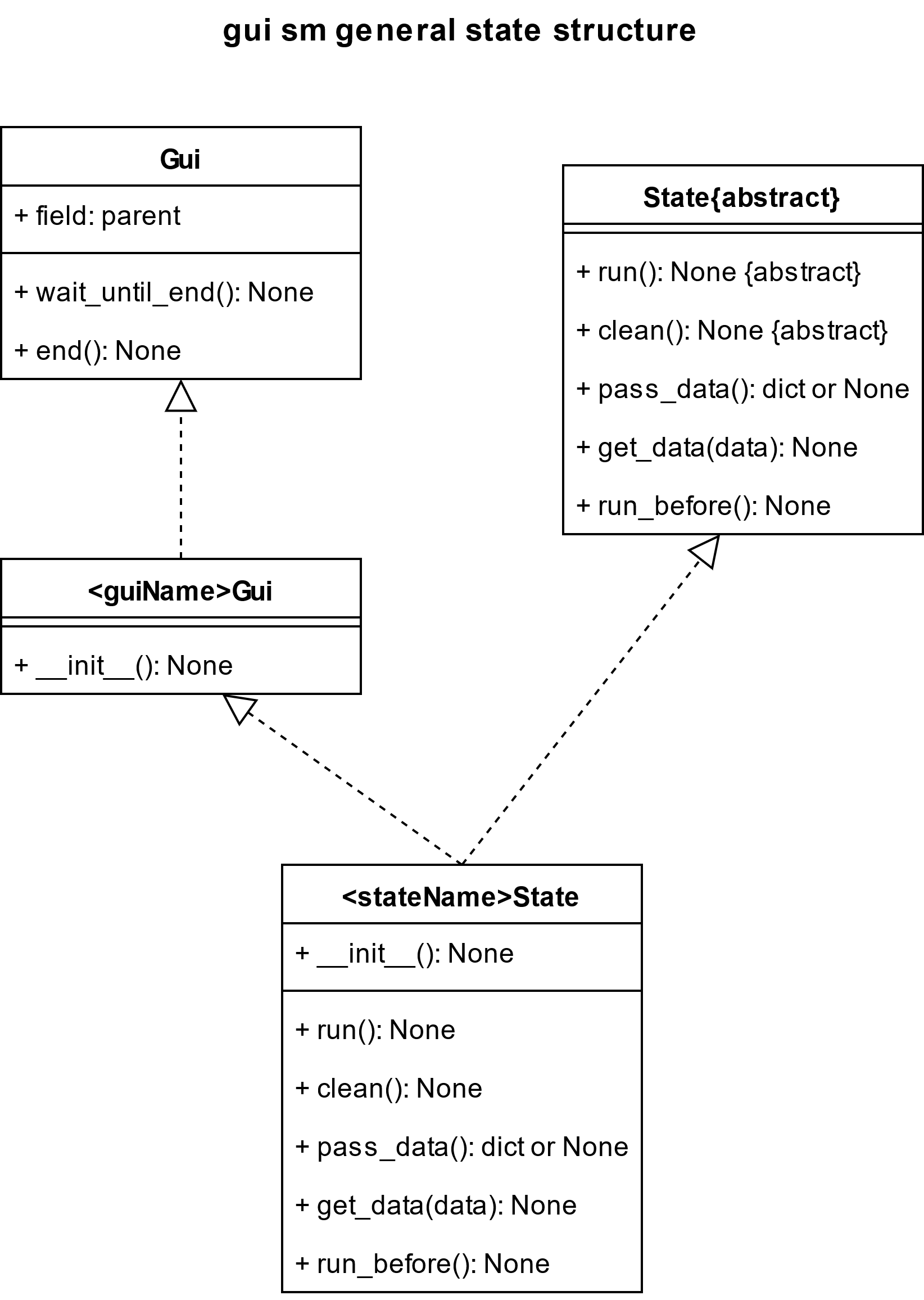
ShasowStates **– מכיל את כל המצבים למכונה השנייה, המצבים לא יורשים ממשקים גרפיים, חלקם משתמשים בהם.**

Api **– מטפל בכל התקשורת. מימוש צד לקוח לממשק של הסרבר. בשימוש על ידי שתי המכונות.**

**מכונת מצבים ממשק משתמש:**

**במכונה זו כל מצב הוא חלון של ממשק משתמש.**

**מבנה של מצב:**

****

**Gui – הגדרה של דף ממשק, כל דף ממשק הוא** frame **בחלון התוכנה. – אחת לכל המצבים**

**<guiName>Gui – מוסיף את כל האלמנטים הגרפים של דף ממשק ספציפי (טקסט כפתורים וכולי).**

**State – מגדיר את הפעולות של כל מצב. מחלקה אבסטרקטית. – אחת לכל המצבים**

**<stateName>State – מממש את כל הפעולות הנדרשות למצב. מממש את כל הפעולות שנעשות לאחר לחיצה על כל אחד מכפתורי המסך.**

**במעבר בין כל מצב למצב יש מספר שלבים:**

1. **קבלת המידע ממצב המקור-** get\_data
2. **ניקוי המידע במצב המקור –** clean
3. **הרצת פונקציה להכנת מצב היעד –** run before
4. **העברת המידע למצב היעד –** pass\_data
5. **העלאת ה**frame **של מצב היעד לקדמת החלון.**
6. **לחכות עד שמצב היעד יודיע שסיים את ריצתו.**

**כל מצב מעביר ומקבל מידע לפי הפירוט הבא:**

**רוב המצבים פועלים גם עם לא קיבלו את המידע, זהו מידע אופציונאלי.**

**Login:**

* **מקבל: שם משתמש וסיסמא**
* **מעביר:**

**Register:**

* **מקבל:**
* **מעביר: שם משתמש וסיסמא**

**Get username:**

* **מקבל: שם משתמש**
* **מעביר: שם משתמש**

**Reset password:**

* **מקבל: שם משתמש - שדה חובה**
* **מעביר: שם משתמש וסיסמא חדשה**

**See all:**

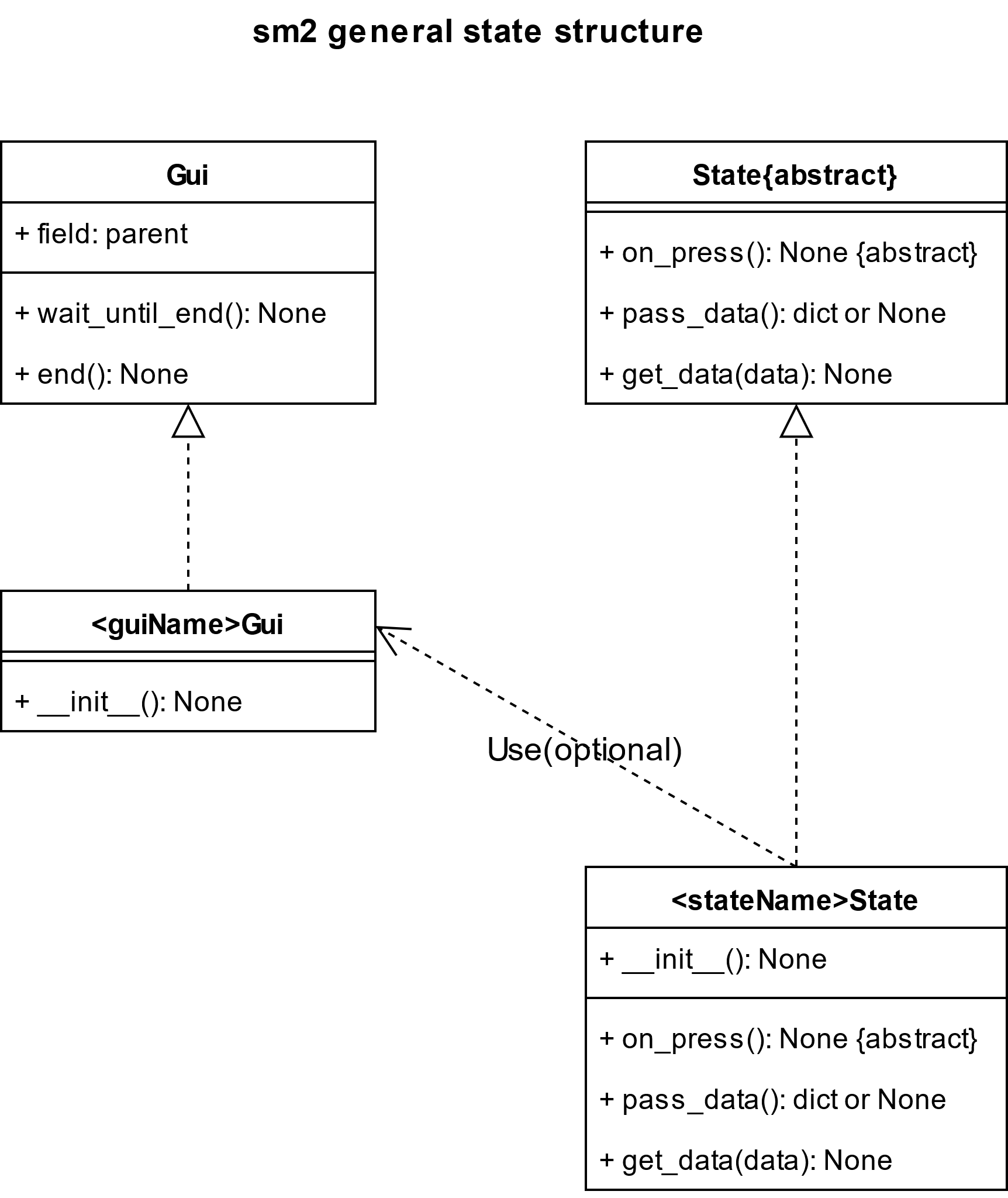
* **מקבל:**
* **מעביר:** uri **במקרה שסומנה על ידי משתמש, אחרת כלום**

**Edit record:**

* **מקבל:** uri **של הרשומה לעריכה – שדה חובה**
* **מעביר:**

**מכונת מצבים קיצורי מקלדת:**

**כל מצב מתאר חלון משתמש או הזרקה של נתונים למסך המשתמש.**

**מבנה כללי של מצב:**

**Gui- הגדרה של חלון. לא אותה הגדרה כמו במכונת המצבים הקודמת.**

**<guiName>Gui - מוסיף את כל האלמנטים הגראפיים של החלון הספציפי.**

**State – מגדיר מצב, לא אותה הגדרה כמו במכונת המצבים הקודמת.**

**<stateName>State – מממש את המצב ואת פונקציונליות כפתורי ממשק המשתמש (עם קיים).**

**במעבר בין כל מצב למצב יש מספר שלבים:**

1. **קבלת המידע ממצב המקור –** get\_data
2. **העברת המידע למצב היעד –** pass\_data
3. **הרצת מצב היעד. –** on\_press

**המצבים היחידים שמקבלים או מעבירים מידע מפורטים להלן:**

**Insert username:**

* **מקבל:**
* **מעביר: שם משתמש וסיסמא**

**Insert password:**

* **מקבל: סיסמא – שדה חובה**
* **מעביר:**

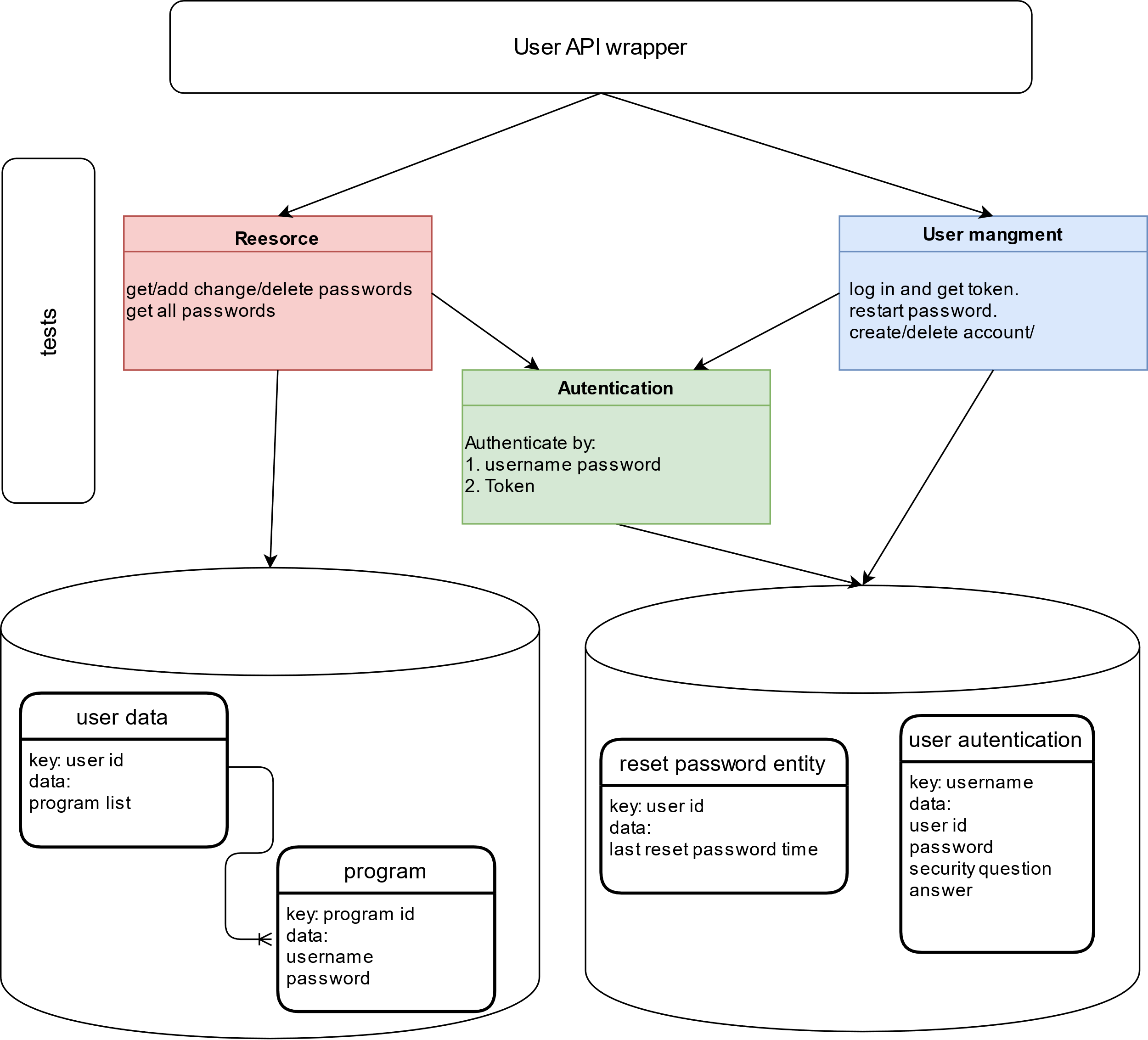
דיון בנושא העיצוב הנבחר

החלטתי לבנות את תוכנת הלקוח בצורת מכונת מצבים. בממשק משתמש בסיסי בו ישנו חלון אחד בלבד הגיוני שתהיה מכונת מצבים, המצב הנוכחי הוא הframe שנמצא בקדמת החלון. עיצוב כל החלונות עם אינטרפייס יחיד מאפשר גמישות בשינוי הקשרים בניהם והוספת קשרים חדשים. כל מצב מדווח בסוף הרצתו את מה עשה (הצלחה, כישלון בגלל סיבה מסוימת, או לחיצה על לחצן מסוים). מכונת המצבים מאפשרת הפרדה מלאה בין המצבים ושינוי המבנה הפנימי של מצב ללא שינוי מצבים אחרים. מקל על הוספה של מצבים חדשים או מעברים חדשים.

במכונה הראשונה המצבים מבוססי ממשק משתמש. המצבים במכונה זו בסיסים וכוללים ממשק משתמש ופעולות בסיסיות על ממשק זה. החלון המרכזי הוא הקובץ שמנהל את מכונת המצבים ומחליט איזה frame להראות למשתמש כל פעם.

במכונה השנייה המצבים עושים דברים שונים והם אינם מבוססי ממשק משתמש ולכן לא יורשים מהם אלה רק משתמשים בהם. מכונה זו ממשיכה לרוץ גם כאשר הראשונה נסגרת שכן הזרקת הסיסמאות צריכה לרוץ ברקע כל הזמן ולא רק כאשר המשתמש מתחבר או מסתכל על נתוני החשבון שלו.

2.3 ארכיטקטורת שרת

****

**השרת מחולק לשלושה חלקים מרכזיים:**

זיהוי משתמשים(**authentication**)**:**

**מממש את שתי אמצעי ההזדהות הקיימים, סיסמא וטוקנים. ממשק פנימי בלבד לשימוש מודולים אחרים.**

ניהולמשתמשים**(user management):**

**מטפל בכל בקשות הלקוח שקשורות לבסיס הנתונים של הזדהות המשתמשים. כגון הרשמה איפוס סיסמא ומחיקת חשבון.**

**שני חלקים אלה משתמשים בממסד נתונים בעל 2 טבלאות:**

* **טבלה של שמות משתמש וסיסמאות.**
* **טבלה של הזמן האחרון בו נעשה שינוי סיסמא (בשימוש בשביל** JWT**, הסבר בהמשך)**

טיפול בבקשות המשתמשים(**Resource**):

**מטפל בבקשות משתמשים שזהותם אומתה, לניהול הסיסמאות.**

**משתמש בממסד נתונים בעל טבלה אחת:**

* **טבלה לכל משתמש, מכילה רשימה של התוכנות שלו.**

**כל תוכנה מכילה מזהה תוכנה ואת שם המשתמש והסיסמא לתוכנה זו.**

**בנוסף לשרת קימות בדיקות. לכל מודול ישנה תיקיה של בדיקות פונקציות מסוימות.**

**הבדיקות מתמקדות בפעולות תקשורת עם ממסד הנתונים ופעולות שחשופות לשימוש מודולים אחרים.**

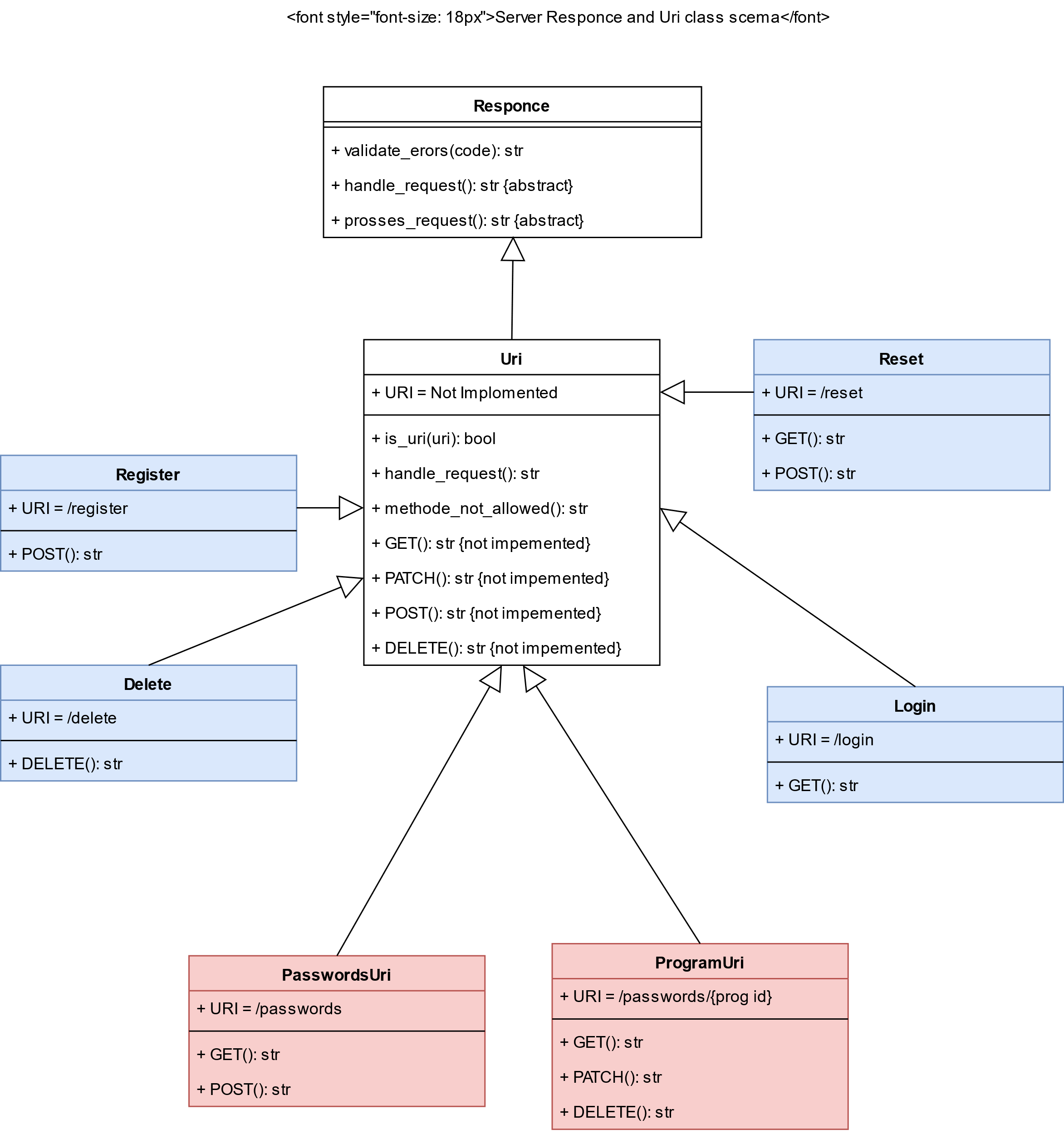
**בנוסף ישנה תיקיה שבודקת את כל הפעולות החשופות לתוכנת הלקוח ומוודא את תקינותו.**

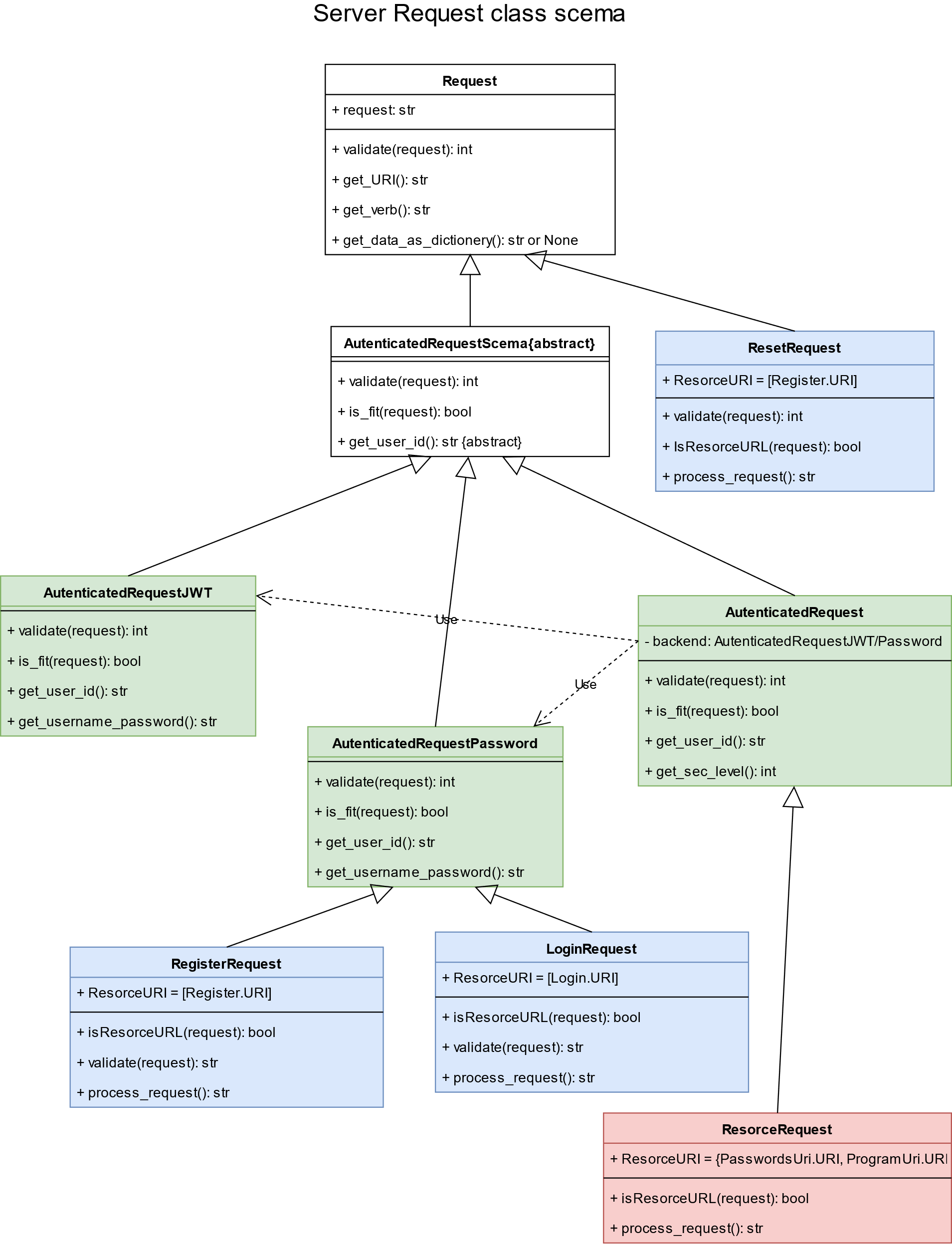
2.3.1 פירוט רכיבי המערכת

**ישנן שתי עצי מחלקות מרכזיים בפרויקט.**

**מחלקות** (Uniform Resource Identifier)URI - **מחלקות הממשות את הפונקציונליות השונות של השרת.**

**מחלקות** Request – **מגדירות מצבי הזדהות שונים, ומקבצות את ה**uri **שניתן לעשות בהם שימוש לאחר הזדהות בפורמט זה.**

**מחלקות URI:**

**מחלקות request:**

**מחלקות ה**Request**:**

**לכל סוג בקשות יש מחלקה כדי לאפשר חלוקה לסוגים, מאפשר לקבץ** URI **בעלי מאפיינים דומים.**

**כל מחלקה מגדירה את התנאים לאישור שימוש ב**URI **שבגבול המחלקה. התנאים קשורים להזדהות המשתמש.**

**חלוקת ה**URI **כדלקמן:**

ResetRequest**- בקשות ללא צורך בהזדהות כלל- שינוי סיסמא**

ResorceRequest**- בקשות לאחר הזדהות, לא משנה באיזו שיטה- בקשה להתחברות**

LoginRequest **– לאחר הזדהות על ידי סיסמא בלבד- התחברות**

RegisterRequest **– לאחר בקשה בפורמט הזדהות תקין של שם משתמש וסיסמא, בלי בדיקה עם קיימת במערכת– הרשמה.**

דיון בנושא העיצוב הנבחר

**כל רכיב לוקח אליו תפקיד אחד משמעותי. זיהוי המשתמשים, טיפול בבקשות קבלת המידע שלהם וטיפול בבקשות הגדרות המשתמש. חלוקה למודולים לפי תפקידים מרכזים עוזרת לארגן את הקוד ולחלק את הפיתוח לחלקים שיכולים להיכבת ולהיבדק בפני עצמם.**

**בחרתי בחלוקה לשני עצי מחלקות מרכזיים.**

**לכל** uri **ישנה מחלקה המטפלת בכל הפעולות של אותו המשאב.**

**לכל סוג בקשות בחרתי לתת מחלקה כדי לאפשר חלוקה לסוגים, מאפשר לקבץ** URI **בעלי מאפיינים דומים. החלוקה לסוגים אפשרה לקבוע לאיזה פעולות למשתמש יש גישה אליהם לפי צורת ההזדהות שמצורפת לבקשה.**

**בחרתי בעיצוב זה כדי להפריד את ההזדהות וניהול ההרשאות של כל משתמש מהפעולות. מחלקות ה** Request **לסוגיהם מגדירות סוגים שונים של בקשות לפי ההזדהות הנדרשת בעוד שמחלקות ה**URI **מטפלות במשתמש לאחר שהזדהה וידוע שיש לו הרשאות לביצוע פעולה זו.**

1. עיצוב נתונים ופרוטוקולים

**תקשורת בין השרת ללקוח:**

פרוטוקול התקשורת בין השרת ללקוח הוא https.

השרת הוא Stateless – כל הבקשות מטופלות באותה הצורה, השרת לא שומר מידע נוסף על הלקוח בין בקשות.

מבנה הAPI בין השרת ללקוח:

**URI:/reset**

GET:

Register

* Data Params:

{username: [string]}

* Success Response:

Code: 200

Content :{question: [str]}

* Error Response:

Code: 442 unexpected entity.

Content:{ username: username error}

Code: 500 internal server error

* Sample Call:

PATCH:

Change user username or password

* Headers:
* Data Params:

 {username:[string],answer:[string], NewPassword:[string] , NewUsername[string]} all the new fields are optional.

* Success Response:

Code: 200

* Error Response:

Code: 442 unexpected entity, one of the follows:

* + Username is wrong (not found)
  + Answer is wrong
  + Data is missing.

Related massage will be in the response body.

Code: 500, internal error.

* Sample Call:

**URI:/register**

POST:

Register

* Data Params:

Authentication=[string]- username:password in base64 encoding

Phone number: = [int]

Trusted account username= [str] – username of other client

* Success Response:

Code: 200

* Error Response:

Code: 442 unexpected entity.

Content:{password: pass error, username: username error} all the fields are optional.

Code: 500 internal server error

Code 401 unauthorized, can’t register with JWT authentication.???? Is correct??

* Sample Call:

**URI: /login**

GET:

Get JWT token to use when access data

* Data Params:

Authentication=[string]- username:password on base 64 encoding

* Success Response:

Code: 200   
Content: {Authentication: JWT token}

* Error Response:

Code: 442 unexpected entity, username or passwords is wrong.

Code 401 unauthorized, can’t log in with JWT authentication.

* Sample Call:

**URI: /passwords**

GET:

Get all programs ID and usernames

* Headers:

Authentication=[JWT token]

* Success Response:

Code: 200  
Content: {

records:[

{username:[string], programID:[string], sec\_level: [string]}

……

]

}

* Error Response:401, JWT token have invalid data.
* Sample Call:

POST:

Add new program username and password record

* Headers:

Authentication=[JWT token]

* Data Params:

{username:[string], password:[string], programID:[string], 'sec\_level': [int]

* Success Response:

Code: 200

Code: 442 unexpected entity, if one of the data params values is missing.

Data params:

{ username : [str]….}, all the values are optional.

Code: 500 internal server error

* Error Response:
* Sample Call:

**URI: /passwords/{program ID}**

GET:

Get program username and password

* Headers:

Authentication=[JWT token]

* Success Response:

Code: 200  
Content: {username : [string], password:[string], SecLevel:[string]}

* Error Response:

Code:401 unauterized, when authentication level is not enough.

Header:

SecLevel: int

Code: 401 unauthorized, when authentication is wrong.

Code: 404 not found, program id don’t exist.

* Sample Call:

PATCH:

Change program username/password/programID

* Headers:

Authentication=[JWT token]

* Data Params:

{: {username:[string], password:[string], programID:[string] } all of the fields are optional.

* Success Response:

Code: 200

* Error Response:

Code: 404 not found, program id doesn’t exist.

* Sample Call:

DELETE:

Delete password username pair from database

* Headers:

Authentication=[JWT token]

* Success Response:

Code: 200   
Content: { }

* Error Response:

Code: 404 not found, program id doesn’t exist.

* Sample Call:

**URI: /delete**

DELETE:

Delete user

* Headers:

Authentication= username:password in url safe base64 encode

* Success Response:

Code: 200

* Error Response:

Code: 400 bad request, Authentication entity is wrong formatted.

Code: 442 unexpected entity, username or passwords is wrong.

Code 401 unauthorized, can’t log in with JWT authentication

Code: 500 internal server error

* Sample Call:

**מידע שנשמר בלקוח:**

מפתח הזדהות- בקובץ בשם token שנמצא בתוך קבצי הקוד.

ssl certification – האישור שהתקבל מהשרת, נשמר בקובץ cert בתוך תיקיות הקוד.

**מידע שנשמר בשרת:**

בסיסי נתונים - mongo dB שומר נתונים בצורת JSON וכך נכתב בסיס הנתונים.

1. זיהוי ואימות משתמשים.
   1. זיהוי משתמשים
      1. אינדקס ראשי: שם משתמש - מחרוזת
      2. סיסמת המשתמש (hash+salt) - מחרוזת
      3. שאלת אבטחה - מחרוזת
      4. תשובה לשאלת הבטחה (hash+salt) - מחרוזת
   2. אימות טוקנים:
      1. אינדקס ראשי: מספר משתמש
      2. התאריך האחרון של שחזור סיסמא.
2. משאבי משתמשים
   1. אינדקס ראשי: שם משתמש
   2. האתרים והסיסמאות של המשתמשים – רשימה המכילה:
      1. מזהה ייחודי לתוכנה (הURL של אותו אתר ב safe\_url base 64)
      2. שם המשתמש לתוכנה - מחרוזת
      3. סיסמא לתוכנה - מחרוזת

קבצים נוספים:

לוגים: לוגים נשמרים בתיקיה logs. הלוגים מכילים את כל הלוגים מכל המרכיבים השונים של השרת. שמורים באופן טקסטואלי.

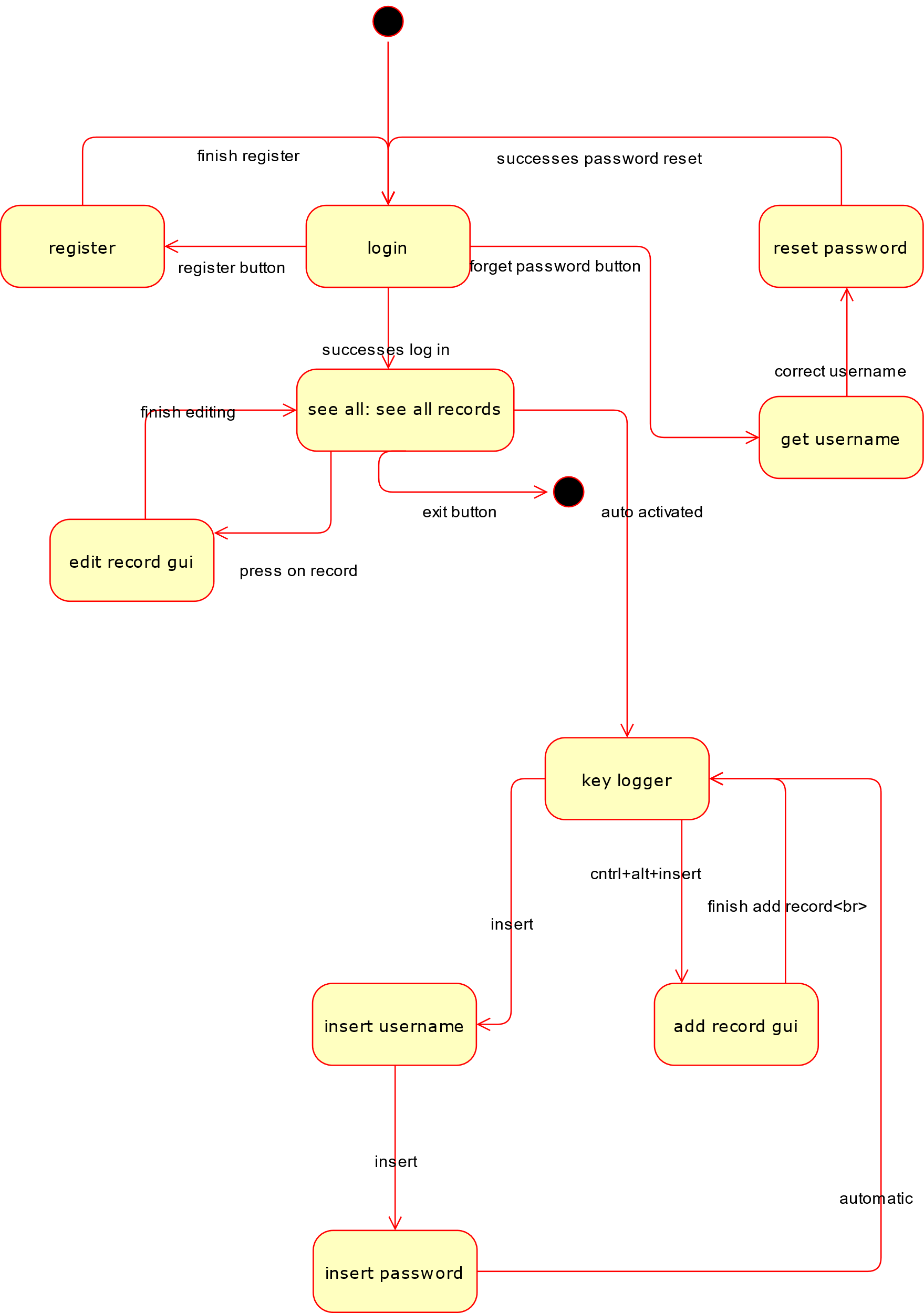
מפתחות הצפנה: נמצאים בקבצים בתיקיות הקוד.

אישור ssl ומפתח פרטי: נמצא בתיקיית ssl בקוד.

4. ממשק משתמש

מכונת מצבים זו מתארת את כל תוכנת הלקוח.

כל מצב מתאר חלון שצריך למלא או פעולת הזרקה שנעשית על יד התוכנה. כל מעבר מייצג פעולה של המשתמש (ישנם מספר מעברים אוטומטים- דבר המצוין בשמם)



**תמונות מסכי ממשק המשתמש למצבים השונים:**

|  |  |
| --- | --- |
| **Login:** | **Register:** |
| **Get username:** | **Reset password:** |
| **See all:** | **Edit record:** |
| **add record gui:** |  |

**מצבים ללא מסך:**

**Insert username – מזריק את שם המשתמש למקום בו העכבר נמצא (אחריות המשתמש למקם את העכבר בתוך התיבה המיועדת לשם המשתמש)**

**Insert password – מזריק את הסיסמא למקום בו העכבר נמצא (אחריות המשתמש למקם את העכבר בתוך התיבה המיועדת לסיסמא)**

5.מסמך חקר

**בשל הרגישות של סיסמאות המשתמש הפרויקט מתמקד בהבטחת מידע.**

**בתחילה רציתי לבחור את אמצעי ההזדהות שידרשו לשירות.**

**אמצעי ההזדהות הראשון והשימושי ביותר לכל שירות קיום הוא שם משתמש וסיסמא. שימוש בהצפנת** ssl **יאפשר העברה מוצפנת של שם משתמש וסיסמא בתהליך ההתחברות בין השרת ללקוח.**

**בנוסף שמירת הסיסמאות בבסיס הנתונים צריכה להיות מאובטחת, כך שפריצה אליו לא תוביל לגילוי סיסמאות המשתמשים.**

סיסמא (והתשובה לשאלת הבטחה) הם ערכים שמצריכים אימות בלבד, השרת לא צריך לדעת את סיסמאות המשתמשים אלא רק לדעת האם הסיסמא שהמשתמש הזין נכונה או לא. במקרים כאלה ניתן להשתמש בפונקציית hash.

פונקציית hash- פונקציה שמוציא מכל פלט קלט שנראה רנדומלי. מהסיסמא לhash שלה ניתן לעבור בקלות אך לא ניתן מהhash של הסיסמא לשחזר את הסיסמא. כך בבסיס הנתונים נשמר הhash של הסיסמא ובמקרה של פריצה לא מתגלות הסיסמאות.

בנוסף בhash למטרות הבטחת מידע פונקציית הhash מורצת פעמים רבות על הקלט. הפיכת פונקציית הhash לאיטית וקשה לחישוב מגדיל בצורה ניכרת את הזמן שצריך בשביל לעשות brute force לסיסמא. לכן בחרתי בhash איטי והרצתי אותו מספר רב של סיבובים כך שכל hash ייקח כ0.1 שניות.

בנוסף הסיסמאות נשמרות בsalt. פונקציית הhash מוכרת ולכן לעשות hash רק לסיסמאות לא יספיק. התוקף יכול לקחת את הסיסמאות הנפוצות ביותר, להעביר אותם בפונקציית הhash ולחפש בממסד הנתונים, כך ניתן להשיג סיסמאות רבות.

הוספה של מחרוזת רנדומלית בתחילת ובסוף הסיסמא (לפני הפעלת פונקציית הhash), שונה לכל משתמש מונעת התקפה זו ומגנה על סיסמאות המשתמשים.

**בנוסף לכך רוב השירותים קיום תומכים בצורה כזו או אחרת בהשארת המשתמש מחובר זמן רב לאחר התחברות ראשונית, דבר שנעשה בעזרת מפתח מסוים. מפתח זה מונפק לאחר התחברות, ומשמש את המשתמש להוכיח את זהותו.**

**ישנם שני דרכים עיקריות לשמור מידע זה:**

**Session based – המידע נשמר על השרת, ללקוח ניתן** ID **(מחרוזת רנדומאלית) שאיתה יוכל להזדהות. על השרת נשמר זהות הלקוח לו הונפק** ID **זה, לכל בקשה הלקוח מצרף מחרוזת זו וכך השרת יכול לאמת את זהותו בלי לבקש סיסמא מחדש.**

**Token based – המידע נשמר על מפתח הזדהות שנשלח ללקוח. מפתח זה מכיל את המידע על זהות הלוקח, חתום קרטוגרפית ולפעמים גם מוצפן. בכל בקשה מפתח ההזדהות נבדק ועם תקין מתקבלת ממנו זהות הלקוח.**

**הבדלים מהותיים:**

**במפתח הזדהות המידע על המשתמש המשתמש רשום בטוקן ועוד ב** session **הוא נכתב בבסיס נתונים.**

**יתרון עיקרי של מפתח הזדהות:**

**חוסך שימוש בבסיס נתונים**

**הופך את השרת ל**stateless**.**

Stateless**- לא שומר מידע על המשתמש בין בקשות ומטפל בכל בקשות המשתמשים אותו הדבר.**

**יתרונות:**

* + **מקל על הפיתוח**
  + **מפחית שימוש בזיכרון (לא צריך לשמור מידע על המשתמש)**

**חסרון עיקרי של מפתח הזדהות:**

**לא מאפשר שינוי המידע מהרגע שהונפק הטוקן ועד שהוא פג(זמן רב).**

**בתוכנה זו המידע היחידי שצריך להיות על משתמש הוא זהותו. הפעם היחידה בה צריך לשנות את המידע על משתמש הוא בזמן שינוי סיסמא. לאחר שינוי סיסמא צריך לנתק את כל מי שהיה מחובר לחשבון ולדרוש ממנו את הסיסמא החדשה.**

**הפתרון הוא שימוש בטוקן עם בסיס נתונים. בתוקן נשמר ה**client id **כך שמשתמש יוכל להזדהות עם התוקן. ישנו סיס נתונים ששומר את זמן שינוי הסיסמא האחרון של כל משתמש כך שטוקנים שנוצרו לפני שינוי סיסמא לא יהיו תקפים. ממסד נתונים זה קטן בצורה משמעותי מזה שהיה צריך להישמר בשביל** session **שכן הוא מכיל רשומות רק למי ששינה סיסמא בזמן האחרון ולא לכל המשתמשים המחוברים לשירות. דבר זה מתאפשר בגלל שלמפתח הזדהות יש זמן תפוגה. עם משתמש שינה סיסמא לפני זמן רב ניתן להיות בטוחים שכל התוקנים שהונפקו לפני שינוי הסיסמא כבר פגו, ולכן הוא לא מצריך רשומה בבסיס הנתונים. פתרון זה מאפשר להפוך טוקן ללא ולידי אחרי שהונפק ועדיין שומר על בסיס נתונים קטן יותר משהיה נדרש בפתרון המבוסס על** session**.**