**<שם הפרויקט>**

מסמך עיצוב

<עדי בלייאר>

<גרסה X>

<תאריך>

**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. |



**1. הקדמה**

החלק הזה ישמש להצגה כללית של מסמך העיצוב. הסבר כללי, וקישור למסמכים קודמים כמו מסמך האפיון עליו אנו מתבססים

1.1 מטרה

*מה המטרה של המסמך ומי קהל היעד שלו*

1.2 המוצר

*חלק זה צריך להכיל את הדברים הבאים (ניתן לתמצת מתוך מסמך האפיון):*

* *שם המוצר אותו מפתחים*
* *הסבר כללי של מה המוצר עושה*

1.3 קישור למסמכים קודמים

*קישור למסמכים / קבצים חיצוניים הרלוונטיים למסמך. לדוגמא: מסמך אפיון, עמוד web עם עיצוב הממשק הכללי שאתם מתכננים וכו'.*

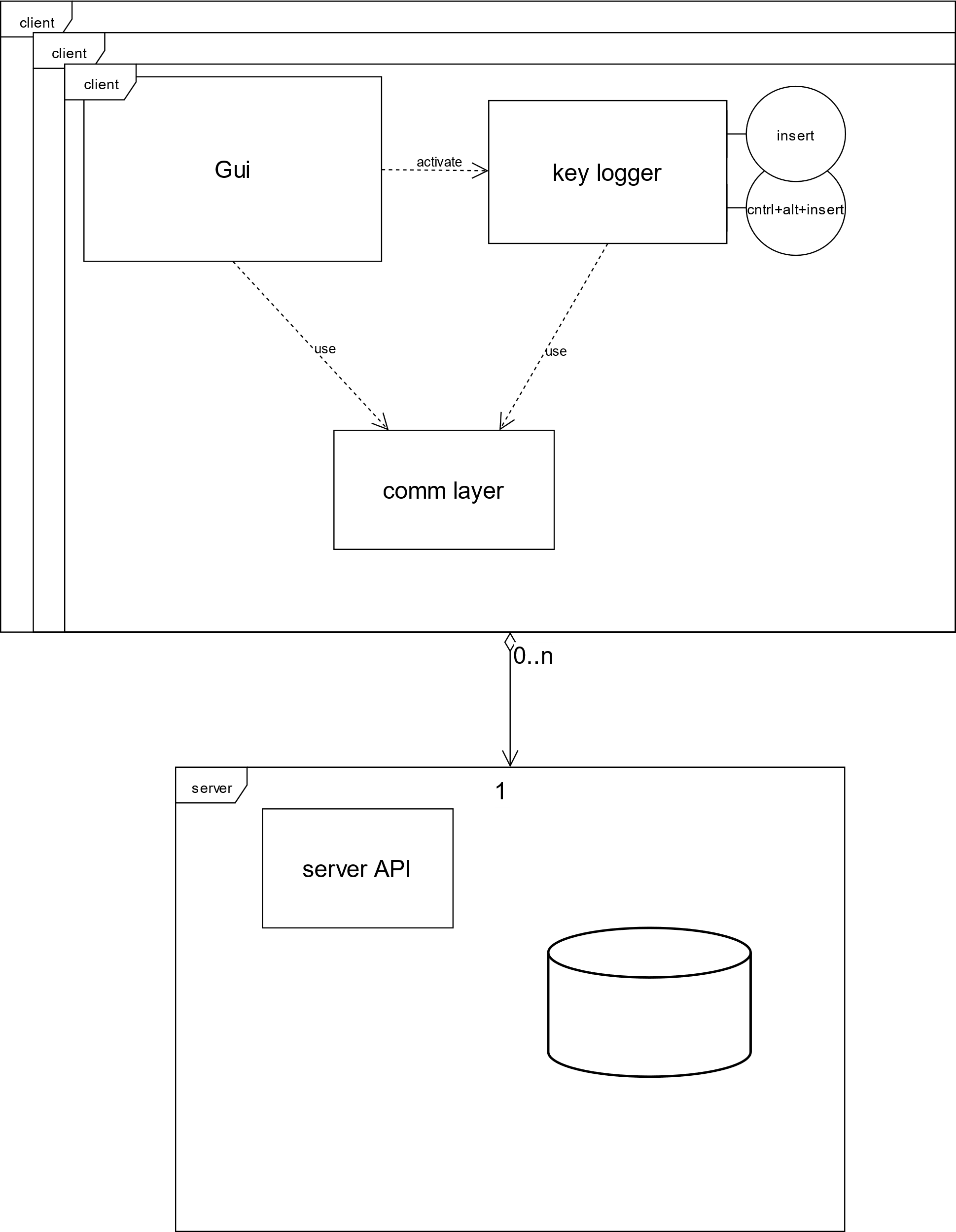
1.4 הגדרות

*רשומה – שם משתמש וסיסמא של שירות אחר.*

1. ארכיטקטורת מערכת

חלק זה כולל את תיאור מבנה המערכת ופירוט המודולים השונים בה

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

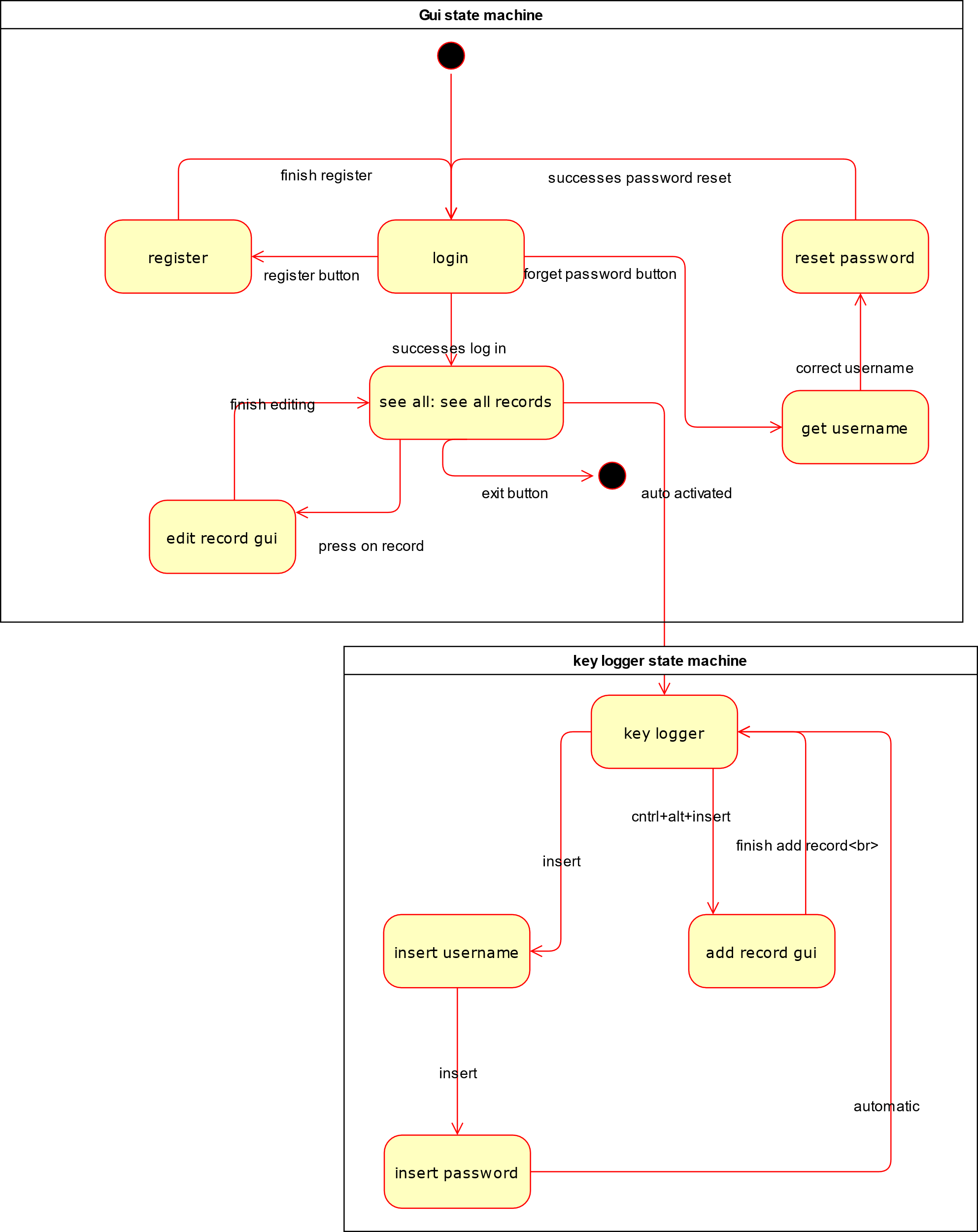
**תוכנת הלקוח מורכבת משני מכונות מצבים.**

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

**מכונת מצבים שנייה(שם???)- 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.**

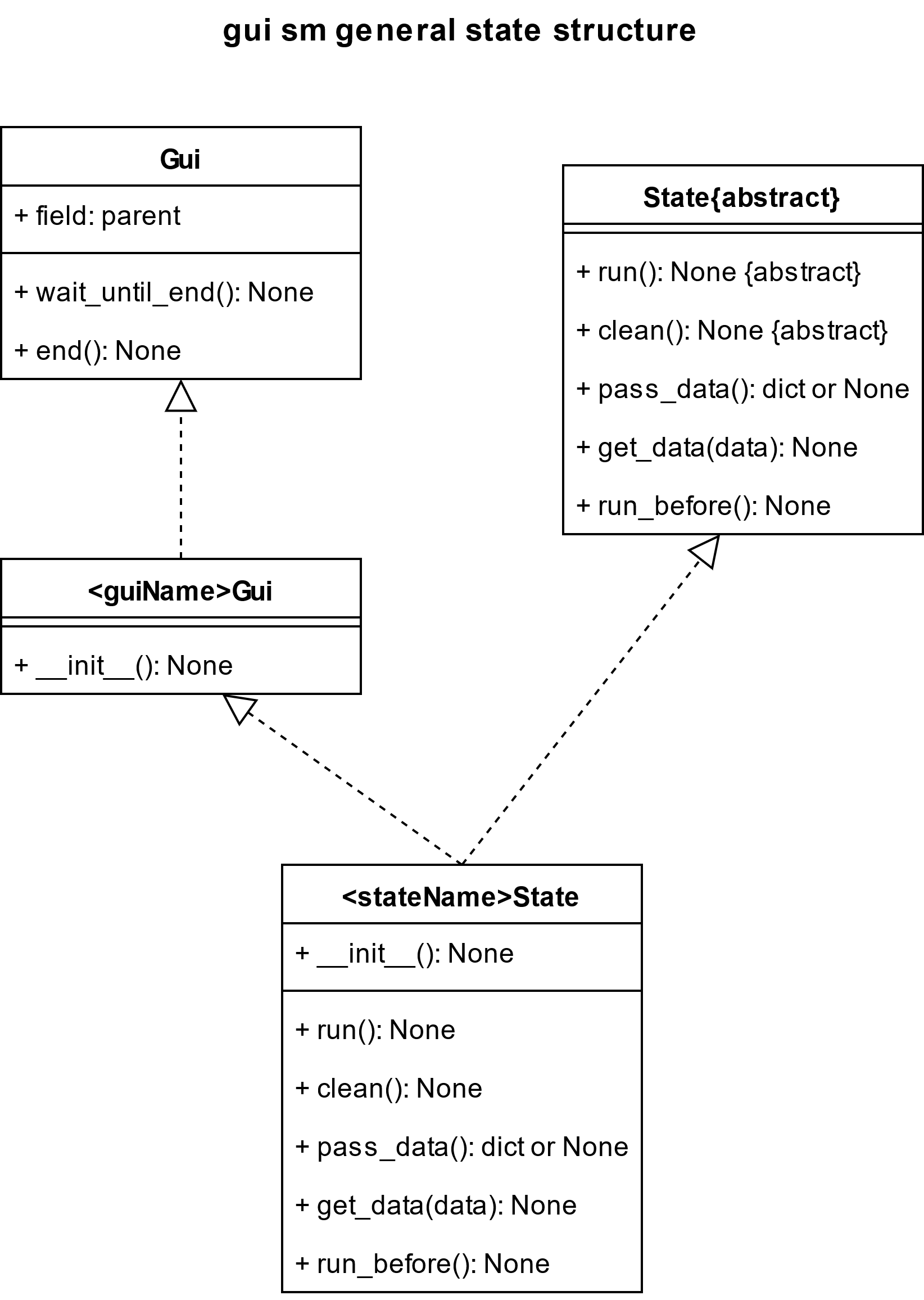
ShasowStates **– מכיל את כל המצבים למכונה השנייה, המצבים לא יורשים ממשקים גרפיים, חלקם משתמשים בהם.**

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:**

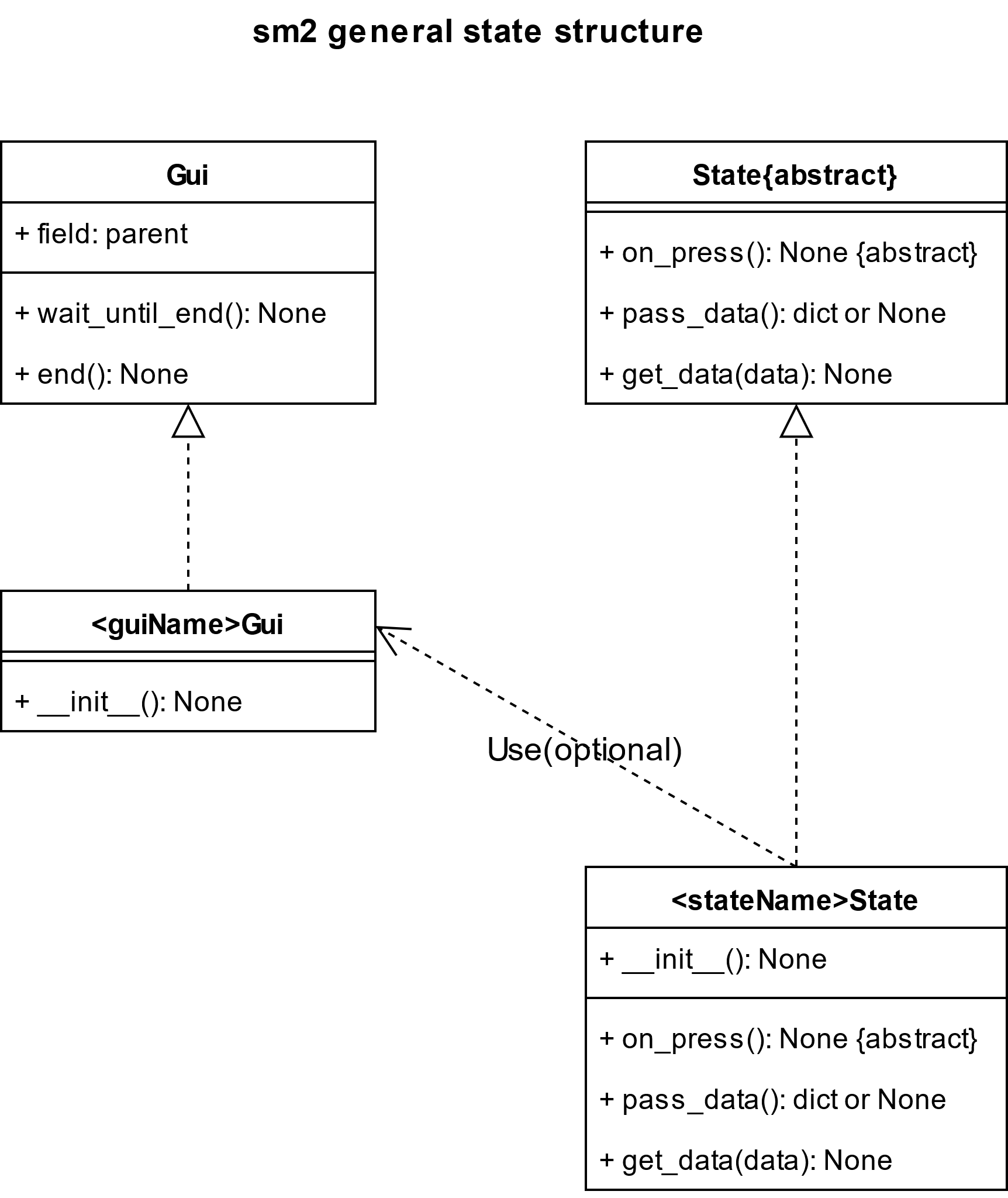
* **מקבל:**
* **מעביר:** uri **במקרה שסומנה על ידי משתמש, אחרת כלום**

**Edit record:**

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

**מכונת מצבים key logger:**

**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:**

2.2.3 דיון בנושא העיצוב הנבחר

לאחר שחשבתי על המבנה הכללי של ממשק המשתמש החלטתי לבנות אותו 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)**

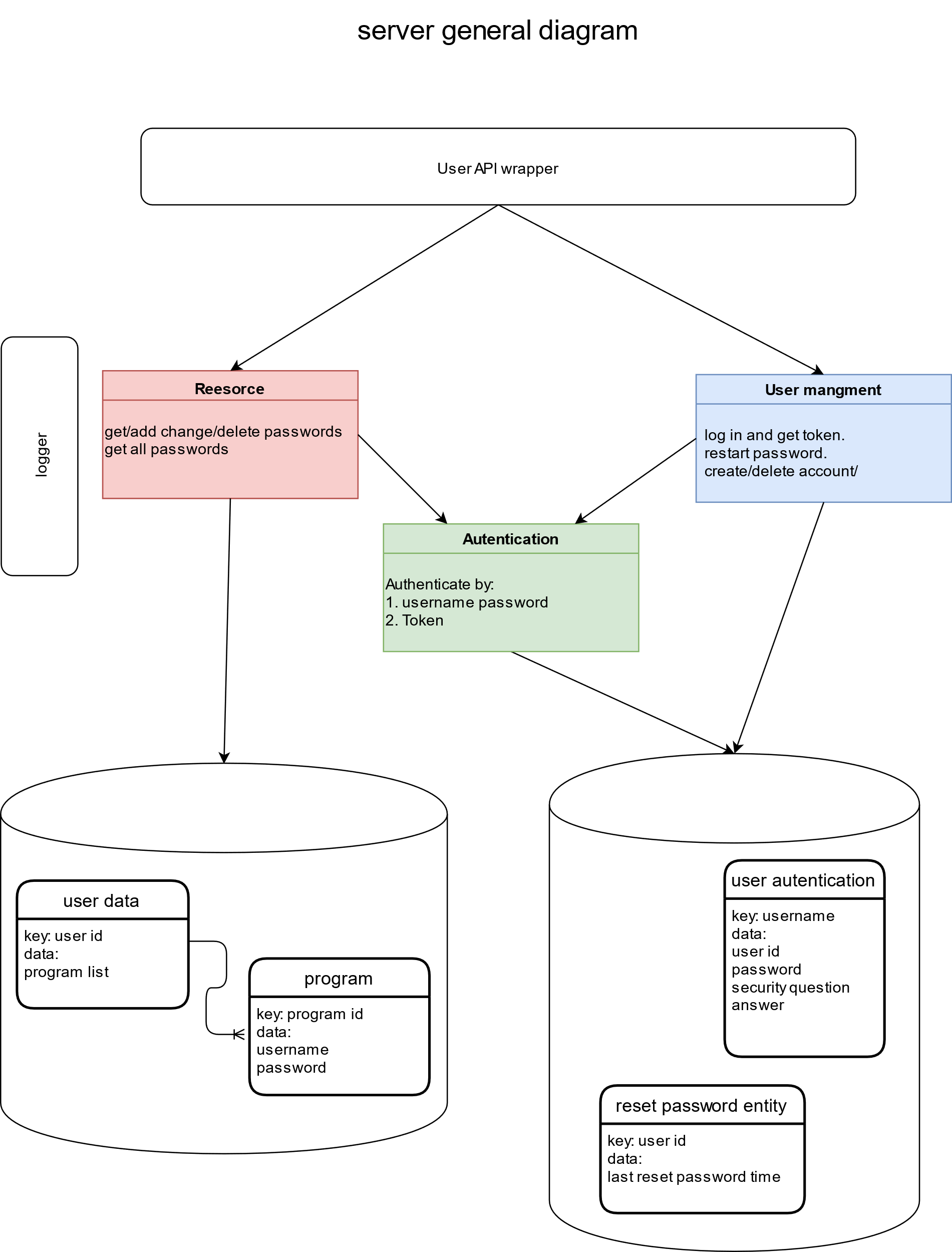
טיפול בבקשות המשתמשים(**Resource**):

**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.**

****

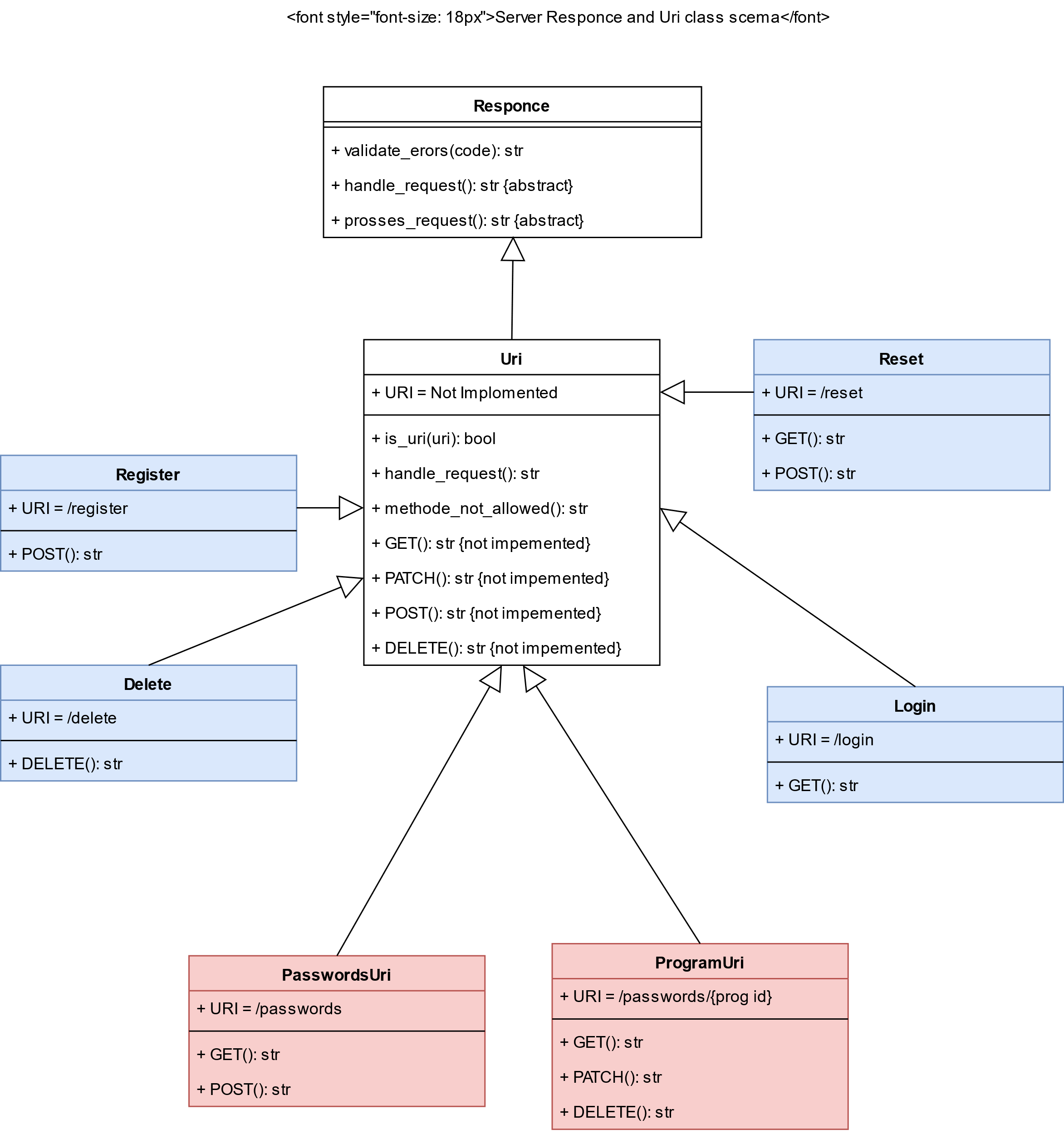
2.3.1 פירוט רכיבי המערכת

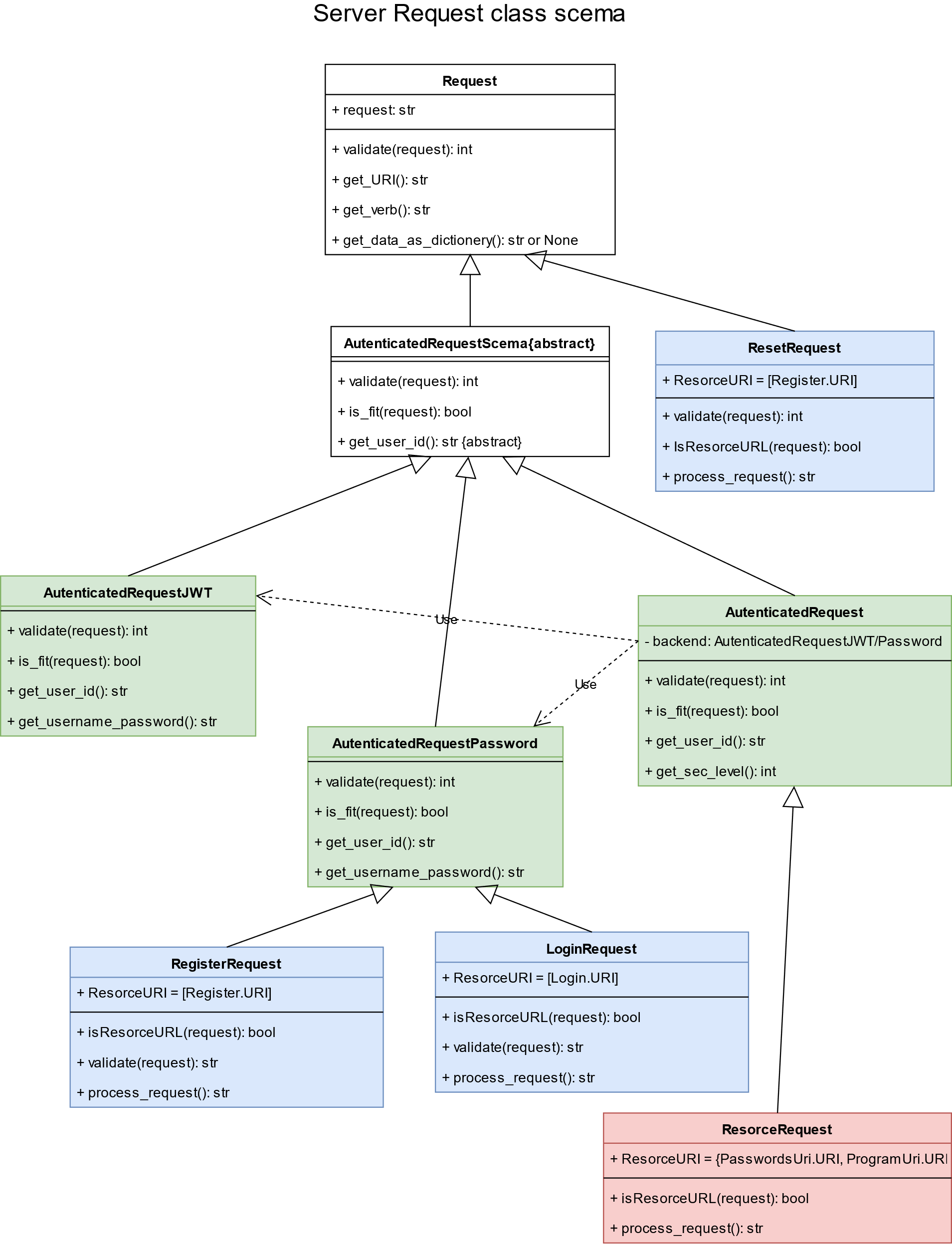
**ישנן שתי עצי מחלקות מרכזיים בפרויקט.**

**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 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

הבחירה בשרת stateless:

בשומר סיסמאות מספר הפניות לשרת מהתחברות עד להתנתקות הוא מועט ונפרש על זמן רב. השארת socket פתוח זמן רב אינה אפקטיבית.

לאחר ההתחברות המשתמש צריך להישאר מחובר זמן ללא דרישה להזדהות מחדש.

לא ניתן לשמור את המידע על המשתמש כחלק מsession שכן בין התחברות להתנתקות יכולים להיות session רבים.

השימוש בשרת stateless מאפשר התחברות לתווך זמן ארוך.

**הבחירה בצורת הזדהות JWT: חסר- opcode שם לא נכון????**

**ישנם שני סוגי הזדהות נפוצים:token/ session:**

**session- המשתמש מקבל מהשרת מספר. השרת שומר בממסד נתונים את ההרשאות שיינתנו למחזיק מספר זה.**

**JWT – המשתמש מקבל טוקן חתום על ידי הסרבר שמכיל את הרשאותיו, כאשר השרת מקבל אישור זה הוא מוודא שהחתימה נכונה ונותן ללקוח את ההרשאות כפי שרשומות בטוקן.**

**הבדל מהותי: בjwt המידע על הרשאות המשתמש רשום בטוקן ועוד ב opcode הוא נכתב בבסיס נתונים.**

**יתרונות של jwt:**

1. **חוסך שימוש בבסיס נתונים**

**חסרונות של jwt:**

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

**הפתרון הוא שימוש בטוקן עם בסיס נתונים.**

**בתוקן נשמר הclient id כך שמשתמש יוכל להזדהות עם התוקן. שנו 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.**

**חלוקה למודולים ומחלקות:**

**כל רכיב לוקח אליו תפקיד אחד משמעותי. זיהוי המשתמשים, טיפול בבקשות קבלת המידע שלהם וטיפול בבקשות הגדרות המשתמש.**

**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 לפעולה זו.**

1. עיצוב נתונים ופרוטוקולים

כאן יופיע תיעוד של מבני נתונים שונים / פרוטוקולים המשמשים אותנו במערכת

בסעיף זה יש לפרט את כל סוגי המידע אשר מועבר או נשמר במערכת - בין אם בין רכיבים, בין מודולים או בכל צורה אחרת (כלומר גם אם זה משמש רק לתקשורת פנימית ולא ע"ג הרשת, או למשל אם זה משהו ששומרים לקובץ/בסיס נתונים).

* לכל סוג מידע כזה יש לפרט על השדות אותן הוא מכיל ומאיזה סוג כל שדה, מה טווח הערכים הרלוונטי לגביו וכל הגבלה או מידע נוסף אחר שאתם מוצאים לנכון (למשל: האם מותר שהשדה יהיה ריק ומה המשמעות של זה, אורך או ערך מינימלי/מקסימלי למספר או למחרוזת)
* יש לציין לאיזו מטרה משמש כל מבנה נתונים.
* עבור פרוטוקול תקשורת-נתונים בין רכיבים שונים:
* מה המצבים השונים לכל סוג של תקשורת (למשל אותנטיקציה, הרשמה, שליחת קובץ וכו')
* באילו ייצוגי מידע נעשה שימוש + תרשים זרימה של המצבים השונים שיכולים להיות וסדר השלבים שלהם (למשל כיצד נעביר מידע על מקרים של סיסמה שגויה, משתמש חסום, התחברות מוצלחת וכו')
* מומלץ להגדיר מראש קודים של בקשות/פעולות, תוצאות/תגובות ושגיאות אשר ישותפו בין כך הרכיבים הרלוונטיים
* יש להתייחס לאופי המידע הנשמר (האם הוא טקסטואלי או בינארי) ולייצגו בצורה נוחה והולמת בהתאם
* יש לפרט אודות מבנה בסיס הנתונים (חלוקה לטבלאות, השם והטיפוס של כל עמודה/שדה, קשרים בין טבלאות ואילוצים כלשהם על עמודות)

פרוטוקול התקשורת בין השרת ללקוח מבוסס על http.

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

הערה: מבנה הAPI הכללי של התוכנה מוגדר בקובץ נפרד תיאור ממשק

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. סיסמאת 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. - ???? איזה סוג זמן זה????
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
      4. רמת אבטחה – מספר (אחד או אפס)

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: נמצא בתיקיה בקוד.

**צורת שמירת סיסמאות ותשובה לשאלת הבטחה:**

סיסמא והתשובה לשאלת הבטחה הם ערכים שמצריכים אימות בלבד, אני 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 למתרות הבטחת מידע פונקציית הhash מורצת פעמים רבות. הפיכת פונקציית הhash איטית וקשה לחישוב מגדיל בצורה ניכרת את הזמן שצריך בשביל לעשות bruteforce 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 פונקציית הhash), שונה לכל משתמש מונעת התקפה זו ומגנה על סיסמאות המשתמשים.

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)

[client state machine-without.svg](client%20state%20machine-without.svg)

**תמונות מסכי ממשק המשתמש למצבים השונים:**

**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)**