**Joined Calendar**

תמונה שמכילה טקסט, צילום מסך, גופן, גרפיקה

התיאור נוצר באופן אוטומטי

**שם: יובל צ'יקו**

**ת"ז: 328373618**

**בת ספר: כרמים בנימינה**

**מורה מלווה: מרי גבע**

**מועד הגשה: מאי 2024**

פרק א' – יומן שיתופי - ייזום

1. **תיאור ראשוני של המערכת:**

פרויקט המאפשר לנהל יומן עם עוד אנשים וכך לראות מה הזמן הפנוי המשותף, מה הלוז של האחר... בפרויקט אפשר לפתוח כמה יומנים שרוצים עם אנשים שונים, ולנהל עם כל אחד יומן שונה – להכניס את האירועים – מה, מתי ,ואיפה האירוע. אפשר גם לראות רק את האירועים שלך ולקבל התראות לטלפון לפני שקורים. בנוסף אפשר לבקש משאר משתתפי היומן לקבוע אירוע לשעה מסוימת, והם יוכלו לאשר – או לא. במידה וכולם יאשרו האירוע ייכנס ליומן המשתתפים.

בחרתי בפרויקט מכיוון שאני נתקלת המון בבעיה שקשה לקבוע אירועים להרבה אנשים כך שכולם יוכלו להגיע, ואני חושבת שזה יכול לחסוך המון זמן בחיי היום יום.

הקשיים הצפויים לי בפרויקט הם ה עיצוב הפרויקט כך שיהיה נוח ונגיש למשתמש, סנכרון יומנים רבים.

1. **הגדרת הלקוח:**

המערכת מיועדת למשפחות עם ילדים גדולים שמנסות לקבוע לבלות זמן יחד אך מתקשות למצוא זמן, לקבוצות חברים גדולות שלכל אחד לוח זמנים שונה, ואף חברות עבודה גדולות שרוצות לנהל פגישות בצורה יעילה יותר.

1. **הגדרת יעדים/מטרות:**

המטרות המרכזיות של המערכת הן לנהל זמן בצורה נכונה יותר, יעילה יותר וקלה יותר. להקל על משפחות עם ילדים גדולים שמנסות לקבוע לבלות זמן יחד אך מתקשות למצוא זמן, על קבוצות חברים גדולות שלכל אחד לוח זמנים שונה, ואף על חברות עבודה גדולות שרוצות לנהל פגישות בצורה יעילה יותר.

1. **בעיות, תועלות, וחסכונות:**

הבעיה היא שלכל אדם יש לוח זמנים שונה ולכן לפעמים קשה לקבוע דברים משותפים – במיוחד להרבה אנשים יחד.

התועלות שסביר לצפות מהמערכת הן שכך יהיה קל יותר לקבוע דברים משותפים. זה יחסוך זמן של הסתכלות ביומנים ואפילו יחסוך קביעת תוכניות שבסוף לא יקרו כי לא כולם יכולים. בנוסף כך הורים יוכלו לדעת איפה ילדיהם ולא ידאגו להם כשהם לא בבית, מנהלים ידעו איפה עובדיהם ועל מה עובדים ועוד.

המערכת תיתן שירותים כגון ניהול יומנים משותפים עם קבוצות אנשים שונות, יומן פרטי שגם משכלל את האירועים של הלקוח בכל היומנים, תזכורות לטלפון לאירועים בשעה רצויה, לשאול את שאר המשתתפים ביומן האם הם יכולים לקבוע תוכנית מסוימת בשעה ספציפית – ואם כן, להכניס אותה ליומנים שלהם.

Groupcal –

<https://apps.apple.com/il/app/groupcal-%D7%A9%D7%99%D7%AA%D7%95%D7%A3-%D7%99%D7%95%D7%9E%D7%A0%D7%99%D7%9D/id1472335927?l=he>

האפליקציה מאפשרת פתיחת יומנים משותפים עם אנשים שונים והצגת כל יומן בנפרד.

family organizer -

<https://apps.apple.com/il/app/family-organizer-calendar/id978882773?l=he>

האפליקציה מאפשרת לפתוח יומן משותף אחד – למשפחה. היא מאפשרת לשים תזכורת לפני האירוע. בנוסף חוץ מאירועים היא מאפשרת להכניס דברים שצריך לעשות, לכתוב דברים שרוצים (כמו סיכום יום), לכתוב כמה כסף הרווחת או בזבזת באותו יום, מאפשרת לראות את מזג האוויר ועוד.

1. **האם צפויים קשיים או מגבלות בהגדרת המערכת:**

הטכנולוגיה שמצריך הפרויקט מוכרת ולא צריך בשבילה ציוד מיוחד.

1. **תיחום הפרויקט:**

הפרויקט מתעסק בתחום המערכות הפעלה, רשתות, אבטחה, אך מתעסק רק בתקשורת מקומית ולא בחיצונית. בנוסף הפרויקט מתעסק באנדרואיד אך רק בקבלת התראות.

1. **קיימות:**

הפרויקט מאפשר ניהול יומנים מרובים, גם פרטיים וגם משותפים, וכך חוסך הדפסה של המון יומנים ודפים שמזהמים את כדור הארץ וגורמים לכריתה של עצים.

פרק ב' – יומן שיתופי - אפיון

1. **פירוט המערכת:**

פרויקט המאפשר לנהל יומן עם עוד אנשים וכך לראות מה הזמן הפנוי המשותף, מה הלוז של האחר... בפרויקט אפשר לפתוח כמה יומנים שרוצים עם אנשים שונים, ולנהל עם כל אחד יומן שונה – להכניס את האירועים – מה, מתי ,ואיפה האירוע. אפשר גם לראות רק את האירועים שלך ולקבל התראות לטלפון לפני שקורים. בנוסף אפשר לבקש משאר משתתפי היומן לקבוע אירוע לשעה מסוימת, והם יוכלו לאשר – או לא. במידה וכולם יאשרו האירוע ייכנס ליומן המשתתפים. לדוגמא הורים שרוצים לקבוע זמן לאירוע משפחתי יוכלו לשלוח לילדים "הזמנה" לאירוע, ואם כולם יוכלו האירוע ייכנס ליומנים שלהם.

המערכת אמורה להקל וליעל קביעת תוכניות עם אנשים רבים, להזכיר על התוכניות לפני שקורות. בנוסף להקל על המעקב של לוז המשתתפים (לדוגמא שהורים שידעו איפה ילדיהם.

1. **מה היכולות שהיא תעניק למשתמש, פירוט היכולות:**

* רישום כמשתמש במערכת / התחברות למערכת (משתמש קיים)
* פתיחת יומנים עם אנשים שונים – הלקוח יכול לפתוח מספר יומנים שונים עם אנשים שונים לבחירתו, שבהם יראה את האירועים המשותפים
* יומן אישי – אופציה לראות את כל האירועים של הלקוח ביומן נפרד
* "הזמנה" לאירוע – לשלוח הזמנה לאירוע למשתתפי היומן (או חלק מהמשתתפים) ובמידה וכולם מאשרים האירוע יתווסף ליומן המשותף (ולכל משתתף ביומן הפרטי)
* יצירת אירוע אישי / משותף
* הזמנה ליומן – לשלוח הזמנה ליומן למשתמשים אחרים ובמידה ומאשרים מתווספים ליומן
* תזכורת – הלקוח יכול להגדיר שלאירוע תהיה תזכורת זמן מסוים (שהוא בוחר) לפני שהוא מתקיים
* ביטול אירוע מהיומן (אם אני בעל האירוע) ו/או ביטול השתתפות באירוע (אם הוזמנתי לאירוע)
* עריכת אירוע (שם, משתתפים, שעה, תאריך) ויומן (שם, משתתפים)

1. **פירוט הבדיקות:**

| **מספר** | **שם הבדיקה** | **מה אמורה לבדוק** | **איך מתכננים לבדוק** |
| --- | --- | --- | --- |
| 1 | הוספת יומן | האם המערכת מאפשרת הוספת יומן | להיכנס למערכת ולנסות לפתוח יומן |
| 2 | הוספת אדם ליומן | האם אפשר להוסיף אדם ליומן קיים | להיכנס ליומן קיים ולהוסיף לו משתתף |
| 3 | הוספת אירוע | לבדוק האם אפשר להוסיף אירוע | להיכנס ליומן קיים ולהוסיף אירוע |
| 4 | מחיקת אירוע | האם אפשר למחוק אירוע קיים | להיכנס לאירוע קיים ביומן קיים ולמחוק אותו |
| 5 | יציאה מיומן | לבדוק האם משתתף יכול לצאת מיומן וימחקו כל האירועים שלו | להיכנס ליומן קיים ולצאת ממנו. פעם אחת כאשר המשתמש הוא מנהל היומן ופעם אחת כאשר לא |
| 6 | יצירת משתמש | האם אפשר ליצור משתמש חדש במערכת | להיכנס למערכת וליצור משתמש חדש עם הפרטים הנחוצים |
| 7 | התחברות למשתמש | לבדוק האם אפשר להתחבר למשתמש קיים | להתחבר למשתמש קיים |
| 8 | הוספת משתמש לאירוע | האם אפשר להוסיף משתמש לאירוע | להיכנס לאירוע ולשלוח הזמנה למשתמש |
| 9 | שינוי שם יומן | האם המנהל יכול לשנות את שם היומן | ליצור יומן ולנסות לשנות את השם שלו |
| 10 | שינוי שם אירוע | לבדוק אם אפשר לשנות שם אירוע | ליצור אירוע ולנסות לשנות את השם שלו |
| 11 | שינוי שעות אירוע | לבדוק אם אפשר לשנות את שעות האירוע | ליצור אירוע ולנסות לשנות את השעות שלו |
| 12 | שינוי תאריך אירוע | לבדוק אם אפשר לשנות את התאריך שלו | ליצור אירוע ולנסות לשנות את התאריך שלו |
| 13 | הוספת משתתף ליומן כאשר אתה לא מנהל היומן | לבדוק אם המערכת חוסמת הוספת משתתף ליומן אם אתה לא המנהל שלו | להצטרף ליומן שלא אתה יצרת (דרך ההזמנות) ולנסות לצרף משתמש |
| 14 | שינוי זמן האירוע כאשר משתתף באירוע לא פנוי בזמן זה | לבדוק אם המערכת חוסמת שינוי זמן אירוע לזמן שאחד או יותר ממשתתפיו לא פנויים בו | ליצור אירוע רב משתתפים (להזמין אנשים לאירוע ושיאשרו אותו) ולאחר מכן לנסות לשנות את השעה לשעה שלא כל המשתתפים פנויים בה |
| 15 | הוספת משתתף לאירוע ביומן שהוא לא משתתף בו | לבדוק אם המערכת חוסמת הוספת משתתף לאירוע דרך יומן שלא משתתף בו | ליצור יומן ואירוע ולנסות להזמין אליו משתמש שלא נמצא ביומן |
| 16 | אישור הזמנה לאירוע | לבדוק האם המערכת מאפשרת אישור הזמנה לאירוע | להיכנס להזמנות ולאשר הזמנה לאירוע (שמשתמש אחר כבר שלח) |
| 17 | אישור הזמנה ליומן | לבדוק האם המערכת מאפשרת אישור הזמנה ליומן | להיכנס להזמנות ולאשר הזמנה ליומן (שמשתמש אחר כבר שלח) |

1. **תכנון לוח זמנים לפרויקט:**

| **פעילות** | **זמן התחלה מתוכנן** | **זמן סיום מתוכנן** | **זמן התחלה בפועל** | **זמן סיום בפועל** | **הערות** |
| --- | --- | --- | --- | --- | --- |
| יזום | 5.11 | 23.11 | 5.11 | 6.11 | הקדמת זמנים |
| אפיון | 23.11 | 1.12 | 6.11 | 9.11 | הקדמת זמנים |
| ניתוח | 1.12 | 15.12 | 9.11 | 9.11 | הקדמת זמנים |
| עיצוב | 15.12 | 15.1 | 9.11 | 27.1 | איחור זמנים |
| בניית לקוח ושרת בסיסיים  בניית בסיס נתונים  ממשק משתמש בסיסי(הרשמה/התחברות)  טיפול ביומן (פתיחת יומן, הזמנת חברים, הוספת אירוע, מחיקת אירוע)  הצפנה | 10.1  13.1  16.1  19.1  16.2 | 13.1  16.1  19.1  16.2  28.2 | 27.1  27.1  20.2  20.2  18.2 | 27.1  18.2  25.3  11.5  19.2 | איחור בזמנים  איחור בזמנים  איחור בזמנים בעקבות שינוי בסדר עדיפויות (עשיית כל פעולות הלוגיקה לפני התחלת הגרפיקה)  איחור בזמנים  הקדמת זמנים בעקבות בסדר עדיפויות |
| גרסה ראשונית | 10.1 | 1.3 | 27.1 | 1.3 | עמידה בזמנים |
| מסמך בדיקות | 1.3 | 10.4 | 16.5 | 16.5 | איחור בזמנים |
| מדריך למשתמש | 10.4 | 1.5 | 17.5 | 17.5 | איחור בזמנים |
| גרסה סופית | 1.3 | 10.5 | 1.3 | 11.5 | איחור בזמנים |
| סגירת תיק פרויקט | 10.5 | 15.5 | 11.5 | 18.5 | איחור בזמנים |

1. **ניהול סיכונים בפרויקט:**

| **הסיכון** | **פירוט הסיכון** | **רמת הסיכון**  **(קל/בינוני/קשה)** | **תיאור דרכים (לפחות 2 ) להתמודדות עם הסיכון ולהקטין אותו** | **מה בוצע בפועל** | **תאריך** |
| --- | --- | --- | --- | --- | --- |
| אי עמידה בזמנים | פרויקט לא יושלם | קשה | * הקדמת לו"זים משימות * ארגון הזמן בצורה יעילה * להתחיל בדברים היותר קשים |  |  |
| הוספת לקוח אנדרואיד | לא יהיו התראות לטלפון | **קל** | * למידה מקדימה בנושא * הקדמת לוח זמנים |  |  |
| עבודה עם wxpython | לא להצליח להתקדם בפרויקט | קשה | * ללמוד איך לעבדו עם זה מראש * להקדים את לוח הזמנים (כך שיהיה זמן לעיכוב) |  |  |
| הצפנה | פרויקט לא מאובטח | קשה | * למידת סוגי הצפנות רבות * הקדמה בלוח הזמנים |  |  |
| מערכת משתמש נגישה | מערכת לא נוחה | בינוני | * לתת לאנשים אחרים להפעיל את המערכת ולבדוק במה התקשו * לחקור מה מקל על אנשים להפעיל מערכות ומה מקשה * לנסות כמה אופציות עד שיש אחת טובה |  |  |
| סנכרון יומנים | פרויקט לא יאפשר כמה יומנים ללקוח | בינוני | * להקדים את לוח הזמנים * חקירה מוקדמת על דרכים שונות לאיך לסנכרן הרבה מידע |  |  |
| עומסים במערכת | הפרויקט יגיב לאט לבקשות המשתמשים | בינוני | * להגביל מספר משתמשים * לבדוק איך ליעל את הפעולות |  |  |

פרק ג' – יומן שיתופי - ניתוח

**פרוט יכולות המערכת:**

יכולות בצד לקוח:

* שם היכולת: הרשמה למערכת

מהות היכולת: רישום משתמש חדש במערכת (קליטת פרטיים אישיים נדרשים)

אוסף יכולות:

* ממשק משתמש – מסך הרשמה
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת

* שם היכולת: התנתקות ממשתמש

מהות היכולת: התנתקות המשתמש מהמערכת

אוסף יכולות:

* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – יציאה של המשתמש מהמערכת

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת

* שם היכולת: התחברות למערכת (למשתמש קיים)

מהות היכולת: התחברות למשתמש הקיים במערכת

אוסף יכולות:

* ממשק משתמש – מסך הרשמה
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש
* פתיחת מידע של המשתמש (יומנים ואירועים קיימים...)

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, אירוע

* שם היכולת: פתיחת יומן

מהות היכולת: לפתוח יומן למשתמש

אוסף יכולות:

* התחברות למשתמש
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – פתיחת היומן

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן

* שם היכולת: הוספת אירוע

מהות היכולת: להוסיף אירוע ליומן קיים

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – הוספת האירוע ליומן

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, אירוע, תזכורת

* שם היכולת: עריכת אירוע

מהות היכולת: לערוך אירוע קיים

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* פתיחת אירוע
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – שינוי האירוע ביומן

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, אירוע, תזכורת

* שם היכולת: מחיקת אירוע

מהות היכולת: למחוק אירוע קיים

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* פתיחת אירוע
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – מחיקת האירוע מהיומן והתזכורת (במידה ויש)

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, אירוע, תזכורת

* שם היכולת: עריכת יומן והוספת משתמשים

מהות היכולת: לערוך יומן קיים ולהוסיף לו משתמשים

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – הוספת משתמש / שינוי צבע אירועים

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, צבע אירוע

* שם היכולת: עריכת יומן והוספת משתמשים

מהות היכולת: לערוך יומן קיים ולהוסיף לו משתמשים

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – הוספת משתמש / שינוי צבע אירועים

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, צבע אירוע

* שם היכולת: יציאה מיומן

מהות היכולת: יציאת משתמש מיומן

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול
* הצגת התשובה למשתמש – הורדת היומן

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן

* שם היכולת: שליחת הזמנה לאירוע

מהות היכולת: שליחת הזמנה לאירוע לחברי היומן – אם מאשרים האירוע מתווסף

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* פתיחת אירוע
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, אירוע

* שם היכולת: אישור / סירוב הזמנה לאירוע

מהות היכולת: אישור / סירוב הזמנה לאירוע שנשלחה מאחד חברי היומן – אם מאשרים האירוע מתווסף

אוסף יכולות:

* התחברות למשתמש
* פתיחת יומן
* פתיחת אירוע
* קליטת נתונים
* בדיקת תקינות
* בנייה לפי פרוטוקול
* הצפנה
* שליחה לשרת
* קבלת תשובה מהשרת
* פענוח
* פירוק לפי פרוטוקול

רשימת אובייקטים: ממשק משתמש, פרוטוקול, הצפנה/פיענוח, תקשורת, יומן, אירוע

יכולות בצד שרת:

* שם היכולת: הרשמה למערכת

מהות היכולת: רישום משתמש חדש במערכת

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* גיבוב סיסמא
* הוספה לבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול

* שם היכולת: התחברות למערכת (למשתמש קיים)

מהות היכולת: התחברות למשתמש הקיים במערכת

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* גיבוב סיסמא
* לקיחת מידע על המשתמש מבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול

* שם היכולת: פתיחת יומן

מהות היכולת: פתיחת יומן והכנסת המידע שלו לבסיס הנתונים

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* הוספה לבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן

* שם היכולת: הוספת אירוע

מהות היכולת: הוספת אירוע והכנסת המידע שלו לבסיס הנתונים

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* הוספה לבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, אירוע, יומן

* שם היכולת: עריכת אירוע

מהות היכולת: עריכת אירוע ושינוי המידע שלו בבסיס הנתונים

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* הוספה לבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן, אירוע

* שם היכולת: מחיקת אירוע

מהות היכולת: מחיקת אירוע ומחיקת המידע שלו מבסיס הנתונים (במידה והמשתמש הוא בעל האירוע אז מחיקה מכל המשתתפים בו ובמידה והוא לא בעל האירוע אז רק אצלו)

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* מחיקה מבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן, אירוע, בעל האירוע

* שם היכולת: עריכת יומן

מהות היכולת: עריכת יומן והוספת משתמשים

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* הוספה לבסיס נתונים – את האירועים החופפים
* לקיחת מידע מבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן

* שם היכולת: יציאה מיומן

מהות היכולת: לאפשר למשתמש יציאה מיומן קיים

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* בדיקה מול בסיס הנתונים
* מחיקה מבסיס נתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן

* שם היכולת: סנכרון יומן אישי ומשותפים

מהות היכולת: להציג את האירועים בצורה הנוחה למשתמש ולרכז את כל האירועים שלו ביומן אישי

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* הוספה לבסיס הנתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת – של היומנים המסונכרנים

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן, אירוע

* שם היכולת: הזמנה לאירוע

מהות היכולת: הזמנה של שאר משתתפי היומן לאירוע – אם מאשרים האירוע מתווסף

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* הוספה לבסיס הנתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת
* שליחת הזמנה מוצפנת לשאר משתתפי היומן

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן, אירוע

* שם היכולת: אישור / סירוב לאירוע

מהות היכולת: אישור / סירוב לאירוע הנשלח מחבר אחר ביומן – אם כולם מאשרים האירוע מתווסף

אוסף יכולות:

* קבלת נתונים מהלקוח
* פענוח
* פירוק לפי פרוטוקול
* הוספה לבסיס הנתונים
* בנייה לפי פרוטוקול
* החזרת תשובה מוצפנת
* אם כולם אישרו – הוספת האירוע ליומן אצל שאר חברי הקבוצה

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, יומן, אירוע

* שם היכולת: שליחת תזכורת \*\* במידה ויהיה לי מספיק זמן אממש

מהות היכולת: שליחה לטלפון המשתמש תזכורת על אירוע – בזמן על פי בקשת המשתמש

אוסף יכולות:

* בדיקה של זמני התזכורות והשעון
* בדיקת בסיס הנתונים
* מחיקת התזכורת – לאחר השליחה
* הצפנה
* בנייה לפי פרוטוקול
* שליחה לטלפון תזכורת מוצפנת

רשימת אובייקטים: הצפנה/פענוח, תקשורת, בסיס נתונים, פרוטוקול, תזכורת, אנדרואיד

פרק ד' – יומן שיתופי – עיצוב

1. תיאור הארכיטקטורה של המערכת המוצעת

שרת מרובה לקוחות על בסיס socket. כל הלקוחות, השרת ובסיס הנתונים באותה רשת, אך הטלפון (android) ברשת נפרדת



1. תיאור הטכנולוגיה הרלוונטית

שפת תכנות: python בשרת ובלקוח, sql בבסיס נתונים, java באנדרואיד

מערכת הפעלה: אנדרואיד ו-windows

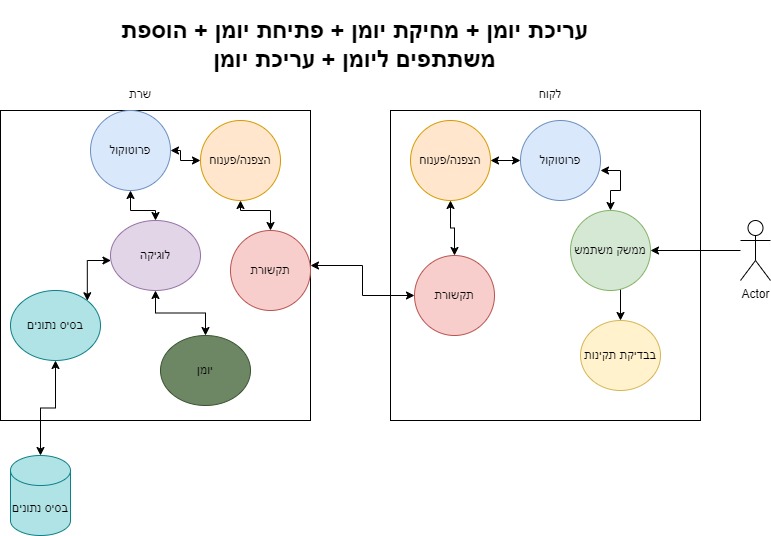
תקשורת: שרת מרובה לקוחות דרך socket

גרפיקה: באמצעות wxPython

תחומי עניין: תקשורת, הצפנה, מערכות הפעלה של אנדרואיד וwindows, בסיס נתונים.

1. תיאור מודולים בהם נעשה שימוש









 \*\* במידה ויהיה לי מספיק זמן אממש



**מודלים שאני מייבאת:**

|  |  |
| --- | --- |
| שם | מטרה |
| Socket | לתקשורת עם סוקטים |
| Queue | לשימוש בתורים |
| Threading | להריץ כמה תהליכונים במקביל |
| Select | לאפשר שרת מרובה לקוחות |
| Hashlib | הצפנת הסיסמאות |
| Datetime | התעסקות עם ימים ושעות וקבלת היום הנוכחי |
| Wx | גרפיקה |
| Pubsub | תקשורת בין גרפיקה ללוגיקה |
| Time | בדיקת תקינות של זמנים ותאריכים |
| Sqlite3 | בסיס נתונים |
| Sys | יציאה מהתוכנית בעת שגיאות |
| Random | לרנדומליות |
| Base64 | להצפנה |
| Cryptodome | להצפנה |

**מודלים שלי:**

| Client\_comm | | |
| --- | --- | --- |
| אחראית על התקשורת של הלקוח עם השרת - תפקידה לקבל הודעות מהשרת ולהעביר הודעות של הלקוח לשרת | | |
| משתנים | | |
| שם המשתנה | **תיאור המשתנה** | |
| Socket | עצם הסוקט של הלקוח שדרכו הוא מתחבר לשרת | |
| Port | הport שצריך להתחבר אליו בשרת | |
| Server\_ip | כתובת הip של השרת | |
| Recv\_q | תור שבו נמצאות ההודעות מהשרת | |
| G | להחלפת מפתחות עם דיפי הלמן | |
| P | להחלפת מפתחות עם דיפי הלמן | |
| personal\_key | המפתח האישי להחלפת מפתחות בדיפי הלמן | |
| Encryption | מפתח הצפנה | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| \_\_init\_\_ | מקבלת server\_ip, port, recv\_q | פעולה בונה שמאתחלת את המשתנים ומתחילה thread בשם \_main\_loopשמתחבר לשרת ומקבל ממנו הודעות |
| \_main\_loop | לא מקבל כלום - Thread שרץ מהרגע שבונים את העצם עד שמתנתקת התקשורת עם השרת | מתחברת לשרת,  מחליפה מפתח באמצעות הפעולה \_set\_key  ואז בלולאה אינסופית : מקבלת ממנו הודעות, מפענחת באמצעות המפתח ושמה אותם בrecv\_q |
| Send | מקבלת msg (הודעה לשלוח לשרת) | מצפינה את ההודעה שולחת אורך הודעה מוצפנת (האורך לא מוצפן) ושולחת את ההודעה לשרת |
| \_set\_key | כלום | מקבל בתקשורת את ה B של השרת, שולח את ה A שלו  מחשב את המפתח הסימטרי המשותף ויוצרת עצם Encryption |

| Server\_comm | | |
| --- | --- | --- |
| אחראית על התקשורת של השרת עם הלקוחות - תפקידה לקבל הודעות מהלקוחות ולהעביר אליהם הודעות | | |
| משתנים | | |
| שם המשתנה | **תיאור המשתנה** | |
| Socket | עצם הסוקט של הלקוח שדרכו הלקוחות מתחברים | |
| Port | הport שהלקוחות צריכים להתחבר אליו | |
| Recv\_q | תור שבו נמצאות ההודעות מהלקוחות | |
| G | להחלפת מפתחות עם דיפי הלמן | |
| P | להחלפת מפתחות עם דיפי הלמן | |
| open\_clients | מילון של הלקוחות הפתוחים של הכתובת ip ושל הsocket ושל עצם ההצפנה האישי של כל לקוח open\_clients[socket]=[ip, encryption] | |
| is\_running | האם השרת רץ | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| \_\_init\_\_ | מקבלת port, recv\_q | פעולה בונה שמאתחלת את המשתנים ומתחילה thread \_main\_loopשמריץ את התקשורת של השרת |
| \_main\_loop | לא מקבל כלום - Thread שרץ מהרגע שבונים את העצם עד שמנתקים את השרת | ממתינה ללקוחות:  לקוח חדש – מפעילה thread להחלפת מפתח set\_key\_  לקוח קיים – מקבלת הודעות, מפענחת עם מפתח אישי ומכניסה לתור recv\_q כ tuple : (ip,msg) |
| Send | מקבלת msg (הודעה לשלוח ללקוח) וip של הלקוח אליו צריך לשלוח | מוצאת את הסוקט של הלקוח  מצפינה את ההודעה שולחת אורך הודעה מוצפנת (האורך לא מוצפן)  ושולחת את ההודעה ללקוח |
| \_set\_key  (מופעלת כ thread) | Client\_socket  Client\_IP | שולחת את A  מקבלת את B  מחשבת מפתח סימטרי ויוצרת עצם Encryption – מוסיפה את הלקוח לרשימת הלקוחות הזמינים  Open\_clients[client\_socket]= [client-IP, Encryption] |
| \_disconnect\_client | מקבל את הip של הלקוח שמתנתק | מוריד את הלקוח מהמילון של הלקוחות |
| \_find\_socket\_by\_ip | מקבל ip | מחזיר את הסוקט של הip |
| sendAll | מקבלת msg לשלוח לכולם | עבור כל לקוח קיים מוצאת את ה IP ומפעילה את send |
| close\_server | כלום | סוגר את השרת ע"י שינוי הis\_running |
| Is\_running | כלום | מחזיר את הis\_running |

| Client\_protocol | | |
| --- | --- | --- |
| פעולות אשר עוטפות את השליחות בפרוטוקול שהשרת יבין ומורידות את העטיפה של הפרוטוקול שהשרת שולח | | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| Unpack | מקבלת הודעה מהשרת msg | מפרקת את ההודעה ומחזירה טאפל של הopcode ורשימה של המשתנים מהשליחה |
| Pack\_login | מקבלת שם משתמש וסיסמא | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_signup | מקבלת שם וסיסמא, ומספר טלפון | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_new\_calendar | מקבלת שם ליומן, רשימת מוזמנים | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_new\_event | מקבלת מזהה יומן, שם אירוע, שעת התחלה, שעת סיום, תאריך, משתתפים | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_info | מקבלת מזהה אירוע ומזהה יומן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_calendar\_invitation | מקבלת מזהה יומן, שם משתמש של המוזמן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_calendar\_response | מקבלת אישור/סירוב, מזהה יומן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_invitation | מקבלת מזהה יומן, מזהה אירוע, שם משתמש של המוזמן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_response | מקבלת אישור/סירוב, מזהה יומן, מזהה אירוע | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_calendar\_name\_edit | מקבלת מזהה יומן ושם חדש | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_name\_edit | מקבלת מזהה אירוע, ושם חדש | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_time\_edit | מקבלת מזהה אירוע, שעת התחלה, שעת סיום, תאריך | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_delete | מקבלת ומזהה אירוע | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_exit\_calendar | מקבלת ומזהה יומן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_calendar\_ids | כלום | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_day\_event | מקבלת מזהה יומן, תאריך | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_month\_events | מזהה יומן, חודש, שנה | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_key | מקבלת מפתח | מחזירה מחרוזת לפי הפרוטוקול |

| Server\_protocol | | |
| --- | --- | --- |
| פעולות אשר עוטפות את השליחות בפרוטוקול שהלקוח יבין ומורידות את העטיפה של הפרוטוקול שהלקוח שולח | | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| Unpack | מקבלת הודעה מהלקוח msg | מפרקת את ההודעה ומחזירה טאפל של הopcode ורשימה של המשתנים מהשליחה |
| Pack\_login | מקבלת 0/1/2 כסטטוס אם הצליח להתחבר | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_signup | מקבלת 0/1 כסטטוס אם הצליח להירשם | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_new\_calendar | מקבלת 0/1 כסטטוס אם הצליח לפתוח את היומן, מזהה יומן / רשימה של משתמשים לא קיימים (תלוי אם הצליח ליצור את היומן או לא) | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_new\_event | מקבלת 0/1 כסטטוס אם הצליח ליצור אירוע, מזהה אירוע | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_info | מזהה אירוע, תאריך, צבע | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_calendar\_invitation | מקבלת שם היומן, מזהה יומן, שם המזמין | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_new\_calendar\_participant | מקבלת סטטוס הצלחה (0/1) מזהה יומן, שם משתמש | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_invitation | מקבלת מזהה יומן, מזהה אירוע, שם אירוע, שם מזמין, זמן התחלה, זמן סיום, תאריך | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_invitation\_succeed | מקבלת סטטוס הצלחה (0/1), שם משתמש, שם אירוע, שם יומן | מחזירה מחרוזת לפי הפרוטוקול |
| pack\_calendar\_name\_edit | מקבלת סטטוס, מזהה יומן ושם חדש | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_name\_edit | מקבלת סטטוס, מזהה יומן, מזהה אירוע, שם חדש | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_time\_edit | מקבלת סטטוס, מזהה יומן, מזהה אירוע, שעת התחלה, שעת סיום, תאריך, רשימה של משתתפים שלא יכולים | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_event\_delete | מקבלת מזהה יומן, מזהה אירוע | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_exit\_calendar | שם משתמש, מזהה יומן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_calendar\_ids | מקבלת רשימה של מזהי היומנים | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_day \_events | מקבלת טאפל של האירועים של המשתמש או המשתתפים ביומן ביום ספציפי | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_month\_events | מקבלת טאפל של האירועים החודשיים ביומן | מחזירה מחרוזת לפי הפרוטוקול |
| Pack\_key | מקבלת מפתח | מחזירה מחרוזת לפי הפרוטוקול |

|  |  |  |
| --- | --- | --- |
| Encryption | | |
| מצפינה ומפענחת מסרים שנשלחים בין השרת ללקוח | | |
| משתנים | | |
| שם המשתנה | **תיאור המשתנה** | |
| Bs | בלוק באורך 16 ביטים של aes לצורך ההצפנה והפענוח | |
| Key | מפתח להצפנה ולפענוח | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| \_\_init\_\_ | מקבלת מפתח | יוצר עצם הצפנה עם התאמת המפתח ויצירת bs |
| Encrypt | מקבל מידע | מצפין את המידע ומחזיר אותו |
| Decrypt | מקבל מידע מוצפן | מפענח את המידע ומחזיר אותו |
| \_pad | כלום | מדפנת את ההודעה ע"פ הנדרש לטובת תהליך ההצפנה |
| \_unpad | כלום | מקזזת את ההודעה כנדרש לאחר תהליך הפענוח |

| Joined\_calendar\_db | | |
| --- | --- | --- |
| מחלקה לעבודה מול בסיס הנתונים של המערכת | | |
| משתנים | | |
| שם המשתנה | **תיאור המשתנה** | |
| Db\_conn | מצביע לבסיס נתונים | |
| Db\_cursor | מצביע לטבלאות שבבסיס נתונים | |
| Name\_of\_db | שם הבסיס נתונים | |
| Calendars | טבלה של היומנים שתכיל את השמות של היומנים, המזהים שלהם והמנהלים שלהם | |
| Users | טבלה של המשתמשים שתכיל את שמות המשתמש שלהם, את מספר הטלפון שלהם ואת הסיסמא המוצפנת שלהם | |
| Calendars\_participants | טבלה של היומנים שתכיל את שמות המשתמש ושל מזהי היומנים ושל הצבע של המשמש ביומן זה | |
| Event\_participants | טבלה של האירועים שתכיל את המשתתפים בהם ואת המזהים שלהם | |
| Event\_info | טבלה של המידע על האירועים שתכיל את מזהי היומן שבו נוצר האירוע, את מזהה האירוע, את השם שלו, המנהל, שעת ההתחלה והסיום והתאריך | |
| Calendar\_invitations | טבלה של ההזמנות ליומנים הפתוחות (שלא ענו עליהן עדיין) שמכילה את שם המשתמש שאליו נשלחה ההזמנה, שם המשתמש של המזמין ומזהה היומן | |
| Event\_invitations | טבלה של ההזמנות לאירועים הפתוחות שמכילה את שם משתמש המוזמן, המזמין, מזהה היומן ומזהה האירוע | |
| Reminders | טבלה של התזכורות שצריכות להישלח שמכילה את שם המשתמש אליו צריכה להישלח התזכורת, השעה, התאריך ומזהה האירוע | |
| Colors | רשימה של צבעים לשמות משתמש ביומנים | |
| Joined\_color | הצבע של האירועים המשותפים לכמה משתתפים | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| \_\_init\_\_ | לא מקבלת כלום | מאתחלת המשתנים ומפעילה את \_create\_DB |
| \_create\_db | לא מקבלת כלום | מתחברת לבסיס הנתונים  ויוצרת את כלל הטבלאות אם לא קיימות |
| Is\_calendar\_exists\_ | מזהה יומן | בודק אם היומן קיים בטבלת היומנים ומחזיר משתנה בוליאני |
| Is\_event\_exists\_ | מזהה אירוע | בודק אם האירוע קיים בטבלת האירועים ומחזיר משתנה בוליאני |
| Is\_user\_exists\_ | שם משתמש | בודק אם המשתמש קיים בטבלת המשתמשים ומחזיר משתנה בוליאני |
| Add\_calendar | פרטי היומן (משתתפים, שם, מנהל) | יוצר את היומן בטבלת הcalendars, ולטבלת הcalendar\_participants ובוחר צבע לכל משתתף בו, שם את מזהה היומן ומחזיר אותו. |
| Add\_calendar\_participant | מזהה היומן ושם משתמש | אם היומן קיים - מוסיף אותו ליומן בטבלת היומנים calendar\_partitcipants ומחליט על צבע בשביל המשתמש |
| Add\_calendar\_invitation | מזהה יומן, שם מזמין, שם מוזמן | אם היומן קיים - מוסיף את ההזמנה לטבלת ההזמנות של היומנים |
| get\_day\_ids | שם משתמש ותאריך | מחזיר את מזהה האירועים של האירועים שיש למשתמש באותו תאריך |
| Add\_event | פרטי האירוע (משתתפים, שם, מזהה היומן בו נוצר, מנהל, שעת התחלה וסיום ותאריך | מוסיף את האירוע לטבלת האירועים event\_info ולטבלת הevent\_participants ויוצר את המזהה של האירוע ומחזיר אותו |
| Add\_event\_participant | מזהה היומן מזהה האירוע ושם משתמש | אם האירוע קיים - מוסיף אותו לאירוע בטבלת האירועים event\_partitcipants |
| get\_calendar\_participants\_plus\_colors | מזהה יומן | מחזיר רשימה של משתתפי היומן עם הצבע שלהם |
| Add\_event\_invitation | מזהה יומן, מזהה אירוע, שם מזמין, שם מוזמן | אם האירוע והיומן קיימים - מוסיף את ההזמנה לטבלת ההזמנות של היומנים |
| is\_event\_invitation\_exists | שם משתמש ומזהה אירוע | בוליאני של האם ההזמנה קיימת |
| Add\_user | שם משתמש, סיסמא מוצפנת ומספר טלפון | מוסיף את המשתמש לטבלת המשתמשים אם לא קיים משתמש כזה ומוסיף לטבלת היומנים יומן אישי בשבילו. מחזיר את מזהה היומן או -1 |
| Get\_password | מקבל שם משתמש | אם המשתמש קיים - מחזיר את הסיסמא שלו, -1 אם המשתמש לא קיים |
| is\_participants\_in\_calendar | מקבל שם משתמש ומזהה יומן | מחזיר בוליאני של האם המשתמש ביומן |
| check\_is\_time\_available | שם משתמש, שעת התחלה, שעת סיום, תאריך | בוליאני של האם המשתמש פנוי בשעות |
| Change\_calendar\_name | מזהה יומן ושם חדש | אם היומן קיים - משנה את השם של היומן בטבלה calendars |
| Change\_event\_name | מקבל שם חדש | אם האירוע קיים - משנה את השם של האירוע בטבלה של event\_info |
| events\_in\_time | שם משתמש, שעת התחלה, שעת סיום, תאריך | מחזיר רשימת אירועים שבזמן הנתון |
| Change\_event\_time | מקבל מזהה אירוע, שעת התחלה, סיום ותאריך | אם האירוע קיים - משנה את שעת ההתחלה הסיום והתאריך הטבלה של event\_info |
| get\_today\_reminders | כלום | מחזיר רשימה של התזכורות של האירועים של היום |
| Add\_reminder | מקבל שעה, תאריך, מזהה אירוע ושם משתמש אליו התזכורת צריכה להישלח | מוסיף תזכורת לטבלת התזכורות reminders |
| get\_personal\_calendar | שם משתמש | מחזיר מזהה יומן אישי שלו |
| get\_today\_reminders | כלום | עובר על טבלת התזכורות reminders ומחזיר רשימה של כל התזכורות של האירועים באותו היום ולאחר מכן מוחק את התזכורת מהטבלה |
| Delete\_reminder\_for\_user | מזהה אירוע, שם משתמש | מוחק תזכורת של אדם אחד |
| Delete\_reminder\_for\_all | מזהה יומן | מוחק את כל התזכורות של אירוע מסוים |
| Get\_calendar\_participants | מזהה יומן | מחזיר את משתתפי היומן |
| Get\_event\_participants | מזהה אירוע | מחזיר את משתתפי האירוע |
| exit\_calendar | מזהה יומן, שם משתמש של האדם שביקש לצאת | אם היומן והמשתמש קיימים - מוחק את המשתמש ואת אירועיו מהיומן מהבסיס נתונים אם המשתמש שביקש למחוק הוא המנהל אז מוחק את היומן לגמרי מהבסיס נתונים |
| Delete\_event | מזהה אירוע, שם המשתמש של האדם שביקש למחוק | אם האירוע והמשתמש קיימים - מוחק את האירוע מבסיס הנתונים אם המשתמש הוא מנהל האירוע |
| Get\_calendar\_manager | מזהה היומן | אם היומן קיים - מחזיר את המנהל שלו |
| Get\_event\_manager | מזהה האירוע | אם האירוע קיים - מחזיר את המנהל שלו |
| find\_color | מזהה יומן | אם היומן קיים - שם למשתמש צבע שלא קיים אצל המשתתפים האחרים ביומן |
| get\_day\_color | מזהה יומן ותאריך | מחזיר את צבע היום |
| Get\_calendar\_invitations | שם משתמש | אם המשתמש קיים - מחזירה את ההזמנות ליומנים של המשתמש |
| get\_calendar\_info\_with\_colors | מזהה יומן | מחזיר משתתפי היומן ביחד עם הצבע שלהם, את השם של היומן, המנהל שלו |
| Get\_event\_invitations | שם משתמש | אם המשתמש קיים - מחזירה את ההזמנות לאירועים של המשתמש |
| Delete\_event\_invitation | שם משתמש, מזהה אירוע | מוחק את ההזמנה |
| Delete\_calendar\_invitation | שם משתמש, מזהה יומן | מוחק את ההזמנה |
| Get\_user\_calendars | שם משתמש | אם המשתמש קיים - מחזירה את מזהי היומנים שהמשתמש משתתף בהם |
| get\_event\_date | מזהה אירוע | מחזיר את התאריך שלו |
| get\_color | מזהה יומן ושם משתמש | מחזיר את צבע המשתמש ביומן |
| get\_some\_event\_info | מזהה יומן ומזהה אירוע | מחזיר צבע אירוע ותאריך |
| get\_calendar\_from\_event | מזהה אירוע | מחזיר את מזהה היומן שבו נוצר האירוע |
| Get\_events\_of\_calendar | מזהה יומן | אם היומן קיים - מחזירה את התאריך של האירועים שחברי היומן משתתפים בהם, את צבע האירוע (לפי המשתתפים) ואת מזהה האירועים של כל יום |
| Get\_day\_events | שם משתמש, תאריך, מזהה יומן | אם המשתמש קיים - מחזירה את האירועים של היומן באותו יום עם פרטיהם (תלוי אם המשתמש משתתף באירוע או לא - שם, שעת התחלה וסיום, מנהל, תאריך, משתתפים או שעת התחלה וסיום, תאריך, משתתפים) |
| Get\_calendar\_info | מזהה יומן | אם היומן קיים - מחזירה את פרטי היומן (שם היומן ומשתתפיו) |
| Get\_event\_info | מזהה אירוע, שם משתמש | אם האירוע קיים - מחזירה את פרטי האירוע (משתתפיו, שמו, מנהל, שעת התחלה וסיום, תאריך או שעת התחלה וסיום, תאריך, משתתפים. תלוי אם המשתמש המבקש את הפרטים משתתף באירוע) |
| Get\_some\_event \_info | מזהה אירוע, מזהה יומן | אם האירוע קיים - מחזירה את צבע האירוע (לפי המשתתפים), והתאריך |
| Check\_if\_time\_available | שם משתמש, שעת התחלה, שעת סיום, תאריך | אם המשתמש קיים - מחזירה האם הזמן פנוי למשתמש |
| Is\_participant\_exists\_in\_calendar | מזהה יומן ושם משתמש | בודק אם המשתמש משתתף ביומן |
| Is\_calendar\_invitation\_exists | מזהה יומן ושם משתמש | בודק אם ההזמנה קיימת |
| Is\_participant\_exists\_in\_event | מזהה אירוע ושם משתמש | בודק אם המשתמש משתתף באירוע |
| Is\_manager\_calendar | מזהה יומן ושם משתמש | בודק אם המשתמש מנהל היומן |
| Is\_manager\_event | מזהה אירוע ושם משתמש | בודק אם המשתמש מנהל האירוע |

| Client\_logic | | |
| --- | --- | --- |
| אחראית על הלוגיקה של המשתמש | | |
| משתנים | | |
| שם המשתנה | **תיאור המשתנה** | |
| Comm | עצם של המחלקה client\_comm שאחראי על התקשורת עם השרת | |
| Opcodes | מילון של הopcodes הקיימים עם הפעולות שלהם | |
| Month\_event | מילון של אירועי החודש המוצג אצל המשתמש. של מזהי האירועים, תאריכיהם וצבעיהם | |
| Current\_calendar | מזהה היומן, משתתפים, שם, מנהל של היומן הפתוח כרגע אצל הלקוח | |
| Current\_month | חודש ושנה שפתוחים כרגע אצל המשתמש | |
| Msg\_q | תור של הודעות מהשרת | |
| Graphics\_q | תור של הודעות מהגרפיקה | |
| Current\_day | יום שפתוח אצל המשתמש | |
| Current\_event | אינדקס של האירוע הנוכחי שהמשתמש מסתכל עליו ברשימות האירועים | |
| Current\_invitation | אינדקס של ההזמנה הנוכחית שהמשתמש מסתכל עליה ברשימה של ההזמנות | |
| Invitations | הזמנות לאירועים ויומנים של הלקוח | |
| Day\_events | מזהה האירועים של אותו יום | |
| App | עצם גרפיקה | |
| Frame | יוצר פריים של גרפיקה | |
| User\_calendars | רשימה של מזהה היומנים של המשתמש (מאותחלת ברשימה ריקה) | |
| פעולות | | |
| שם הפעולה | **טענת כניסה** | **טענת יציאה** |
| \_\_main\_\_ | כלום | יוצרת תור להודעות נכנסות  יוצרת עצם תקשורת  מפעילה thread – handle\_msgs לטיפול בהודעות נכנסות  יוצרת עצם גרפיקה  מפעילה את ממשק המשתמש |
| Handle\_recv | תור הודעות שהתקבלו מהשרת | Thread שכל פעם מטפל בהודעה הראשונה בתור. הוא מפרק את ההודעה לפי הפרוטוקול (למשתנים וopcode) ומפעיל את הפעולה המתאימה שמטפלת בopcode שהתקבל |
| Handle\_graphics | תור הודעות מהגרפיקה | Thread שכל פעם מטפל בהודעה הראשונה בתור. לפי הopcode של ההודעה מטפל ומגיב בהתאם |
| Login | שם משתמש וסיסמא שהוכנסו כקלט מהלקוח | בודק תקינות, ואם תקין שולח לשרת לבדוק האם המשתמש קיים |
| Handle\_login | קבלת סטטוס אם הצליח ואם כן אז גם את מזהה היומן האישי | אם הצליח מראה את היומן האישי אצל המשתמש אם לא מציג למה לא הצליח למשתמש |
| Sign\_up | שם משתמש, מספר טלפון וסיסמא שהוכנסו כקלט מהלקוח | בודק תקינות, ואם תקין שולח לשרת לבדוק האם אפשרי |
| Handle\_signup | קבלת סטטוס אם הצליח ואם כן אז גם את מזהה היומן האישי | אם הצליח מראה את היומן האישי אצל המשתמש אם לא מציג למה לא הצליח למשתמש |
| New\_calendar | שם יומן ומשתתפים | בודק תקינות ואם תקין שולח לשרת |
| Call\_error | הודעת השגיאה | פותח שגיאה בגרפיקה |
| Handle\_new\_calendar | רשימת המשתנים מהשרת שהclient\_protocol פירק | מוסיף את מזהה היומן שלו לרשימת מזהה היומנים של המשתמש ומציג את היומן |
| New\_event | שם, משתתפים, שעת התחלה וסיום, תאריך | בודק תקינות ואם תקין שולח לשרת |
| Handle\_new\_event | רשימת המשתנים מהשרת שהclient\_protocol פירק | אם החזיר סטטוס הצלחה אז קורא לקבלת תאריך האירוע וצבעו ומוסיף נקודה בתאריך האירוע ביומן שמוצג אצל המשתמש באותו הרגע |
| handle\_event\_info | רשימת המשתנים מהשרת שהclient\_protocol פירק | מחזירה את צבע האירוע (צבע המשתמש שהאירוע שלו אם זה משתמש יחיד ואם זה אירוע משותף אז הצבע המשותף, תאריך. קוראים לה מתוך פעולות אחרות שמשתמשות בנתונים שמחזירה בהתאם לצורך. |
| invite\_to\_event | שם משתמש, מזהה יומן ומזהה אירוע | שולח לשרת |
| response\_event\_invitation | סטטוס ומזהה אירוע | שולח לשרת |
| handle\_is\_event\_invitation\_work | רשימת משתנים מהמעבדה | מראה הודעה בהתאם |
| handle\_there\_is\_an\_invitation | רשימת משתנים מהשרת | מדליק כפתור של הזמנות בגרפיקה |
| handle\_is\_calendar\_invitation\_work | רשימת משתנים מהשרת | מציג הודעה בהתאם לסטטוס |
| invite\_to\_calendar | מזהה יומן ושם משתמש | בודק תקינות ואם תקין שולח לשרת |
| response\_to\_calendar\_invitation | סטטוס ומזהה יומן | שולח לשרת |
| Handle\_calendar\_invitations | רשימת המשתנים מהשרת שהclient\_protocol פירק | מוסיפה ללקוח את ההזמנה לרשימת ההזמנות ליומנים ומסמנת לו שיש הזמנה חדשה |
| Handle\_add\_participants\_to\_calendar | סטטוס הצלחה, שם משתמש, מזהה יומן | אם סטטוס הצלחה 0 מוסיפה את המשתמש ליומן אם מופיע כרגע אצל המשתמש בתצוגה  אם 1 אז מעדכנת את המשתמש שהוא לא המנהל של היומן אז לא יכול להוסיף משתתפים |
| Handle\_event\_invitations | רשימת המשתנים מהשרת שהclient\_protocol פירק | מוסיפה ללקוח את ההזמנה לרשימת ההזמנות לאירועים ומסמנת לו שיש הזמנה חדשה |
| Handle\_response\_event\_invitation | סטטוס הצלחה, שם האירוע ושם היומן | אם סטטוס הצלחה 1 אז מעדכנת את המשתמש שהוא לא המנהל של האירוע לכן לא יכול להזמין אנשים |
| change\_name\_of\_calendar | מזהה יומן ושם חדש | בודק תקינות ואם תקין שולח לשרת |
| Handle\_calendar\_name\_change | רשימת המשתנים מהשרת שהclient\_protocol פירק | אם סטטוס הצלחה 0 משנה את השם של היומן אם הוא פתוח כרגע אצל המשתמש  אם 1 מעדכנת את המשתמש שלא יכול לשנות כיוון שלא מנהל |
| change\_name\_of\_event | מזהה אירוע ושם חדש | בודק תקינות ואם תקין שולח לשרת |
| Handle\_event\_name\_change | רשימת המשתנים מהשרת שהclient\_protocol פירק | אם סטטוס הצלחה 0 משנה את שם האירוע אצל האדם ששינה אותו  אם 1 מעדכנת את המשתמש שלא יכול לשנות כיוון שלא מנהל |
| change\_time | מזהה אירוע שעת התחלה שעת סיום ותאריך | בודק תקינות ואם תקין שולח לשרת |
| Exit\_calendar | מזהה יומן | בודק אם היומן הנוכחי הוא לא האישי ואם לא אז שולח לשרת |
| delete\_event | מזהה אירוע | שולח לשרת |
| Handle\_event\_time\_change | רשימת המשתנים מהשרת שהclient\_protocol פירק | משנה את זמן האירוע אצל מי שאירועי היום של האירוע (החדש או הישן) פתוחים אצלו ואם שינה את היום מוחק/מוסיף בהתאם את האירוע להצגת האירועים היומיים. אצל מי שהחודש הנוכחי והיומן הנוכחי זהה לתאריך החדש/הישן של האירוע אז מוחק/מוסף למסך נקודה בהתאם ומשנה ברשימת אירועי החודש את התאריך |
| Handle\_delete\_event | רשימת המשתנים מהשרת שהclient\_protocol פירק | מוחק את האירוע מהמשתמשים שפתוחים אצלם היומן של האירוע בחודש המתאים באירועי החודש ומהמסך ובמשתמשים שפתוחים אצלם היום המתאים ברשימת אירועי היום |
| Handle\_exit\_calendar | רשימת המשתנים מהשרת שהclient\_protocol פירק | אצל המשמש שיצא - מוחק את היומן ממזהה היומנים של המשתמש ועובר ליומן הקודם במסך  אצל משתתפי היומן שהיומן פתוח אצלם בתצוגה באותו רגע – מסירה את המשתמש ממשתתפי היומן ומוחקת אותו מהאירועים המשותפים ומוחקת את אירועיו האישיים (ומסדרת את הגרפיקה בהתאם) |
| get\_calendar\_ids | כלום | שולח לשרת בקשה לקבל את המזהים |
| Handle \_calendar\_ids | רשימת המשתנים מהשרת שהclient\_protocol פירק | שומר את רשימת מזהי היומנים אצל המשתמש |
| get\_day\_events | מזהה יומן ותאריך | בודק תקינות ואם תקין שולח לשרת |
| Handle\_day\_events | רשימת המשתנים מהשרת שהclient\_protocol פירק | שומרת את הרשימה של האירועים היומיים אצל הלקוח ומציגה את האירוע הראשון בה על המסך |
| get\_month\_events | מזהה יומן, חודש, שנה | בודק תקינות ואם תקין שולח לשרת |
| Handle\_month\_events | רשימת המשתנים מהשרת שהclient\_protocol פירק | שומרת את הרשימה של האירועים (תאריך וצבע) שבחודש המבוקש אצל הלקוח ומציגה אותה במסך (את החודש והנקודות בצבעים ובתאריכים המתאימים) |
| get\_calendar\_info | מזהה יומן | שולח בקשה לקבל את המידע מהשרת |
| get\_invitations | כלום | שולח לשרת בקשה לקבל הזמנות מהשרת |
| handle\_invitations | מקבל הזמנות מהשרת | מציג על המסך ושומר במשתנה invitations |

| Server\_logic | | |
| --- | --- | --- |
| אחראית על הלוגיקה של השרת | | |
| משתנים | | |
| שם המשתנה | **תיאור המשתנה** | |
| Comm | עצם של server\_comm של תקשורת | |
| msg\_q | תור של הודעות שהשרת מקבל מהלקוחות | |
| Db | עצם של המחלקה של בסיס הנתונים | |
| Opcodes | מילון של הopcodes הקיימים עם הפעולות שלהם | |
| Current\_open\_calendars | מילון של שם משתמש ורשימה של מזהה היומן שפתוח אצלו באותו הרגע, חודש ושנה ויום | |
| Current\_users | מילון של ip שכרגע פתוח ושם המשתמש שמחובר דרכו | |
| פעולות | | |
| \_\_main\_\_ | כלום | יוצרת תור להודעות נכנסות  יוצרת עצם תקשורת  מפעילה thread – handle\_msgs לטיפול בהודעות נכנסות |
| Handle\_msgs | תור הודעות שהתקבלו מהלקוחות | Thread שכל פעם מטפל בהודעה הראשונה בתור. הוא מפרק את ההודעה לפי הפרוטוקול (למשתנים וopcode) ומפעיל את הפעולה המתאימה שמטפלת בopcode שהתקבל |
| Handle\_login | Ip, שם משתמש וסיסמא | בודק אם השם משתמש קיים בבסיס הנתונים בטבלה users ואם כן אז אם הסיסמא שהתקבלה זהה לסיסמא בבסיס הנתונים ומחזיר הודעה עם סטטוס בהתאם (הצלחה, שם משתמש לא קיים, סיסמא לא נכונה, כבר מחובר ממחשב אחר). אם הצליח להתחבר אז שולח ללקוח את מזהה היומנים שלו. בנוסף שולח את ההזמנות הפתוחות של המשתמש (ליומנים ולאירועים) |
| Handle\_signup | Ip, שם משתמש וסיסמא | בודק אם השם משתמש לא קיים בבסיס הנתונים. אם לא אז שם את השם משתמש בבסיס נתונים של users בונה לו יומן אישי ושולח לו אותו. בנוסף הוא שולח סטטוס הצלחה (אם הצליח להירשם או השם משתמש תפוס) |
| Handle\_new\_calendar | Ip, שם היומן, רשימת מוזמנים | יוצר את היומן בטבלת הcalendars כאשר המנהל הוא האדם שיצר את היומן. מחשב את מזהה היומן, בודק האם המשתמשים שברשימת המוזמנים קיימים ואם כן מוסיף לטבלת ההזמנות ליומנים את ההזמנות לכל משתמש ברשימה. שולח ללקוח הודעה עם סטטוס בהתאם (הצליח/לא הצליח) ואם לא הצליח כי יש משתמשים שלא קיימים אז שולח גם את השמות שלהם ברשימה. בנוסף שולח את מזהה היומן. |
| Handle\_new\_event | Ip, מזהה יומן, שם אירוע, שעת התחלה וסיום, תאריך, משתתפים | בודק האם הזמן פנוי, אם לא אז מחזיר סטטוס שלא פנוי ואם כן אז מחזיר שכן ואת מזהה האירוע ויוצר את האירוע בטבלת האירועים. בנוסף מוסיף שהזמן תפוס ביומנים האחרים של המשתמש (שולח למשתמשים שבאותו רגע ביומן שמשותף עם המשתמש בחודש זהה ומוסיף לטבלאות המתאימות).  מוסיף בטבלת ההזמנות לאירועים את ההזמנות למשתמשים שברשימה. |
| get\_event\_info | מזהה אירוע, מזהה יומן | בודק בטבלת האירועים. אם האירוע לא קיים מחזיר רשימה ריקה ואם כן אז מחזיר את התאריך, צבע האירוע ומזהה האירוע (את צבע האירוע מחזיר לפי הצבע של המשתתף שהאירוע שלו ביומן ואם האירוע של כמה אנשים ביומן אז משתמש בצבע המשותף) |
| Handle\_calendar\_invitation | Ip, מזהה יומן, שם מוזמן | אם מי שמבקש להוסיף משתמש הוא מנהל היומן ואם היומן קיים, מוסיף לטבלת ההזמנות ליומנים את ההזמנה ואם המשתמש מחובר באותו רגע אז שולחת לו את ההזמנה |
| Handle\_is\_calendar\_accepted | Ip, סטטוס אישור או סירוב ומזהה יומן | אם הסטטוס הוא אישור אז שולח למשתמש את היומן החדש. בנוסף מוסיף את המשתמש ליומן בטבלה calendar\_participants ושולח למשתתפי היומן המחוברים ליומן באותו הרגע שהמשתמש התווסף.  מוריד את הזמנה מהטבלה |
| Handle\_event\_invitation | Ip, מזהה יומן, מזהה אירוע, שם משתמש של המוזמן | אם מי שמבקש להוסיף משתמש הוא מנהל האירוע והאירוע והיומן קיימים אז מוסיף הזמנה לטבלת ההזמנות לאירועים ואם המשתמש מחובר ברגע זה למערכת אז גם שולחת לו את ההזמנה |
| Handle\_event\_is\_accepted | Ip, סטטוס אישור או סירוב, מזהה יומן, מזהה אירוע | אם הסטטוס אישור אז שולח למשתמש את האירוע החדש. בנוסף מוסיף את המשתמש לאירוע בטבלת event\_participants |
| handle\_change\_name\_of\_calendar | Ip, מזהה יומן, שם חדש | אם המשתמש שמבקש לשנות את השם הוא המנהל משנה את השם של היומן בטבלת היומנים ושולחת לאנשים שהיומן פתוח אצלם באותו הרגע את השינוי, אם לא שולחת לו שלא יכול כיוון שלא מנהל |
| handle\_change\_name\_of\_event | Ip, מזהה אירוע, שם חדש | אם המשתמש שמבקש לשנות את השם הוא מנהל האירוע אז משנה את שם האירוע בטבלת האירועים, אם לא שולחת לו שלא יכול כיוון שלא מנהל |
| Handle\_time\_change | Ip, מזהה יומן, מזהה אירוע, שעת התחלה וסיום, תאריך | אם הזמן פנוי אז משנה את זמני האירוע בטבלת event\_info ושולח סטטוס הצלחה / לא עם פרטי האירוע החדש (מתאפשר רק לאירוע אישי!).  אם התאריך שונה אז שולח לאנשים שבאותו רגע פתוח אצלם יומן שאחד ממשתתפיו הוא הלקוח ששינה את הזמן בתאריך המתאים (לתאריך החדש/ישן) |
| Handle\_delete\_event | Ip, מזהה יומן, מזהה אירוע, שם משתמש | אם המשתמש הוא מנהל האירוע אז מוחק את האירוע מטבלת האירועים ומטבלת הevent\_info ושולח לאנשים שכרגע יומן עם אחד ממשתמשי האירוע בתצוגה אצלם בתאריך האירוע שהאירוע נמחק |
| handle\_get\_invitations | Ip | מחזיר למשתמש רשימה של ההזמנות שלו לאירועים וליומנים (לפי הבסיס נתונים) |
| Handle\_calendar\_exit | Ip, מזהה יומן, שם משתמש | מוחק את המשתמש מהיומן בטבלת calendar\_participants ומוחק אותו מהאירועים המשותפים. אצל האנשים שפתוח אצלם היומן בתצוגה אז מוחק את המשתמש ושולח להם את פרטי האירוע (צבע חדש ומזהה האירוע) של האירועים שהמשתמש יצא מהם. |
| Handle\_calendar\_ids | Ip | מחזיר את רשימת מזהה היומנים למשתמש |
| Handle\_day\_events | Ip, מזהה יומן, תאריך | מחזירה למשתמש טאפלים של אירועי היום של המשתמש ושל האירועים של המשתתפים ביומן. (אם האירוע לא שלו אז מחזירה פרטיים מצומצמים ואם כן שלו אז את כל הפרטים) |
| Handle\_month\_events | Ip, מזהה יומן, חודש, שנה | מחזירה טאפל של כל האירועים הקיימים ביומן בחודש זה עם מידע מצומצם (תאריך, מזהה אירוע, צבע אשר נקבע לפי צבע המשתמש אם האירוע של משתמש אחד ביומן והצבע השיתופי אם האירוע של כמה אנשים ביומן) |
| handle\_get\_calendar\_info | Ip, מזהה יומן | מחזירה מידע על היומן (משתתפים, שם, מנהל) ושולחת ללקוח ושמה במשתנה current\_open\_calendars שזה היומן הפתוח אצל המשתמש |
| disconnect\_client | Ip | מוחק את הלקוח מהמילון של היומנים הפתוחים ומהמילון של המשתמשים המחוברים |

| MyFrame |
| --- |
| הבסיס עליו עובד כל הגרפיקה |

| MainPanel |
| --- |
| הפאנל הראשי עליו כל הגרפיקה משתנה |

| LoginPanel |
| --- |
| פאנל להתחברות למערכת |

| RegistrationPanel |
| --- |
| פאנל להרשמה למערכת |

| FirstPanel |
| --- |
| הפאנל הראשון שרואים כאשר מריצים את התכונית, מפנה להרשמה ולהתחברות |

| CalendarPanel |
| --- |
| פאנל להצגת היומן |

| ParticipantsPanel |
| --- |
| פאנל להצגת המשתתפים בגלילה מטה, שיושב על פאנל היומן |

| EventPanel |
| --- |
| פאנל להצגת אירוע |

| newCalendarPanel |
| --- |
| פאנל לפתיחת יומן חדש |

| newCalendarParticipantPanel |
| --- |
| פאנל להוספת משתתף ליומן |

| invitationsPanel |
| --- |
| פאנל להצגת ההזמנות ליומנים ולאירועים |

| newEventParticipantPanel |
| --- |
| פאנל להוספת משתתפים לאירוע |

| newEventPanel |
| --- |
| פאנל לפתיחת אירוע חדש |

| calNamePanel |
| --- |
| פאנל לשינוי שם יומן |

| evtNamePanel |
| --- |
| פאנל לשינוי שם אירוע |

| evtTimePanel |
| --- |
| פאנל לשינוי זמן אירוע |

1. תיאור סביבת הפיתוח

שפת התכנות שנבחרה לפרויקט: python לשרת והלקוח, sql בבסיס נתונים, java באנדרואיד

פירוט כלי הפיתוח הנדרשים לפיתוח והסביבה והכלים הנדרשים לבדיקות:

* Pycharm – בניית המחלקות ופיתוח הפרויקט.
* wireshark - מעקב אחרי התקשורת בין השרת ללקוחות
* android studio – שליחת התזכורות למכשיר האנדרואיד
* db browser for SQLite – עבודה עם הבסיס נתונים
* task manager – למעקב אחר ה thread-ים וביצועי התוכנית

1. תיאור האלגוריתמים המרכזיים בפרויקט

* **הצפנה –** הצפנה מאוד חשובה בפרויקט מכיוון שאנו לא רוצים שאדם יוכל לגלות פרטים ואירועים של משתמש אחר כיוון שהם יכולים להיות פרטים אישיים. בנוסף אנו לא רוצים שאדם יוכל לשנות את האירועים של אדם אחר. המטרה היא שאירועים של אדם יישארו אישיים ופרטיהם יהיו בשליטתו. בעזרת ההצפנה המידע יעבור בצורה מאובטחת אז לman in the middle יהיה קשה לגשת ולהבין את המידע.

| חלופות | יתרונות | חסרונות |
| --- | --- | --- |
| הצפנה סימטרית | קל למימוש ומהיר | צריך להעביר את המפתח ברשת בצורה לא מאובטחת |
| הצפנה א-סימטרית | לא מעבירים מפתח ברשת בצורה לא מאובטחת | כבדה ואיטית |
| שילוב של החלפת מפתח בדיפי הלמן ולאחר מכן שימוש בהצפנה סימטרית | קל למימוש ומהיר, לא מעבירים מפתח ברשת בצורה לא מאובטחת | מצריך אלגוריתם ליצירת המפתח המשותף |

הפיתרון הנבחר – חלופה 3: שילוב של החלפת מפתח בדיפי הלמן ולאחר מכן שימוש בהצפנה סימטרית. אני חושבת שפיתרון זה משלב הכי יתרונות של הפתרונות האחרים והחסרונות שלו הכי קטנים. מצד אחד לא מעבירים מפתח בצורה גלויה כמו בהצפנה הסימטרית אך גם ההצפנה קלה למימוש ומהירה כמו בהצפנה הא-סימטרית.

* **הצגת היומן המשותף –** אופן הצגת היומן המשותף מאוד חשובה והמטרה היא שתהיה הכי נוחה למשתמש. צריך להציג את האירועים המשותפים ואת הזמנים הפנויים והתפוסים של כל משתמש אך בלי לחשוף את פרטי האירועים האישיים כיוון שיכולים להיות אישיים.

| חלופות | יתרונות | חסרונות |
| --- | --- | --- |
| הצגת היומנים הפרטיים של כל המשתתפים אחד ליד השני | קל למימוש  קל להבחין של מי כל אירוע  מציג את כל המידע | מצריך יותר עבודה למשתמש בהשוואת היומנים ויכול להיות בזה המון טעויות.  חושף את הפרטים של היומנים של כל משתמש שיכולים להיות אישיים.  מעמיס את התצוגה |
| שילוב היומנים כאשר מציגים את כל הפרטים של היומנים האישיים של המשתתפים בתוך יומן אחד | מציג את כל המידע | חושף את הפרטים של היומנים של כל משתמש שיכולים להיות אישיים.  היומן השיתופי יהיה עמוס ולא נוח למשתמש.  מעמיס את התצוגה |
| שילוב היומנים ללא חשיפה של פרטי האירועים האישיים של כל משתמש, אלא רק כתיבה שהמשתמש לא פנוי בזמן האירוע | נוח למשתמש ולא דורש ממנו השוואה  לא חושף את הפרטים האישיים של המשתמשים  פחות עומס על התצוגה | שאר המשתמשים לא יכולים לדעת את דחיפות האירוע הפרטי של משתמש או אם עדיין מתקיים, ולכן יצטרכו לדבר איתו ישירות כדי לבדוק זאת ולא להסתמך על מה שרואים ביומן. |

הפיתרון הנבחר – חלופה 3: שילוב היומנים ללא חשיפה של פרטי האירועים האישיים של כל משתמש, אלא רק כתיבה שהמשתמש לא פנוי בזמן האירוע. בחרתי בפיתרון זה מכיוון שאופן הצגת היומן חשוב מאוד שיהיה נוח למשתמש ושהתצוגה תהיה לא עמוסה וקלה להבנה. בנוסף חשוב שפרטי האירועים האישיים של כל משתמש יישארו אישיים כדי לא לחשוף פרטים לא רצויים.

* **הצגת אירועי היום –** אופן הצגת אירועי היום הבודד מאוד חשובה כיוון שצריכה להיות מובנת למשתמש ולהציג לו את הפרטים החשובים.

| חלופות | יתרונות | חסרונות |
| --- | --- | --- |
| הצגת אירועי היום הפרטיים, המשותפים והזמנים התפוסים של משתתפים אחרים בעמוד אחד | מראה את כל המידע על המסך בבת אחת | יהיה עומס בתצוגה  בלאגן  יכול להיות מצב של חפיפה בין כמה אנשים |
| הצגת כל פעם אירוע אחד עם כל הפרטים ואפשרות דפדוף ביניהם | פחות עומס בתצוגה  לא מבולגן, נגיש  נוח למשתמש לראות את הפרטים של כל אירוע וכך יפספס פחות דברים חשובים | מצריך דפדוף בין האירועים ולא מראה את כולם על המסך בבת אחת |

הפיתרון הנבחר – חלופה 2: הצגת כל פעם אירוע אחד עם כל הפרטים ואפשרות דפדוף ביניהם. בחרתי בפיתרון זה מכיוון שאני חושבת שחשוב שלא יהיה עומס בתצוגה כדי שלמשתמש יהיה נוח להבין את המידע. בנוסף בחלופה הראשונה יכול להיות חפיפה בין כמה אנשים וזה יכול לגרום לכך שהמשתמש לא ישים לב לחלק מהפרטים והאנשים שלא פנויים בזמן הנתון.

* **שליחת תזכורת –** בפרויקט צריך לשלוח תזכורות על אירועים למשתמש (בזמן לפי בחירתו), כדי לעשות את חווית המשתמש יותר טובה ולהזכיר לו על האירועים שהוא קובע. המטרה היא שהשליחה תהיה הכי יעילה והכי נוחה למשתמש. \*\* במידה ויהיה לי מספיק זמן אממש

| חלופות | יתרונות | חסרונות |
| --- | --- | --- |
| שליחת תזכורת באמצעות מייל | יעיל מאוד וקל למימוש. לא מצריך שימוש בשפה שונה מpython | פחות נוח למשתמש – המשתמש צריך להיכנס למייל בשביל לראות את ההתראה במקום ש"תקפוץ" לו למסך |
| שליחת תזכורת דרך sms | נוח למשתמש | יותר קשה למימוש ומצריך למידה של אנדרואיד |

הפיתרון הנבחר – חלופה 2: שליחת תזכורות דרך sms. היתרונות עולים על החסרונות. אני חושבת שחווית המשתמש בנושא זה חשובה מאוד כיוון שחשוב להזכיר למשתמש על האירועים שלא ישכח מהם, כי המטרה של היומן השיתופי הוא שהאירועים שקובעים בו יצליחו לקרות.

1. תיאור מסכי הפרויקט:



התחברות

הרשמה

יומן משותף

יומן אישי

הוספת יומן

הזמנות ליומנים

הצגת פרטי יום

הזמנות לאירועים

הוספת אירוע

**תכנון המסכים:**



התחברות משתמש קיים



הרשמת משתמש חדש



ריכוז והצגת כל אירועי המשתמש



לאשר / לסרב להזמנות ליומנים משותפים



ריכוז והצגת כל האירועים של אותו יום



הצגת היומן המשותף והאירועים המשותפים + הזמנים של האירועים הפרטיים של המשתתפים (כדי לראות מתי יש זמן פנוי משותף)



הוספת אירוע פרטי / משותף של חברי יומן

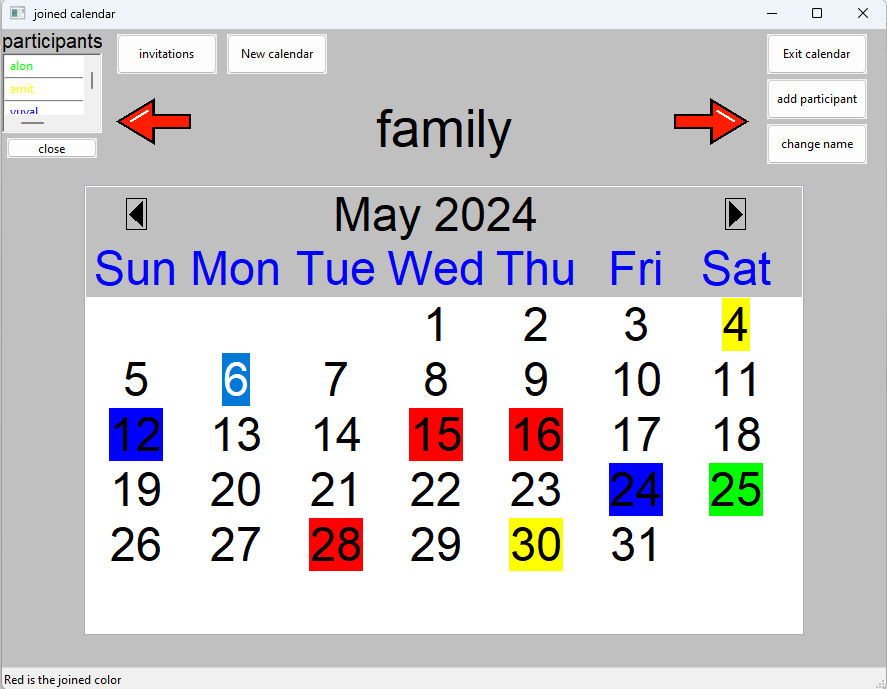


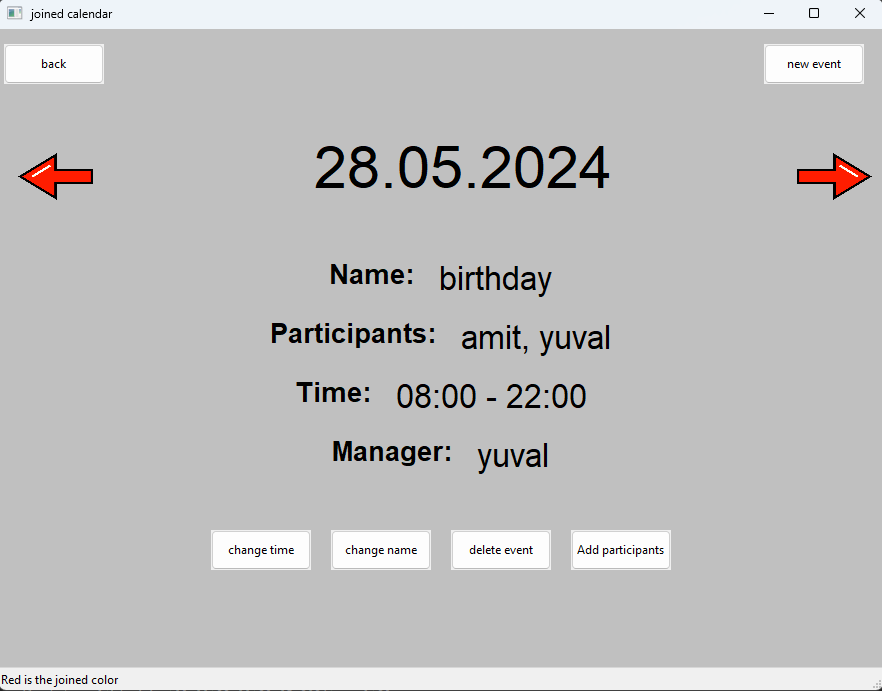
אפשרות להוסיף יומן עם משתתפים



לאשר / לסרב לאירועים שמשתתפי היומנים המשותפים של המשתמש הזמינו אותו

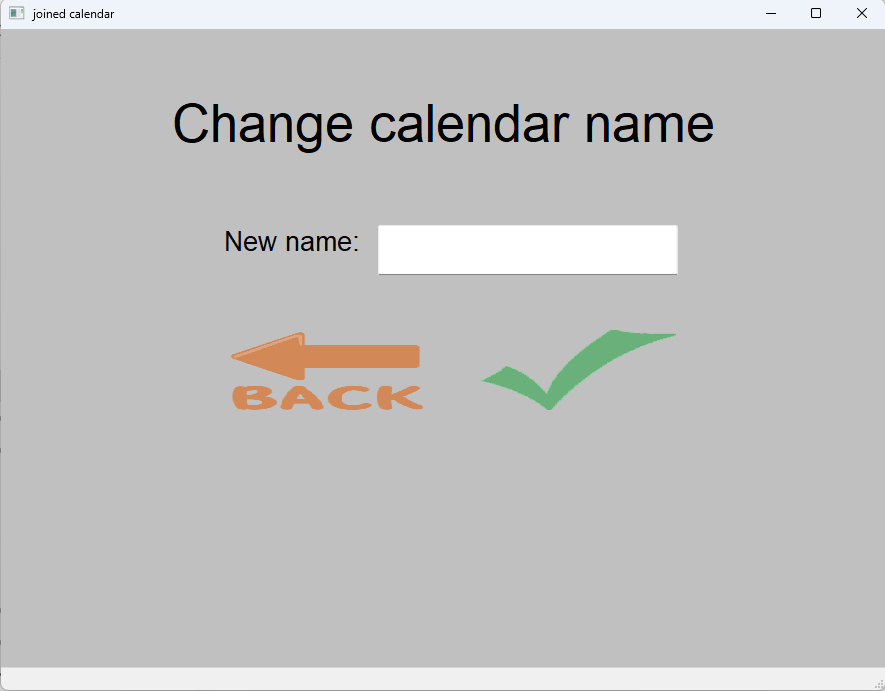
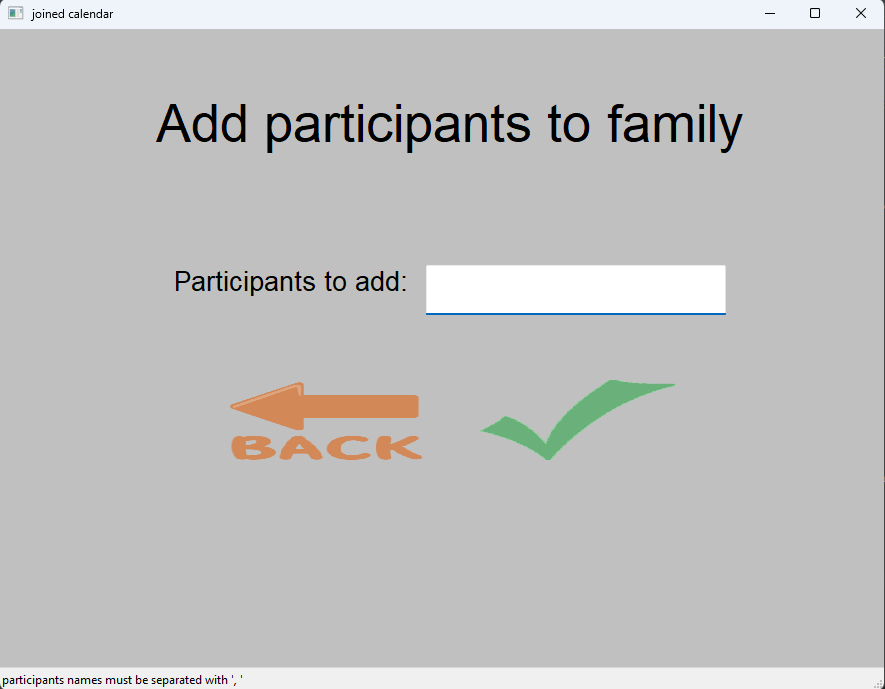
**המסכים בפועל:**





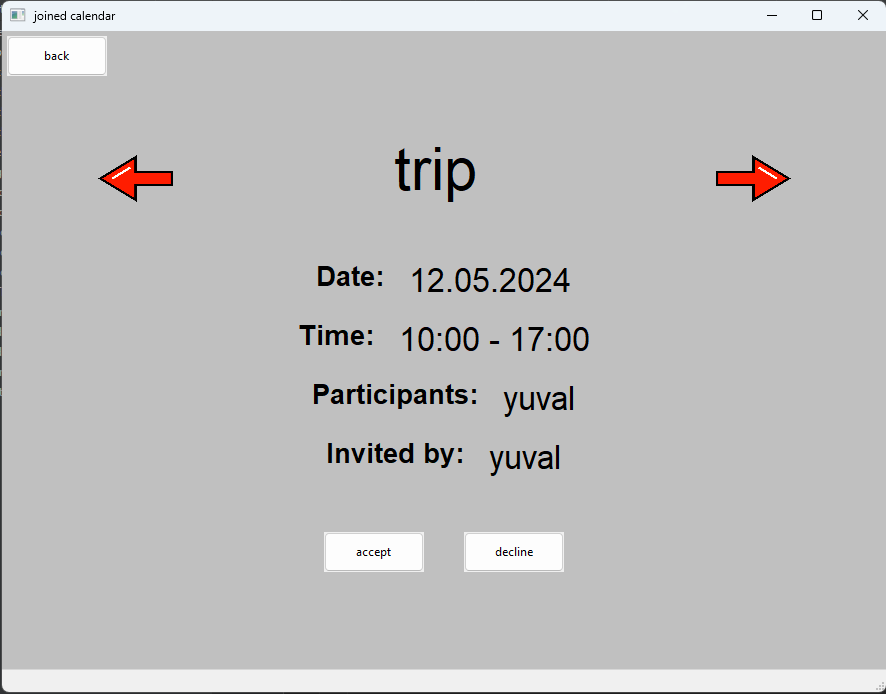
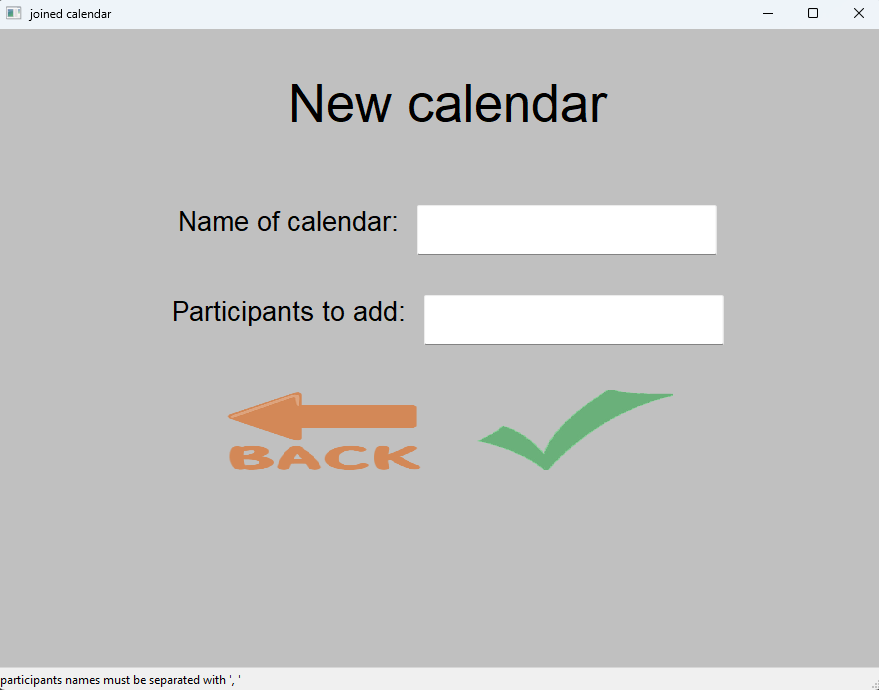
אירועי יום

יומן



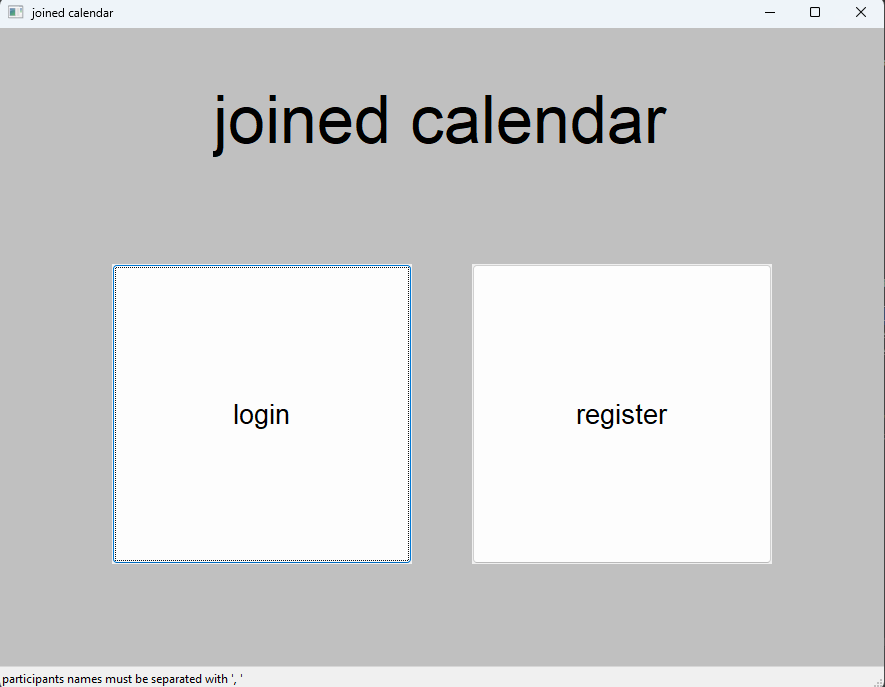
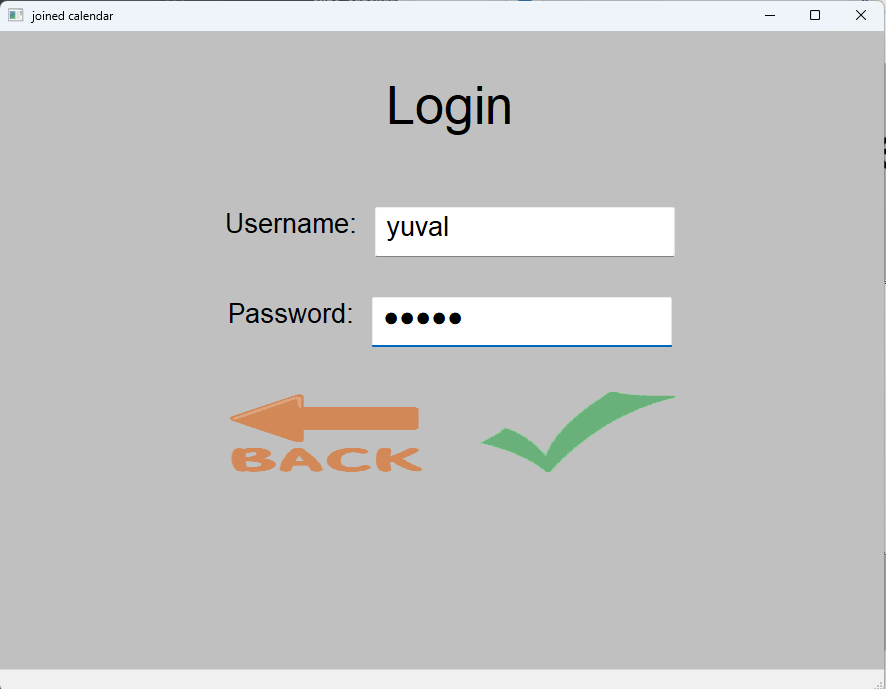
הוספת משתתפים ליומן

שינוי שם יומן



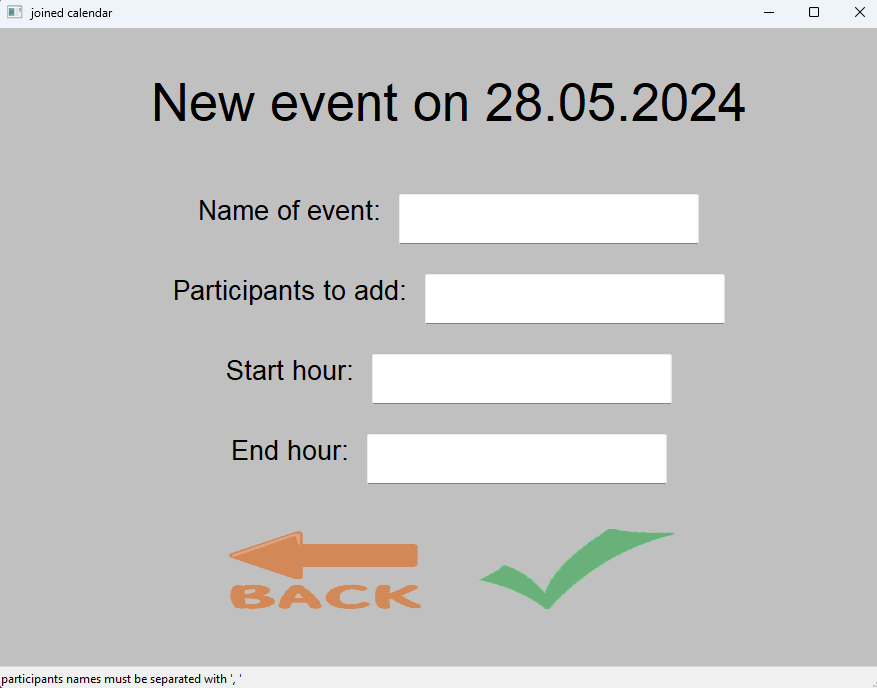
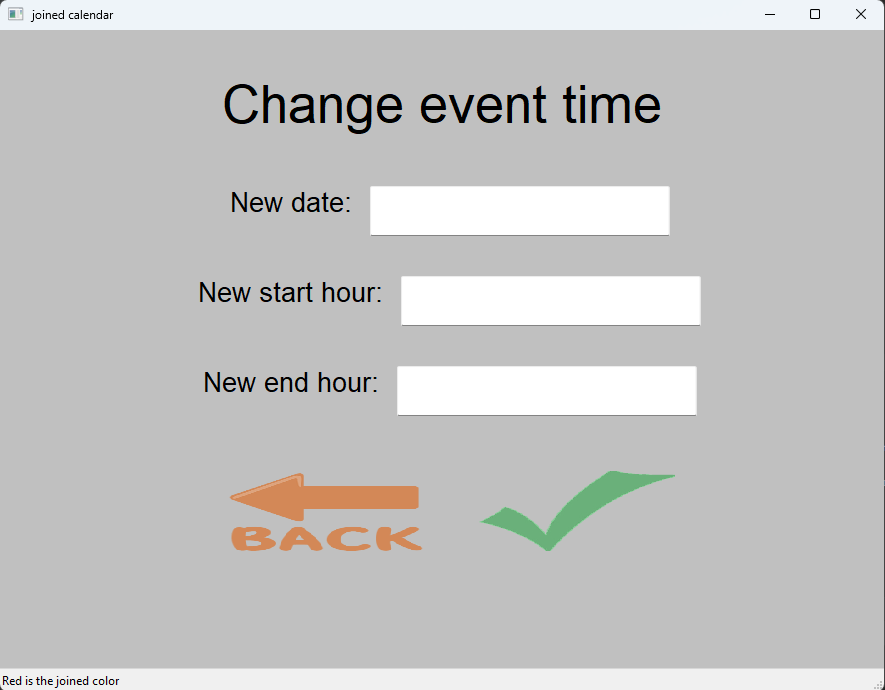
פתיחת יומן חדש

הזמנות לאירועים ויומנים



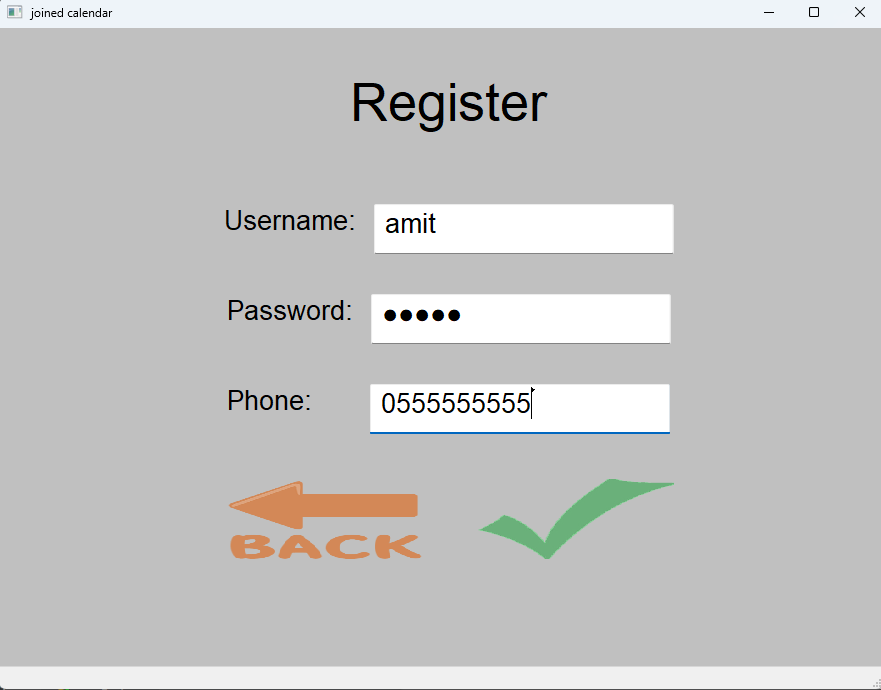
התחברות למשתמש קיים

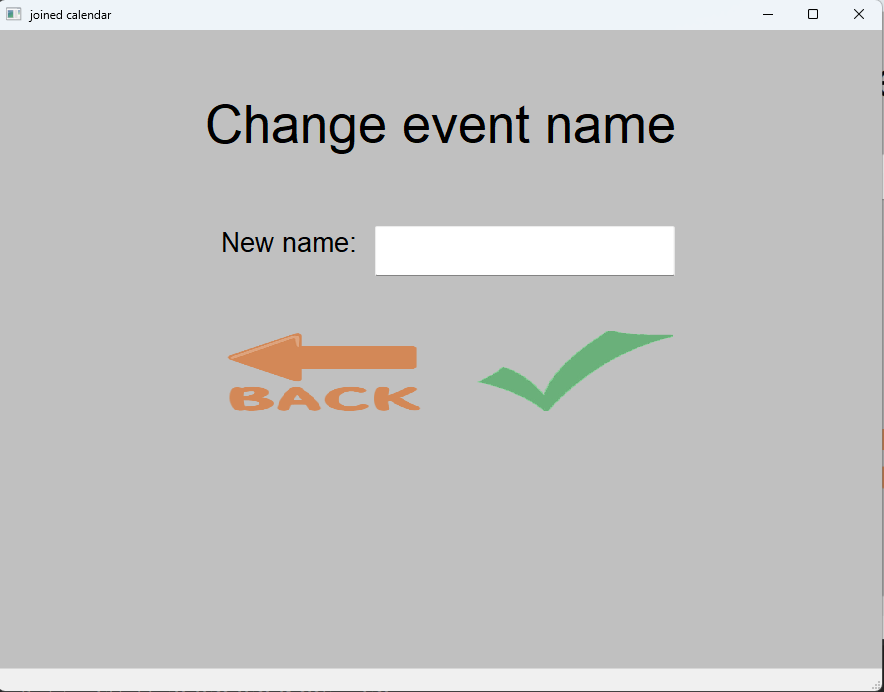
מסך ראשון המפנה להרשמה והתחברות



שינוי זמן אירוע

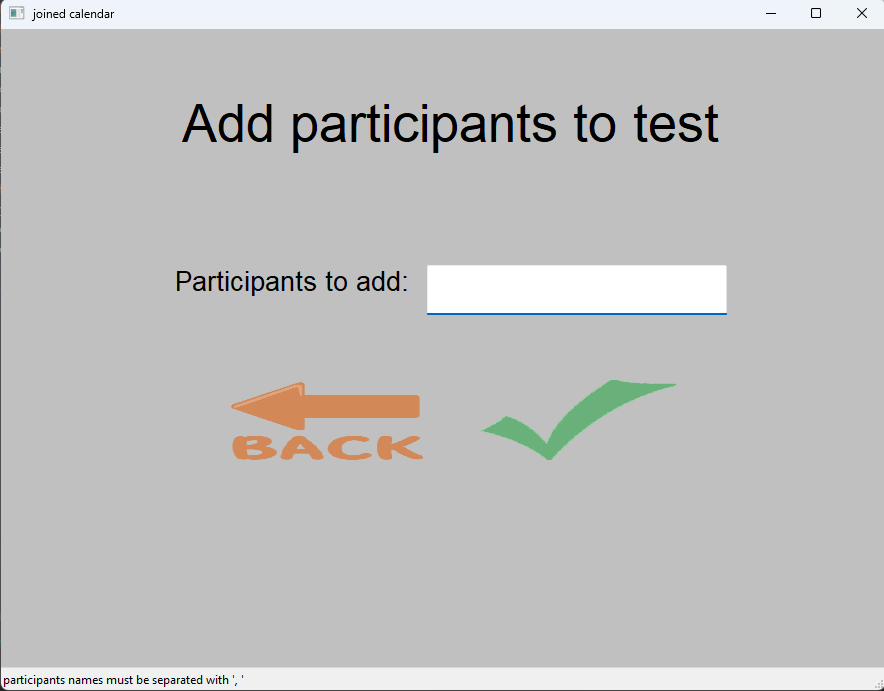
פתיחת אירוע חדש



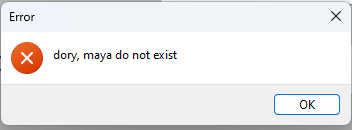


שינוי שם אירוע

הרשמה למשתמש חדש



הוספת משתתפים לאירוע



לפעולות לא חוקיות יוצג חלון שגיאה הנראה כך

1. תיאור פרוטוקול התקשורת:

| תיאור | ממי | למי | קוד | איזה מידע עובר | דוגמא |
| --- | --- | --- | --- | --- | --- |
| בקשה להתחברות | לקוח | שרת | 00 | שם משתמש וסיסמא | "00yuval,5757" |
| תשובה להתחברות | שרת | לקוח | 00 | 0 – הצליח  1 – סיסמא לא נכונה  2 – שם משתמש לא נכון  3 – המשתמש כבר מחובר ממחשב אחר ברגע זה | "001" |
| בקשה להרשמה | לקוח | שרת | 01 | שם משתמש וסיסמא + מספר טלפון | "01yuval,5757,,0555555555" |
| תשובה להרשמה | שרת | לקוח | 01 | 0 – שם משתמש נוצר  1 – שם משתמש תפוס | "010" |
| יומן חדש | לקוח | שרת | 02 | שם היומן, מנהל היומן, רשימת מוזמנים ליומן | "02family,yuval-amit-alon ,amit" |
| תשובה לפתיחת יומן | שרת | לקוח | 02 | 0 – פתח יומן  1 – לא הצליח  + מזהה יומן  או רשימה של משתמשים שלא קיימים ולכן לא הצליח להזמין | "020,1234,[amit]" |
| אירוע חדש | לקוח | שרת | 04 | מזהה יומן, שם אירוע, שעת התחלה, שעת סיום, תאריך, משתתפים | "031234,birthday,14:00,18:00,2.3.2024,amit-alon-yuval " |
| תשובה ליצירת אירוע | שרת | לקוח | 04 | 0 – הצליח  1 – לא הצליח (כי הזמן תפוס או כי משתתפים לא נמצאים ביומן)  + מזהה אירוע או רשימת משתתפים שלא נמצאים ביומן ולכן לא הצליח | "030,1234" |
| בקשת מידע על אירוע | לקוח | שרת | 05 | מזהה אירוע | "05,123" |
| תשובה לבקשת מידע על אירוע | שרת | לקוח | 05 | מזהה אירוע + תאריך + צבע | "05,123,1.2.2024,green" |
| הזמנה ליומן | לקוח | שרת | 10 | מזהה יומן, שם מוזמן | "10maya " |
| הודעה שיש הזמנה ליומן | שרת | לקוח | 10 |  | "10" |
| אישור / סירוב ליומן | לקוח | שרת | 11 | 0 – אישור  1 – סירוב  + מזהה יומן | "110,1234 " |
| הוספת משתמש ליומן | שרת | לקוח | 11 | 0 – הצליח  1 – לא מנהל היומן  + שם משתמש + מזהה יומן | "110,yuval,1234" |
| הזמנה לאירוע | לקוח | שרת | 12 | מזהה יומן, מזהה אירוע, שם מוזמן | "121234,123,amit " |
| תשובה להזמנה לאירוע | שרת | לקוח | 13 | 0 – הצליח  1 – לא  + שם אירוע + שם יומן + שם משתמש | "13,0,birthdayfamily,amit" |
| אישור/סירוב לאירוע | לקוח | שרת | 13 | 0 – אישור  1 – סירוב  + מזהה אירוע + מזהה יומן | "120,1234,1111 " |
| בקשת הזמנות | לקוח | שרת | 14 |  | "14" |
| קבלת הזמנות האירועים והיומנים של המשתמש | שרת | לקוח | 14 | רשימת ההזמנות | "14[[123, amit, family], [1234, dory, test, 13:00, 14:00, 12.11.2024]]" |
| האם הזמנה ליומן עבדה | שרת | לקוח | 15 | סטטוס הצלחה | "15,0" |
| עריכת שם יומן | לקוח | שרת | 20 | מזהה יומן, שם חדש | "20,1234,family " |
| שינוי שם יומן | שרת | לקוח | 20 | 0 – הצליח  1 – לא מנהל היומן  + מזהה יומן + שם חדש | "200,1234,family" |
| שינוי שם אירוע | לקוח | שרת | 21 | מזהה יומן, מזהה אירוע, שם חדש | "21,1234,1111,birthday, " |
| שינוי שם אירוע | שרת | לקוח | 21 | 0 – הצליח  1 – לא מנהל האירוע  + מזהה אירוע, שם חדש | "210 ,1111,birthday" |
| שינוי זמן אירוע | לקוח | שרת | 22 | מזהה אירוע, שעת התחלה, שעת סיום, תאריך | "22 ,1111,14:00,18:00,1.1.2024 " |
| שינוי זמן אירוע | שרת | לקוח | 22 | 0 – הצליח  1 – הזמן תפוס +  מזהה יומן, מזהה אירוע, שעת התחלה, שעת סיום, תאריך | "220,1234,1111,14:00,18:00,1.1.2024" |
| מחיקת אירוע | לקוח | שרת | 30 | מזהה אירוע | "30,1111 " |
| מחיקת אירוע | שרת | לקוח | 30 | סטטוס, מזהה אירוע | "30,0,1111" |
| יציאה מיומן | לקוח | שרת | 31 | מזהה יומן | "31 ,1234 " |
| מחיקת יומן | שרת | לקוח | 31 | סטטוס הצלחה, מזהה יומן | "31,0,1234" |
| יציאה מיומן | שרת | לקוח | 32 | שם משתמש, מזהה יומן | "31,yuval,1234" |
| בקשת מזהה יומנים | לקוח | שרת | 40 | כלום | "40" |
| מזהה יומנים | שרת | לקוח | 40 | מזהי היומנים | 401234,3456 |
| בקשת אירועי יום | לקוח | שרת | 41 | מזהה יומן, תאריך | "41,1234 ,1.2.2024" |
| אירועי יום של המשתמש ושל חברי היומן | שרת | לקוח | 41 | אירועי היום של המשתמש או של המשתתפים ביומן (כמות הפרטים שונה בכל אחד)  אם אירוע של המשתמש - שעות, תאריך, משתתפים, כותרת  אם אירוע של חברי היומן - שם בעל האירוע, שעות, תאריך | 41((birthday, amit-yuval,14:00,18:00,2.3.2024), (holiday, amit-dory-yuval ,14:00,11:00,1.1.2024) |
| בקשה של אירועי החודש | לקוח | שרת | 42 | מזהה יומן, חודש, שנה | "42,1234,1,2024" |
| תשובה לבקשה של אירועי חודש | שרת | לקוח | 42 | רשימה של תאריכים וצבעי המשתמש שמשתתפים באירוע ומזהה האירוע | "42,((1.1.2024,purple,123),(2.1.2024,green,345))" |
| בקשת מידע על יומן | לקוח | שרת | 43 | מזהה יומן | "43,1234" |
| החלפת מפתחות | שרת | לקוח | 50 | מפתח | "50123456" |
| החלפת מפתחות | לקוח | שרת | 50 | מפתח | "50123456" |

1. תיאור מבנה נתונים

מסד נתונים: joined\_calendar\_data

**שם טבלה: calendars**

תיאור: טבלה שתכיל את מזהה היומנים, שמות היומנים והמנהלים שלהם  
מפתח - ID

| Id (int) (PK) | name(str) | manager(str) |
| --- | --- | --- |
| 1234 | Family | Dory |
| 1235 | Friends | Adi |

**שם טבלה: users**

תיאור: תכיל את המשתמשים, הסיסמאות מוצפנות בhash, מספר טלפון

מפתח – שם משתמש

|  |  |  |
| --- | --- | --- |
| (str)username(PK) | Password(str) | Phone\_number(str) |
| Adi | Y21gbjj2 | 0501111111 |
| Noya | Ufrhwfhui34 | 0543333333 |

**שם טבלה: calendars\_participants**

תיאור: תכיל את היומנים, שמות המשתמש של האנשים שבהם

מפתח – מזהה יומן + שם משתמש

|  |  |  |
| --- | --- | --- |
| (str)Calander\_id (PK) | participant(PK) | Color |
| 1234 | adi | Green |
| 1234 | noya | Yellow |

**שם טבלה: event\_participants**

תיאור: משתתפי האירוע ומזהה האירוע

מפתח – מזהה אירוע

|  |  |
| --- | --- |
| (str)event\_id(PK) | participant(PK)(str) |
| 123 | adi |
| 456 | noya |

**שם טבלה: event\_info**

תיאור: תכיל מזהה אירוע, מזהה יומן הראשי של האירוע, שם אירוע, מנהל, שעת התחלה, שעת סיום, תאריך

מפתח – מזהה אירוע

| Event\_id(int)  (PK) | Calendar\_id(str) | name (str) | Manager(str) | Start(str) | End(str) | Date(str) |
| --- | --- | --- | --- | --- | --- | --- |
| 123 | 1234 | Birthday | Noya | 20:00 | 23:00 | 6.1.2024 |
| 456 | 3214 | Movie | Lian | 19:00 | 21:00 | 10.1.2024 |

**שם טבלה: calendar\_invitations**

תיאור: שם משתמש, מזהה יומנים של הזמנות שעדיין לא נשלחו וממי הם

מפתח – מזהה יומן + שם משתמש (מזמין) + שם משתמש (מוזמן)

|  |  |  |
| --- | --- | --- |
| username(str) (PK) | Invited\_by(str) (PK) | id(str) (PK) |
| Etna | noya | 123 |
| Alon | maya | 145 |

**שם טבלה: event\_invitations**

תיאור: שם משתמש, מזהה יומנים, אירועים והמזמינים של ההזמנות שלא נשלחו

מפתח: מזהה יומן + מזהה אירוע + שם משתמש (מזמין) + שם משתמש (מוזמן)

|  |  |  |  |
| --- | --- | --- | --- |
| username(str) (PK) | Invited\_by(str) (PK) | Event\_id(PK) | Calendar\_id |
| Etna | Noya | 123 | 1234 |
| Alon | Maya | 145 | 2345 |

**שם טבלה: reminders**

תיאור: שם משתמש, מזהה אירוע, תאריך תזכורת, שעת תזכורת

מפתח: מזהה אירוע + שם משתמש

|  |  |  |  |
| --- | --- | --- | --- |
| username(str) (PK) | Event\_id (PK) | Date(str) | Time(str) |
| Etna | 123 | 1.1.2024 | 18:00 |
| Alon | 234 | 7.4.2024 | 21:30 |

**משתנים לוקאלים:**

Open\_clients\_socket - מילון של הsockets והip של הלקוחות הפעילים

1. סקירת חולשות והאיומים

Sql injection - הזרקת קוד לבסיס הנתונים בעזרת הקלט. דרך התמודדות: במקום להצמיד את קלט המשתמש לפקודת הsql אני מעבירה אותם כפרמטרים

DOS – denial of service. תקיפת המערכת בעזרת המון שליחות ומידע שיגרמו לה להפסיק להגיב ולקרוס. דרך התמודדות: לא לאפשר למשתמש להתחבר בו זמנית מכמה מקומות. אם יישאר זמן אוסיף גם בדיקה של ההבדל בין זמן ההתנתקות לזמן ההתחברות מחדש ולא אאפשר להתחבר מחדש אם זה לאחר זמן קצר מידי אחרי ההתנתקות.

DDOS – תקיפה מהמון מחשבים שתגרום לשרת להפסיק להגיב. אין סכנה מכיוון שהפרויקט מופעל ברשת המקומית של המעבדה.

Man in the middle(MITM) – אדם ש"עומד באמצע" התקשורת ומקבל את המידע שעובר בין הלקוח לשרת. דרך התמודדות: כל המידע עובר מוצפן כך שגם אם המידע יגיע אל אדם לא רצוי הוא לא יצליח להבין אותו

תקשורת - התקשורת בפרויקט היא באמצעות פרוטוקול tcp אשר אמין.

המידע שעובר ברשת בין הלקוח לשרת מוצפן

גניבת מידע - המידע הקריטי לא נשמר אצל הלקוח אלא אצל השרת ובבסיס הנתונים. הסיסמאות נשמרות בבסיס הנתונים כמוצפנות בhash אז גם אם יפרצו לבסיס הנתונים לא יוכלו לגנוב סיסמאות

התחזות – המערכת לא מאפשרת להתחברות לאותו משתמש מכמה מחשבים בו זמנית

**פרק ה' - 'joined calendar' - הקוד**

* סוגיה 1:

הפונקציה מחליפה מפתחות סימטרים בשיטת דיפי הלמן.

def \_set\_key(self, socket, ip):  
 """  
 set key  
 :param socket:  
 :param ip:  
 :return:  
 """  
 random\_num = random.randint(0, self.p - 3) + 1  
 personal\_key = **"50"** + str(self.g \*\* random\_num % self.p)  
 try:  
 socket.send(str(len(personal\_key)).zfill(3).encode())  
 socket.send(personal\_key.encode())  
 except Exception as e:  
 print(**"server comm - set key "**, str(e))  
 try:  
 data\_len = int(socket.recv(3).decode())  
 data = socket.recv(data\_len).decode()  
 except Exception as e:  
 self.\_disconnect\_client(socket)  
 else:  
 key = str(int(data[2:]) \*\* random\_num % self.p)  
 self.open\_clients[socket] = (ip, Encryption(key))

החלק הקורא לפנוקצית החלפת המפתח אשר לא מאפשר תקשורת אחרת עם לקוחות עד שאין להם מפתח:

while self.is\_running:  
 rlist, wlist, xlist = select.select(([self.socket]) + list(self.open\_clients.keys()),  
 list(self.open\_clients.keys()), [], 0.03)  
 for current\_socket in rlist:  
 # new client  
 if current\_socket is self.socket:  
 client, addr = self.socket.accept()  
 print(**f'**{addr[0]} **connected!'**)  
  
 threading.Thread(target=self.\_set\_key(client, addr[0]))  
  
 continue

* סוגיה 2:

חיבור של שתי רשימות (אחת של כל מזהי האירועים ביום מסוים והשניה של כל מזהי האירועים הכוללים משתתף מסוים) כך שברשימה החדשה יהיו רק המזהי אירועים החוזרים על עצמם בשתי הרשימות ללא כפילויות. כך מקבלים את כל המזהים של אירועים ביום מסוים של משתמש.

def get\_day\_ids(self, participant, date):  
 """  
 get list of this day calendars events ids  
 :param participant:  
 :param date:  
 :return:  
 """  
 sql = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE date = ?"** self.db\_cursor.execute(sql, (date, ))  
 event\_ids = self.db\_cursor.fetchall()  
 event\_ids = set([str(i[0]) for i in event\_ids])  
 sql = **"SELECT event\_id FROM "** + self.events\_participants + **" WHERE participant = ?"** self.db\_cursor.execute(sql, (participant,))  
 event\_ids2 = self.db\_cursor.fetchall()  
 event\_ids2 = set([i[0] for i in event\_ids2])  
 final = list(event\_ids & event\_ids2)  
 return final

* סוגיה 3:

הפיכת עצם (במקרה זה רשימה של הזמנות, שאחד מאבריה של כל הזמנה הוא רשימה של משתתפים באירוע / יומן שההזמנה אליו) למחרוזת כדי להפוך את השליחה בתקשורת ללקוח ליותר קלה.

def pack\_invitations(invitations):  
 """  
 pack msg according to the protocol  
 :param invitations:  
 :return:  
 """  
 if not invitations:  
 day\_events = []  
 for i in invitations:  
 i[2] = **"$"**.join(i[2])  
  
 invitations = [**"^"**.join(x) for x in invitations]  
 invitations = **"\*"**.join(invitations)  
 return **f"14**{invitations}**"**

סוגיה 4:

מילון של פעולות לפי opcodes המאפשר התייחסות גנרית לכל שליחה המתקבלת מהשרת והופך כל פעם שצריך להוסיף פעולה לפשוט יותר.

opcodes = {**"00"**: handle\_login, **"01"**: handle\_sign\_up, **"02"**: handle\_new\_calendar, **"04"**: handle\_new\_event,  
 **"05"**: handle\_event\_info, **"10"**: handle\_there\_is\_an\_invitation, **"11"**: handle\_add\_participant\_to\_calendar,  
 **"13"**: handle\_is\_event\_invitation\_work, **"14"**: handle\_invitations, **"15"**: handle\_is\_calendar\_invitation\_work,  
 **"20"**: handle\_calendar\_name\_change, **"21"**: handle\_event\_name\_change, **"22"**: handle\_time\_change,  
 **"30"**: handle\_delete\_event, **"31"**: handle\_delete\_calendar, **"32"**: handle\_exit\_calendar, **"40"**: handle\_calendar\_ids,  
 **"41"**: handle\_day\_events, **"42"**: handle\_month\_events}

ההפעלה של הפעולות בצורה זו

def handle\_recv(msg\_q):  
 """  
 handle msgs from server  
 :param msg\_q:  
 :return:  
 """  
 while True:  
 msg = msg\_q.get()  
 opcode, params = protocol.unpack(msg)  
 print(**f'got from server:** {msg}**'**)  
 if opcode in opcodes:  
 print(opcodes[opcode](params))  
 else:  
 print(**f'command number** {opcode} **not in dictionary'**)

* סוגיה 5: תקשורת בין הגרפיקה ללוגיקה ובין הלוגיקה לגרפיקה

תקשורת בין הלוגיקה לגרפיקה שמתבצעת באמצעות pub ומשתמשת בcallAfter מכיוון שכך הקריאה לsendMessage מחכה עד שהפעולה הקודמת בגרפיקה מסתיימת ורק אז מתבצעת, דבר שיכול למנוע בעיות.

wx.CallAfter(pub.sendMessage, **"show calendar"**, name=name, manager=manager, participants=participants)

במחלקת הmainPanel בגרפיקה מתבצעת שורה זו אשר מחברת בין הקריאה בלוגיקה, להפעלת הפעולה הרצויה בגרפיקה. במקרה זה הצגת יומן עם הפרמטרים ההועברו מהלוגיקה.

pub.subscribe(self.show\_calendar, **"show calendar"**)

תקשורת בין הגרפיקה ללוגיקה שמתבצעת באמצעות תור. התור הוא של טאפלים. איבר ראשון בטאפל הוא האומר לאיזה פעולה לפנות בלוגיקה והאיבור השני בטאפל הוא שמעביר את הפרמטרים לפעולה זו.

self.frame.graphics\_q.put((**"register"**, (username, password, phone)))

בלוגיקה מוציאים את העצם הראשון מהתור ופועלים לפיו. בדוגמא זו – הפעלת הפעולה להרשמה עם המשתנים שהועברו מהגרפיקה.

def handle\_graphics(graphics\_q):  
 """  
 handle msgs from graphics  
 :param graphics\_q:  
 :return:  
 """  
 while True:  
 msg = graphics\_q.get()  
 opcode, params = msg  
 print(**f'got from graphics:** {msg}**'**)  
 if opcode == **"register"**:  
 username, password, phone = params  
 sign\_up(username, password, phone)

סוגיה 6: יצירת מזהי היומנים ומזהי האירועים באופן אוטומטי בבסיס הנתונים בעזרת autoincrement, כך שכאשר סוגרים את השרת, המזהה האחרון שהיה יישמר, כך שלא יחזרו על עצמם מזהים. ושזה יתבצע ללא יצירת טבלה נוספת בבסיס הנתונים לשמירת המזהים האחרונים.

def \_create\_db(self):  
 """  
 connect to the db and create the tables that dont exists  
 :return: None  
 """  
self.db\_conn = sqlite3.connect(self.name\_of\_db)  
self.db\_cursor = self.db\_conn.cursor()  
sql\_table = [**"CREATE TABLE IF NOT EXISTS "** + self.calendars +  
**" (calendar\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), manager VARCHAR(20))"**,

**"CREATE TABLE IF NOT EXISTS "** + self.event\_info +  
**" (event\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, calendar\_id VARCHAR(3), name VARCHAR(30), manager VARCHAR(20), start\_hour VARCHAR(5), end\_hour VARCHAR(5), date VARCHAR(10))"**,

סוגיה 7: קביעת צבע יום ביומן. מתבצע על ידי בדיקה של צבע כל האירועים בתאריך מסוים ביחס ליומן מסוים (מתחשב באם יש משתתפים חופפים בין היומן לאירוע ואם כן כמה ומה צבעם ביומן), ולאחר מכן בודק אם הצבעים של האירועים הבודדים שונים אחד מהשני (מה שאומר שיש ביום זה אירועים לכמה משתתפים ביומן), ואם כן קובע את הצבע לצבע המשותף – אדום.

def get\_day\_color(self, date, calendar\_id):  
 """  
 get day color  
 :param date:  
 :param calendar\_id:  
 :return:  
 """  
 day\_color = []  
 sql = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE date = ?"** self.db\_cursor.execute(sql, (date,))  
 event\_ids = self.db\_cursor.fetchall()  
 event\_ids = [i[0] for i in event\_ids]  
 for id in event\_ids:  
 temp = self.get\_some\_event\_info(id, calendar\_id)  
 if len(temp) != 0:  
 if len(day\_color) == 0:  
 day\_color = temp  
 if temp[1] == **'RED'**:  
 break  
 else:  
 if temp[1] != day\_color[1]:  
 day\_color[1] = **'RED'** break  
 return day\_color

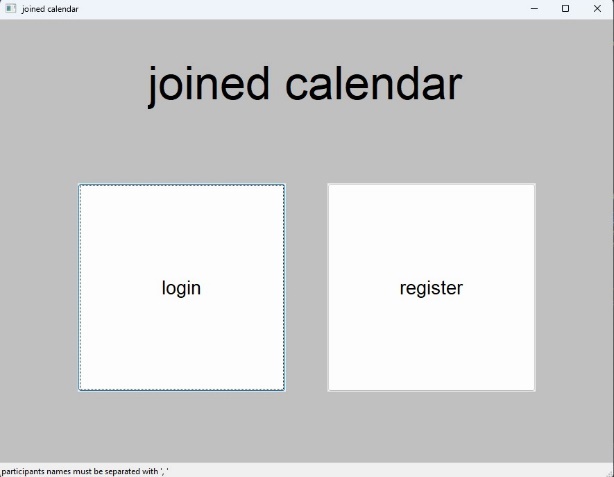
פעולה למציאת צבע אירוע בודד – קוראים לה בפעולה למציאת צבע יום.

def get\_some\_event\_info(self, event\_id, calendar\_id):  
 """  
 get event color (by participants color) and date  
 :param event\_id:  
 :param calendar\_id  
 :return:  
 """  
 info = []  
 if self.\_is\_calendar\_exists(calendar\_id) and self.\_is\_event\_exists(event\_id):  
 participants = self.get\_calendar\_participants(calendar\_id)  
  
 set\_participants = set(participants)  
 p = self.get\_event\_participants(event\_id)  
 set\_p = set(p)  
 both = set\_p & set\_participants  
 sql = **"SELECT date FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (event\_id,))  
 date = self.db\_cursor.fetchone()[0]  
 if both:  
 if len(both) > 1:  
 info = [date, self.joined\_color]  
 else:  
 both = list(both)  
 info = [date, self.get\_color(both[0], calendar\_id)]  
 return info

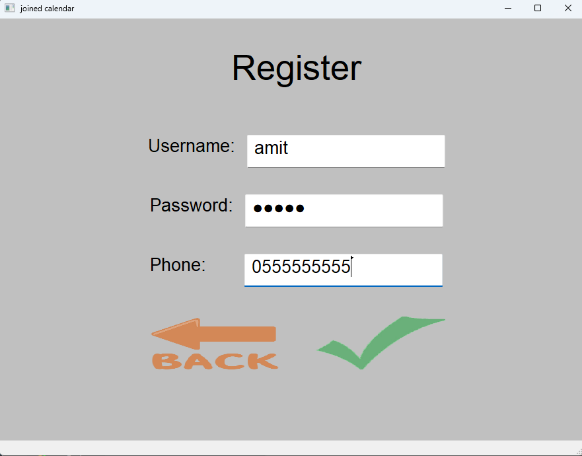
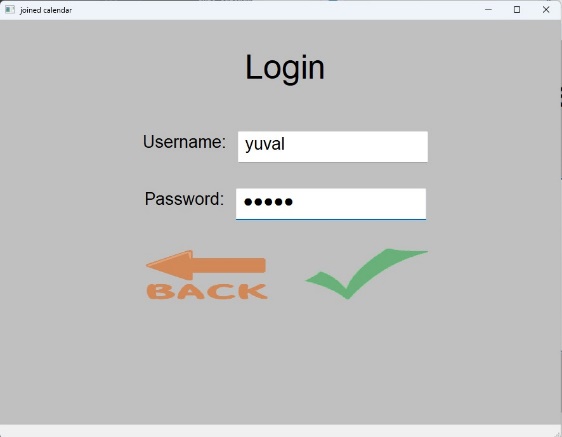
**פרק ו' - 'joined calendar' – בדיקות ('קופסא לבנה')**

| שם הבדיקה | מטרת הבדיקה | מה נדרש לבצע | מתי | מה בוצע בפועל |
| --- | --- | --- | --- | --- |
| הוספת יומן | האם המערכת מאפשרת הוספת יומן | להיכנס למערכת ולנסות לפתוח יומן | 16.5.24 | נפתח היומן החדש ונשלחו הזמנות למשתמשים המבוקשים |
| הוספת אדם ליומן | האם אפשר להוסיף אדם ליומן קיים | להיכנס ליומן קיים ולהוסיף לו משתתף | 16.5.24 | נשלחה הזמנה למשתמש |
| הוספת אירוע | לבדוק האם אפשר להוסיף אירוע | להיכנס ליומן קיים ולהוסיף אירוע | 16.5.24 | נוסף האירוע ונשלחו הזמנות למשתמשים המבוקשים |
| מחיקת אירוע | האם אפשר למחוק אירוע קיים | להיכנס לאירוע קיים ביומן קיים ולמחוק אותו | 16.5.24 | נמחק האירוע |
| יציאה מיומן | לבדוק האם משתתף יכול לצאת מיומן וימחקו כל האירועים שלו | להיכנס ליומן קיים ולצאת ממנו. פעם אחת כאשר המשתמש הוא מנהל היומן ופעם אחת כאשר לא | 16.5.24 | כאשר המשתמש לא היה מנהל - יצא מהיומן ומחק את האירועים של המשתמש מהיומן  כאשר היה מנהל – גם מחק את היומן מכל שאר המשתמשים |
| יצירת משתמש | האם אפשר ליצור משתמש חדש במערכת | להיכנס למערכת וליצור משתמש חדש עם הפרטים הנחוצים | 16.5.24 | נוצר המשתמש ונפתח יומן אישי |
| התחברות למשתמש | לבדוק האם אפשר להתחבר למשתמש קיים | להתחבר למשתמש קיים | 16.5.24 | התחבר למשתמש ונפתח היומן האישי |
| הוספת משתמש לאירוע | האם אפשר להוסיף משתמש לאירוע | להיכנס לאירוע ולשלוח הזמנה למשתמש | 16.5.2024 | שלח הזמנה למשתמש השני והדליק אצלו את כפתור ההזמנות |
| שינוי שם יומן | האם המנהל יכול לשנות את שם היומן | ליצור יומן ולנסות לשנות את השם שלו | 16.5.2024 | שינה את שם היומן |
| שינוי שם אירוע | לבדוק אם אפשר לשנות שם אירוע | ליצור אירוע ולנסות לשנות את השם שלו | 16.5.2024 | שינה את שם האירוע |
| שינוי שעות אירוע | לבדוק אם אפשר לשנות את שעות האירוע | ליצור אירוע ולנסות לשנות את השעות שלו | 16.5.2024 | שינה את שעות האירוע |
| שינוי תאריך אירוע | לבדוק אם אפשר לשנות את התאריך שלו | ליצור אירוע ולנסות לשנות את התאריך שלו | 16.5.2024 | שינה את תאריך האירוע |
| הוספת משתתף ליומן כאשר אתה לא מנהל היומן | לבדוק אם המערכת חוסמת הוספת משתתף ליומן אם אתה לא המנהל שלו | להצטרף ליומן שלא אתה יצרת (דרך ההזמנות) ולנסות לצרף משתמש | 16.5.2024 | לא הוסיף את המשתתף אלא פתח חלונית שגיאה שאומרת שאי אפשר לבצע זאת |
| שינוי זמן האירוע כאשר משתתף באירוע לא פנוי בזמן זה | לבדוק אם המערכת חוסמת שינוי זמן אירוע לזמן שאחד או יותר ממשתתפיו לא פנויים בו | ליצור אירוע רב משתתפים (להזמין אנשים לאירוע ושיאשרו אותו) ולאחר מכן לנסות לשנות את השעה לשעה שלא כל המשתתפים פנויים בה | 16.5.2024 | לא שינה את זמן האירוע אלא פתח חלונית שגיאה שאומרת מי המשתתפים שלא פנויים בזמן זה |
| הוספת משתתף לאירוע ביומן שהוא לא משתתף בו | לבדוק אם המערכת חוסמת הוספת משתתף לאירוע דרך יומן שלא משתתף בו | ליצור יומן ואירוע ולנסות להזמין אליו משתמש שלא נמצא ביומן | 16.5.2024 | לא הוסיף את המשתתף לאירוע אלא פתח חלונית שגיאה שאומרת שדבר זה לא אפשרי כי לא קיים ביומן |
| אישור הזמנה לאירוע | לבדוק האם המערכת מאפשרת אישור הזמנה לאירוע | להיכנס להזמנות ולאשר הזמנה לאירוע (שמשתמש אחר כבר שלח) | 16.5.2024 | מוסיפה את המשתמש לאירוע |
| אישור הזמנה ליומן | לבדוק האם המערכת מאפשרת אישור הזמנה ליומן | להיכנס להזמנות ולאשר הזמנה ליומן (שמשתמש אחר כבר שלח) | 16.5.2024 | מוסיפה את המשתמש ליומן ופותחת אותו אצלו |

**פרק ז' - 'joined calendar' – מדריך למשתמש**

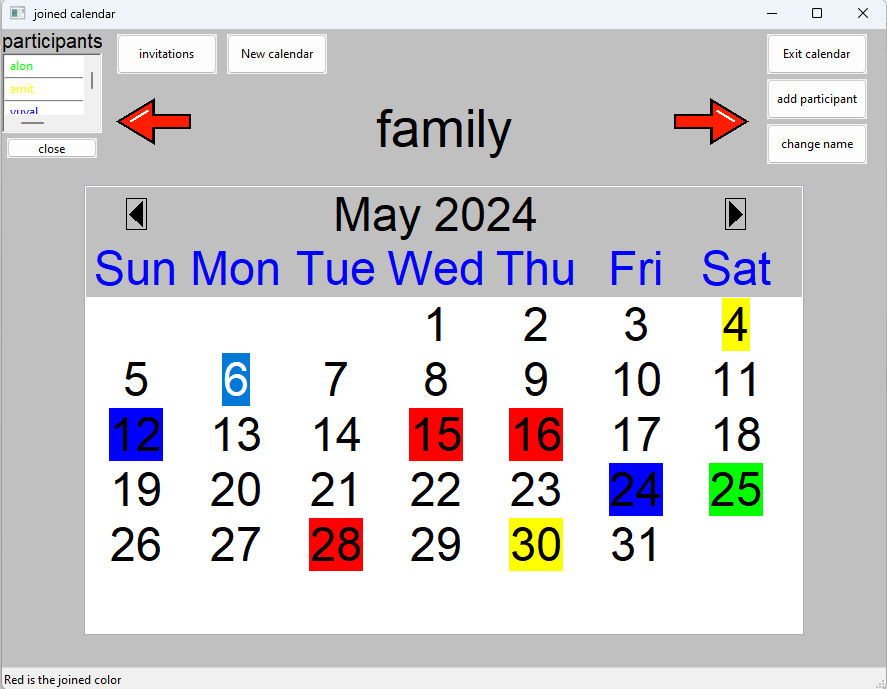


בהתחלה בוחרים בין הרשמה להתחברות.



מכניסים שם משמתמש קיים וסיסמא באורך של לפחות 5

מכניסים שם משתמש וסיסמא באורך של לפחות 5 תווים לפחות שלא כוללים את התווים ",", "$", "^", "\*" , ומספר טלפון באורך 10 ספרות



מוצג היומן האישי

מדפדפים בין היומנים דרך שימוש בחצים האדומים. כל צבע ביומן מייצג משתתף (חלוקת הצבעים מוצגת ברשימת המשתתפים). הצבע האדום זה הצבע המשותף לכמה אנשים.

לחיצה על אחד הימים יפתח את אירועי אותו יום.

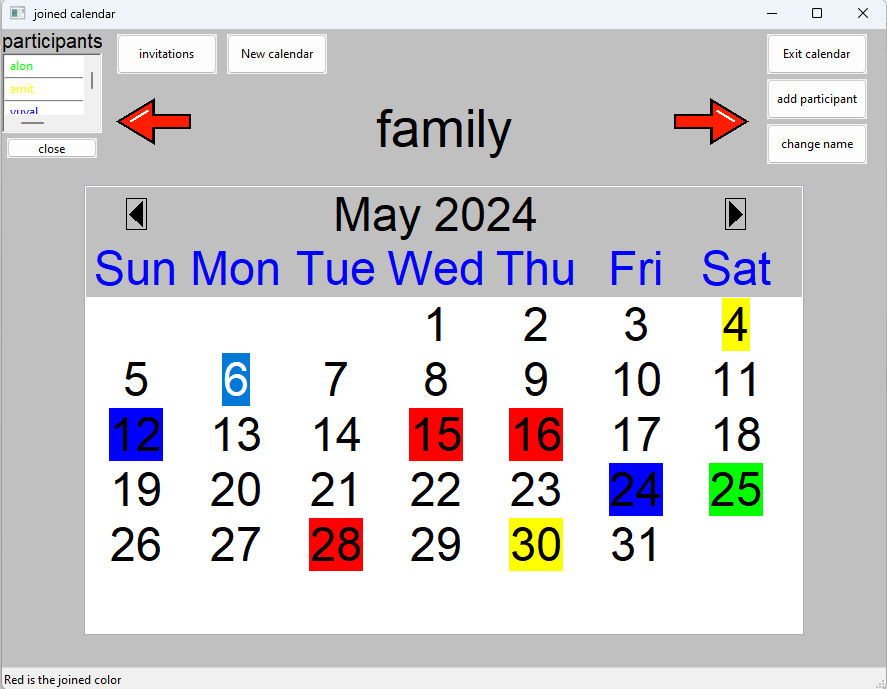
לחיצה על כפתור exit calendar יצא מהיומן

לחיצה על כפתור add participants יוביל למסך של הוספת משתתפים ליומן

לחיצה על כפתור change name יוביל למסך של שינוי שם היומן

לחיצה על כפתור new calendar יוביל למסך לפתיחת יומן חדש

לחיצה על כפתור invitations יפתח את ההזמנות הקיימות ליומנים ולאירועים. כאשר הכפתור ירוק יש הזמנות וכאשר אפור אין.



יציאה מיומן

הוספת משתתפים

שינוי שם יומן

יומן חדש

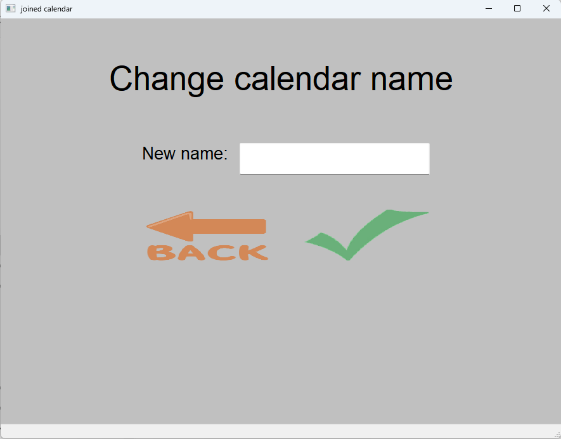
הצגת הזמנות

החלפת שם יומן: בשביל להחליף שם ליומן מקלידים את השם ולוחצים על וי. לחזור אחורה על back.

רק מנהל היומן יכול להחליף שם.

ליומן האישי אי אפשר להחליף שם.

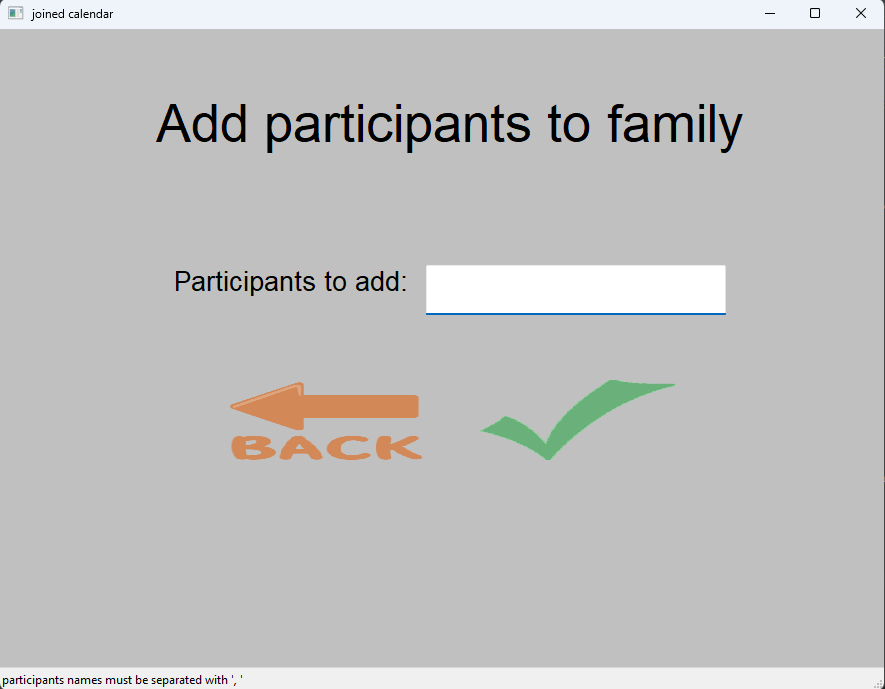
השם לא יכול להיות personal"" או לכלול בתוכו את התווים ",", "$", "\*", "^"



הוספת משתתפים ליומן: בשביל להוסיף משתתפים ליומן מקלידים את שמות המשתמשים שרוצים להוסיף בהפרדה של ", " ולוחצים על וי. אם רוצים לחזור למסך היומן לוחצים על back

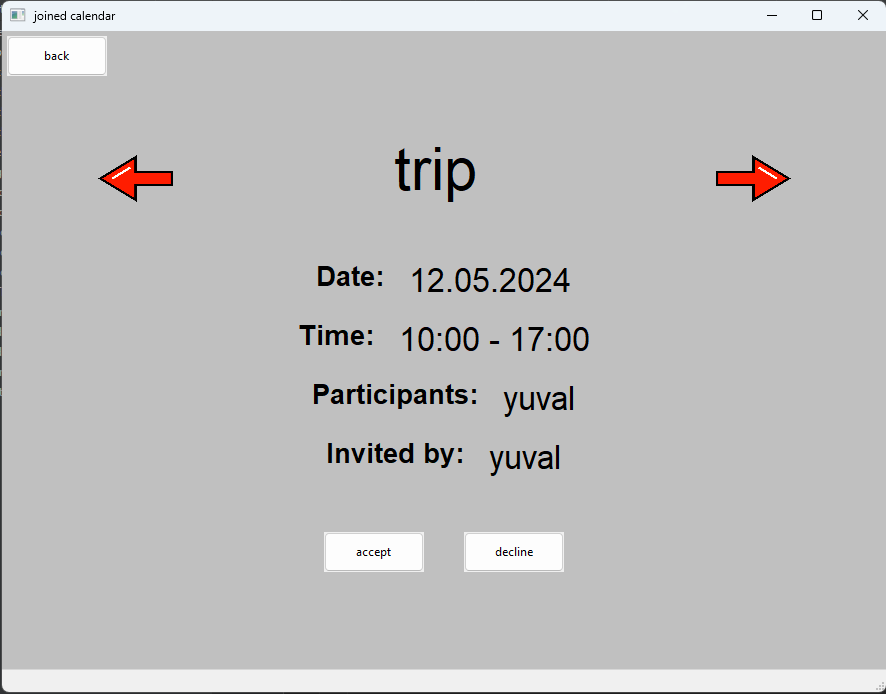
רק מנהל היומן יכול להוסיף משתתפים

ליומן האישי אי אפשר להוסיף משתתפים.



הצגת הזמנות: כדי לאשר הזמנה ליומן או אירוע לוחצים על accept וכדי לסרב על decline. כדי לחזור אחורה לוחצים על back.

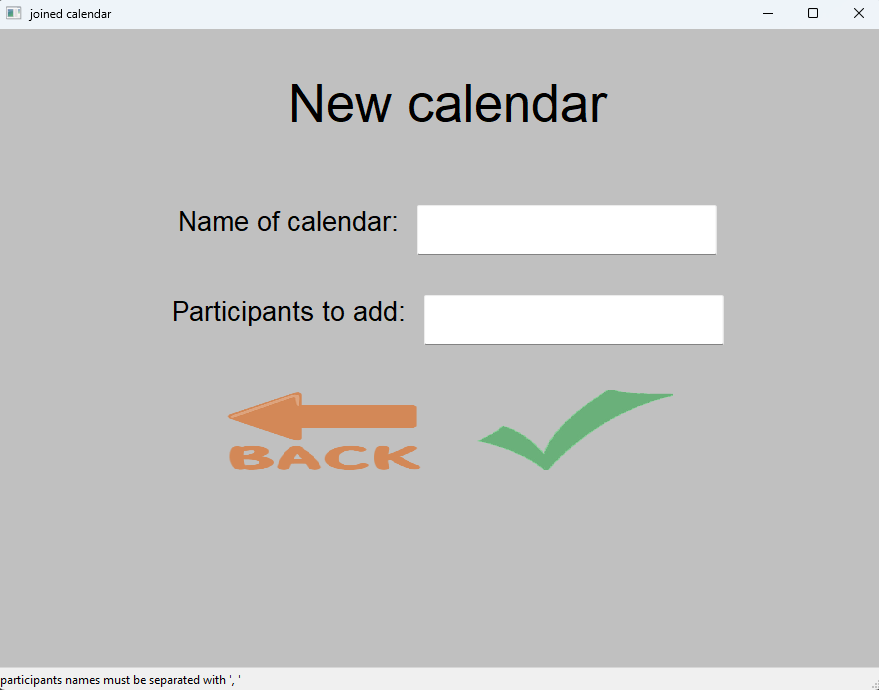
מדפדפים בין ההזמנות על ידי לחיצה על החצים האדומים. כאשר מאשרים יומן היומן מוצג על המסך.



פתיחת יומן חדש: כדי לפתוח יומן חדש מקלידים את השם ואת המשתתפים ולוחצים על וי. כדי לחזור אחורה למסך היומן לוחצים על back.

השם לא יכול להיות personal"" או לכלול בתוכו את התווים ",", "$", "\*", "^"

מקלידים את שמות המשתמשים שרוצים להוסיף בהפרדה של ", "



הצגת אירועי יום: כדי לדפדף בין האירועים לוחצים על החצים האדומים.

כדי ליצור אירוע חדש לוחצים על כפתור new event

כדי לשנות את זמן האירוע לוחצים על כפתור change time

כדי לשנות את שם האירוע לוחצים על כפתור change name

כדי להוסיף משתתפים לאירוע לוחצים על כפתור add participants

כדי למחוק אירוע לוחצים על delete event (אפשר למחוק אירוע רק אם אתה המנהל שלו)

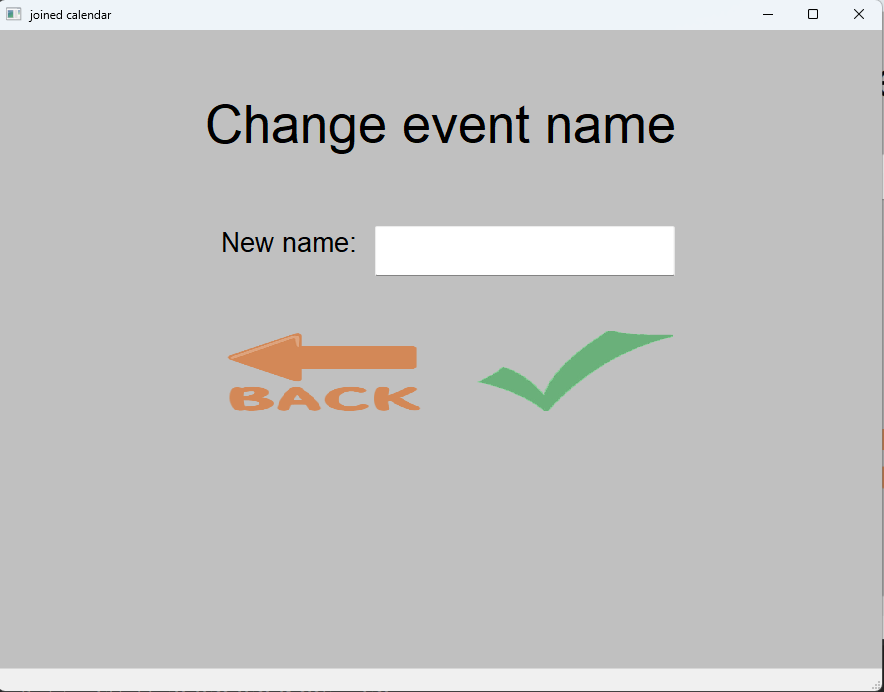
תמונה שמכילה טקסט, צילום מסך, תוכנה, גופן

התיאור נוצר באופן אוטומטיכדי לחזור אחורה למסך היומן לוחצים על back

שינוי שם אירוע: כדי לשנות שם אירוע כותבים את שם האירוע ולוחצים על וי. כדי לחזור אחורה לאירועי היום לוחצים על back.

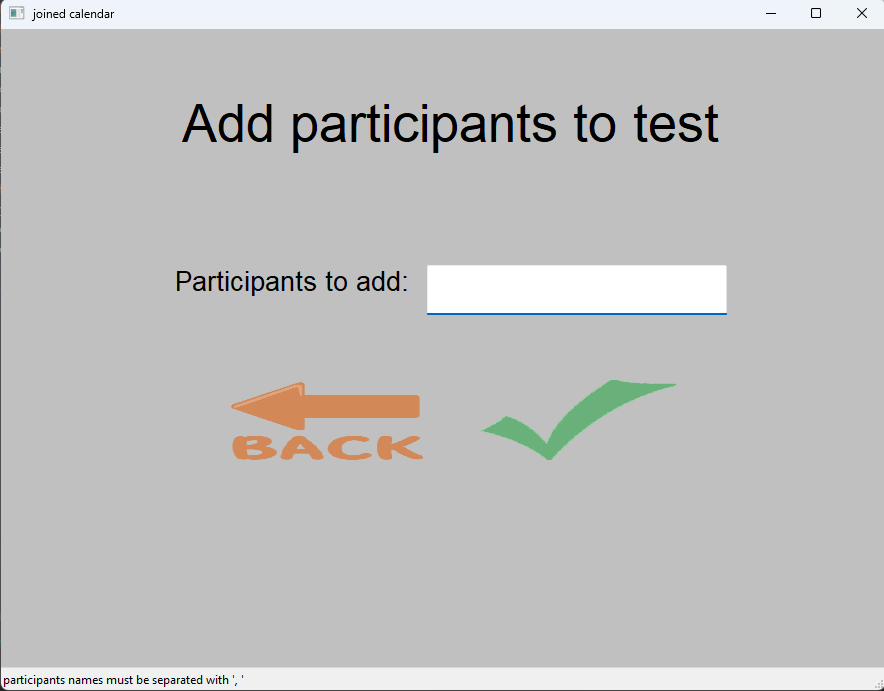
השם לא יכול לכלול בתוכו את התווים ",", "$", "\*", "^"

רק המנהל יכול להחליף שם



הוספת משתתפים לאירוע: בשביל להוסיף משתתפים לאירוע מקלידים את שמות המשתמשים שרוצים להוסיף בהפרדה של ", " ולוחצים על וי. אם רוצים לחזור לאירועי היום לוחצים על back

רק מנהל האירוע יכול להוסיף משתתפים

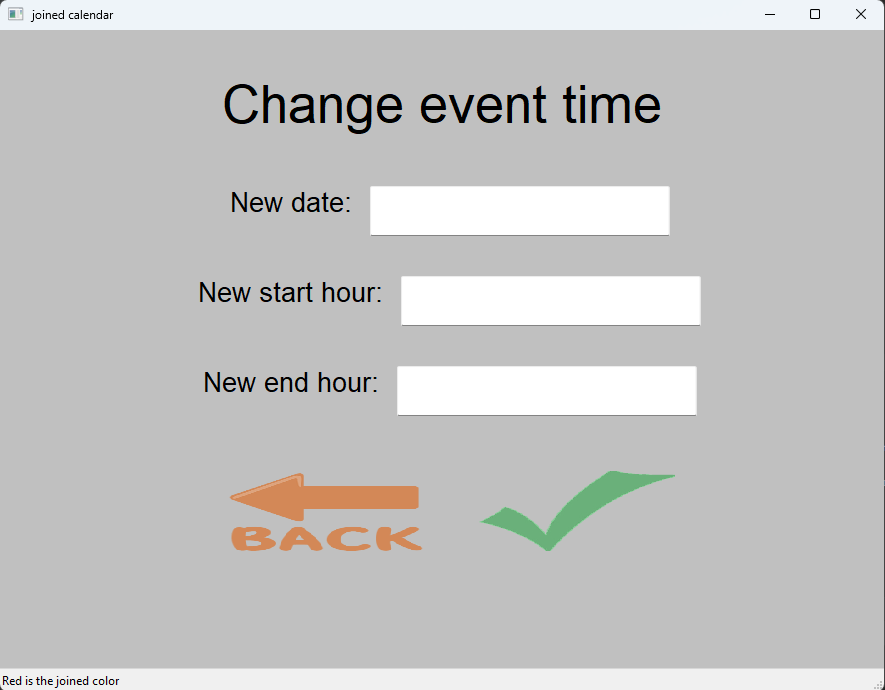


שינוי זמן אירוע: כדי לשנות את זמן האירוע מקלידים את שעת ההתחלה, שעת הסיום והתאריך.

התאריך צריך להיות בצורה כזאת dd.mm.yyyy (ארבע ספרות לשנה, שתיים לחודש ושתיים ליום בהפרדה של נקודה)

השעות צריכות להיות בצורה כזאת hh:mm (שתי ספרות לשעה ושתי ספרות לדקות מופרדות בנקודותיים)

רק המנהל יכול להחליף שעה לאירוע ורק אם כל שאר משתתפי האירוע פנויים בשעה זו.



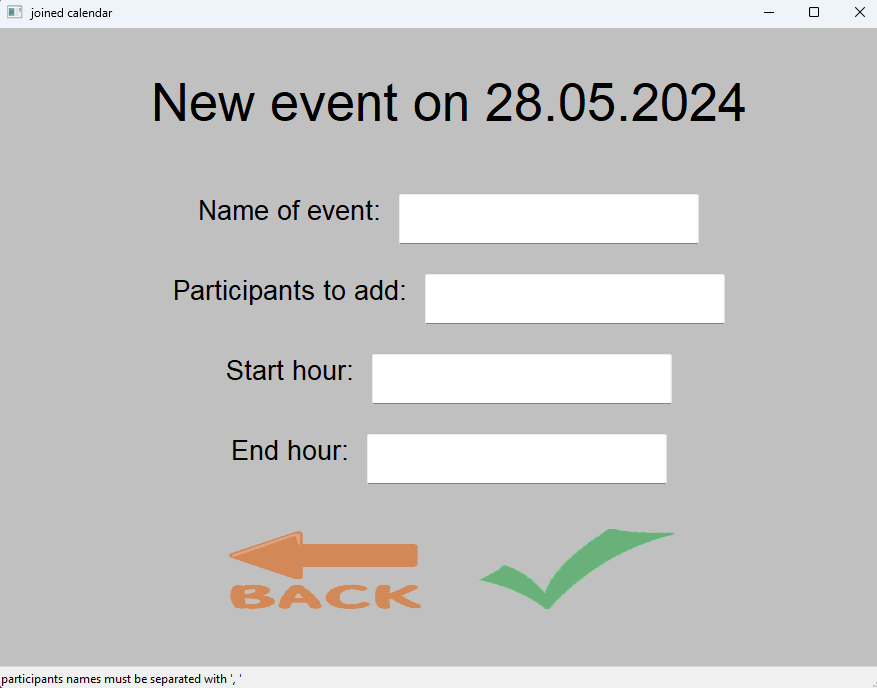
הוספת אירוע חדש: כדי להוסיף אירוע חדש ממלאים את השם, המשתתפים, שעת ההתחלה ושעת הסוים ולוחצים על וי. כדי לחזור לאירועי היום לוחצים על back

השעות צריכות להיות בצורה כזאת hh:mm (שתי ספרות לשעה ושתי ספרות לדקות מופרדות בנקודותיים)

השם לא יכול לכלול בתוכו את התווים ",", "$", "\*", "^"

מקלידים את שמות המשתמשים שרוצים להוסיף בהפרדה של ", "

המשתתפים חייבים להיות קיימים ביומן שבו יוצרים את האירוע.



**פרק ח' - 'joined calendar' – רפלקציה**

לא היה פשוט לבחור את נושא הפרויקט. רציתי פרויקט שיעניין אותי ושייגע בנושא שמפריע לי בחיי היום יום, אך גם יעסוק בתחומים החדשים לי וייאלץ אותי ללמוד ולצאת מאזור הנוחות שלי.

העבודה על הפרויקט הייתה לי כל כך מעניינת וחווייתית. זאת הייתה הזדמנות לחקור נושאים לבד, ללמוד דברים בתחומים שמעניינים אותי וחדשים לי, וליצור תוכנית שמטרתה להקל על קושי שאני חווה פעמים רבות – הקושי לתכנן ולתאם פעילויות יחד עם אנשים נוספים. כל זה דירבן אותי וחיזק את הרצון שלי ליצור ולהתקדם בפרויקט בצורה הכי טובה שאני יכולה.

כשהתחלתי את הפרויקט, הדבר המרכזי שעמד בפני הוא הרצון להפוך את הפרויקט לכמה שיותר נגיש וקל לשימוש למשתמש. חשבתי הרבה מה תהיה הדרך הטובה ביותר לעשות זאת ובתור אחת שלא פעם פספסה דברים על המסך, היו לי שני דברים שהיו לי ברורים שאני רוצה לעשות – שלא יהיה עומס פרטים ודברים על המסך ושהאירועים יהיו בולטים בצבעים שונים.

לכן, החלטתי שהשיטה הטובה ביותר לפי דעתי כדי שלא יהיה מסך עמוס, היא דפדוף בין אירועים על ידי שימוש בחצים אדומים, ולא הצגה של כל אירועי היום בו זמנית.

בנוסף, בגלל המידע הרב שהייתי צריכה לשמור והטבלאות הרבות בבסיס הנתונים, אחד האתגרים המרכזיים שנתקלתי בהם היה איך להצליב את הטבלאות כדי לקבל את המידע הרצוי בכל תרחיש. דבר שהצריך זמן, ניסיונות רבים ואינסוף בדיקות של מקרי קצה.

אך הקושי הגדול ביותר שנתקלתי בו בפרויקט הוא חלוקת הזמן. השקעתי זמן רב על תכנונו של הפרויקט ולא השארתי המון זמן לפיתוח שלו. בנוסף, בגלל שבתקופה זו היו לי מיונים רבים לשנות שירות ולצבא, היה לי עומס גדול והיה קשה לסדר את הזמן. לכן, בעקבות הפרויקט אני חושבת שיכולת תכנון הזמן שלי השתפרה מאוד. ובכל זאת, אם הייתה לי האפשרות לעשות את הפרויקט מחדש, הייתי משנה את חלוקת הזמן שהקדשתי לפרויקט ומתחילה את הפיתוח והקוד יותר מוקדם וכך מקדישה יותר זמן לעיצוב הפרויקט. בנוסף הייתי משנה את סדר העדיפויות לפרויקט כך שהיכולת של "שליחת תזכורות על אירועים למשתמש" תהיה גבוהה יותר ואספיק לממש אותה מוקדם יותר.

בזכות הפרויקט למדתי איך לעבוד בצורה עצמאית. זה פרויקט בסדר גודל הכי גדול שעשיתי עד כה, ולהתחיל אותו הפחיד אותי מאוד. למדתי איך לגשת לפרויקט כזה ואני מקווה שזה יעזור לי לגשת לפרויקטים גדולים בעתיד.

אני רוצה להודות למשפחה שלי, שהתייעצתי איתה הרבה על מה עליי לבחור כנושא הפרויקט, ושהתנהלה כלפיי בסבלנות רבה כאשר הייתי צריכה לעבוד על הפרויקט במקום לבלות איתם.

בנוסף, הייתי רוצה להודות לחבריי ולחבריי למגמה, שייעצו לי ועזרו לי כשנתקלתי בבעיות. ויותר מהכל הייתי רוצה להודות למרי, רכזת המגמה, אשר עזרה, ייעצה וליוותה אותי ואת כל המגמה בפרויקט בצורה כל כך טובה ולא מובנת מאליה בכלל.

**פרק ט' - 'joined calendar' – ביבליוגרפיה**

**הקוד:**

Client comm:

# ClientComm - responsible for communication with the server  
  
import socket  
import threading  
import sys  
import random  
from encryption import Encryption  
import queue  
  
  
class ClientComm:  
 def \_\_init\_\_(self, server\_ip, port, recv\_q):  
 self.server\_ip = server\_ip  
 self.port = port  
 self.recv\_q = recv\_q  
 self.socket = socket.socket()  
 self.key = None  
 self.g = 6269  
 self.p = 1433  
 self.encryption = None  
 threading.Thread(target=self.\_main\_loop, ).start()  
  
  
 def \_main\_loop(self):  
 try:  
 self.socket.connect((self.server\_ip, self.port))  
 except Exception as e:  
 sys.exit(**"server is down, try again later"**)  
 self.\_set\_key()  
 while not self.encryption:  
 pass  
  
 while True:  
 try:  
 data\_len = int(self.socket.recv(3).decode())  
 data = self.socket.recv(data\_len)  
 except Exception as e:  
 sys.exit(**"server is down, try again later"**)  
  
 self.recv\_q.put(self.encryption.decrypt(data)) # decrypted data  
  
  
 def send(self, msg):  
 '''  
 send msg to server  
 :param msg: str  
 :return:  
 '''  
 while not self.encryption:  
 pass  
 msg = self.encryption.encrypt(msg)  
 try:  
 self.socket.send(str(len(msg)).zfill(3).encode())  
 self.socket.send(msg)  
 except Exception as e:  
 print(**'client comm - send'**, str(e))  
 sys.exit(**"server is down, try again later"**)  
  
 def \_set\_key(self):  
 """  
 set key  
 :return:  
 """  
 try:  
 data\_len = int(self.socket.recv(3).decode())  
 data = self.socket.recv(data\_len).decode()  
 except Exception as e:  
 sys.exit(**"server is down, try again later"**)  
  
 random\_num = random.randint(0, self.p-3)+1  
 personal\_key = **"50"** + str(self.g \*\* random\_num % self.p)  
 try:  
 self.socket.send(str(len(personal\_key)).zfill(3).encode())  
 self.socket.send(personal\_key.encode())  
 except Exception as e:  
 print(**"client comm - set key "**, str(e))  
  
 key = str(int(data[2:]) \*\* random\_num % self.p)  
 self.encryption = Encryption(key)  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 msg\_q = queue.Queue()  
 comm = ClientComm(**'127.0.0.1'**, 4500, msg\_q)  
 comm.send(**"hello"**)  
 print(msg\_q.get())

Server comm:

# server\_comm - responsible for communication with the clients  
  
  
import select  
import socket  
import threading  
import queue  
import random  
from encryption import Encryption  
  
  
class ServerComm:  
  
 def \_\_init\_\_(self, port, rcv\_q):  
 self.port = port  
 self.rcv\_q = rcv\_q  
 self.socket = socket.socket()  
 self.open\_clients = {}  
 self.is\_running = False  
 self.g = 6269  
 self.p = 1433  
 threading.Thread(target=self.\_main\_loop).start()  
  
 def \_main\_loop(self):  
 self.socket.bind((**"0.0.0.0"**, self.port))  
 self.socket.listen(3)  
 self.is\_running = True  
 while self.is\_running:  
 rlist, wlist, xlist = select.select(([self.socket]) + list(self.open\_clients.keys()),  
 list(self.open\_clients.keys()), [], 0.03)  
 for current\_socket in rlist:  
 # new client  
 if current\_socket is self.socket:  
 client, addr = self.socket.accept()  
 print(**f'**{addr[0]} **connected!'**)  
  
 threading.Thread(target=self.\_set\_key(client, addr[0]))  
  
 continue  
  
 else:  
 if current\_socket in self.open\_clients:  
 try:  
 data\_len = int(current\_socket.recv(3).decode())  
 data = current\_socket.recv(data\_len).decode()  
 except Exception as e:  
 print(**"main server in server comm"** + str(e))  
 self.\_disconnect\_client(current\_socket)  
 else:  
 if data == **''**:  
 self.\_disconnect\_client(current\_socket)  
 else:  
  
 self.rcv\_q.put((self.open\_clients[current\_socket][0], self.open\_clients[current\_socket][1].decrypt(data)))  
  
  
 def \_disconnect\_client(self, client):  
 '''  
 gets client to  
 :param client:  
 :return:  
 '''  
 if client in self.open\_clients.keys():  
 print(**f'**{self.open\_clients[client]} **- disconnected'**)  
 self.rcv\_q.put((self.open\_clients[client][0], **"99"**))  
 del self.open\_clients[client]  
  
 def \_find\_socket\_by\_ip(self, ip):  
 '''  
 gets ip and returns the socket matched  
 :param ip: ip got (string)  
 :return: socket matched (socket)  
 '''  
 client = None  
 for soc in self.open\_clients.keys():  
 if self.open\_clients[soc][0] == ip:  
 client = soc  
 break  
 return client  
  
 def send(self, ip, msg):  
 '''  
 send msg to client  
 :param ip: str  
 :param msg: str  
 :return:  
 '''  
 if self.is\_running:  
 client = self.\_find\_socket\_by\_ip(ip)  
 if client:  
 msg\_encrypted = self.open\_clients[client][1].encrypt(msg)  
 try:  
 client.send(str(len(msg\_encrypted)).zfill(3).encode())  
 client.send(msg\_encrypted)  
  
 except Exception as e:  
 print(**'server comm - send'**, str(e))  
 self.\_disconnect\_client(client)  
  
  
 def sendAll(self,msg):  
 '''  
 send msg to everyone  
 :param msg: str  
 :return:  
 '''  
 if self.is\_running:  
 for client in self.open\_clients.keys():  
 try:  
 client.send(str(len(msg)).zfill(3).encode())  
 client.send(msg.encode())  
 except Exception as e:  
 print(**'server comm - send'**, str(e))  
 self.\_disconnect\_client(client)  
  
 def close\_server(self):  
 self.is\_running = False  
  
 def is\_running(self):  
 return self.is\_running  
  
 def \_set\_key(self, socket, ip):  
 """  
 set key  
 :param socket:  
 :param ip:  
 :return:  
 """  
 random\_num = random.randint(0, self.p - 3) + 1  
 personal\_key = **"50"** + str(self.g \*\* random\_num % self.p)  
 try:  
 socket.send(str(len(personal\_key)).zfill(3).encode())  
 socket.send(personal\_key.encode())  
 except Exception as e:  
 print(**"server comm - set key "**, str(e))  
 try:  
 data\_len = int(socket.recv(3).decode())  
 data = socket.recv(data\_len).decode()  
 except Exception as e:  
 self.\_disconnect\_client(socket)  
 else:  
 key = str(int(data[2:]) \*\* random\_num % self.p)  
 self.open\_clients[socket] = (ip, Encryption(key))  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 msg\_q = queue.Queue()  
 server = ServerComm(4500, msg\_q)  
 print(msg\_q.get())  
 server.send(**'127.0.0.1'**, **"hi"**)

Client protocol:

# client\_protocol - responsible for the use of protocol in the communication with the server  
  
def unpack(data):  
 """  
 unpack the data the server sent occording to the protocol  
 :param data: str  
 :return: opcode and list of params  
 """  
 opcode = data[:2]  
 params = data[2:].split(**","**)  
 return opcode, params  
  
  
def pack\_login(username, password):  
 """  
 pack msg according to the protocol  
 :param username:  
 :param password:  
 :return:  
 """  
 return **f"00**{username}**,**{password}**"**def pack\_signup(username, password, phone\_number):  
 """  
 pack msg according to the protocol  
 :param username:  
 :param password:  
 :param phone\_number  
 :return:  
 """  
 return **f"01**{username}**,**{password}**,**{phone\_number}**"**def pack\_new\_calendar(name, participants):  
 """  
 pack msg according to the protocol  
 :param name:  
 :param participants:  
 :return:  
 """  
 participants = **"^"**.join(participants)  
 return **f"02**{name}**,**{participants}**"**def pack\_new\_event(calendar\_id, name, start, end, date, participants):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param name:  
 :param start:  
 :param end:  
 :param date:  
 :param participants:  
 :return:  
 """  
 participants = **"^"**.join(participants)  
 return **f"04**{calendar\_id}**,**{name}**,**{start}**,**{end}**,**{date}**,**{participants}**"**def pack\_calendar\_invitation(calendar\_id, username):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param username:  
 :return:  
 """  
 return **f"10**{calendar\_id}**,**{username}**"**def pack\_calendar\_response(status, calendar\_id):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param calendar\_id:  
 :return:  
 """  
 return **f"11**{status}**,**{calendar\_id}**"**def pack\_event\_invitation(calendar\_id, event\_id, username):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param event\_id:  
 :param username:  
 :return:  
 """  
 return **f"12**{calendar\_id}**,**{event\_id}**,**{username}**"**def pack\_event\_response(status, event\_id):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param event\_id:  
 :return:  
 """  
 return **f"13**{status}**,**{event\_id}**"**def pack\_invitations\_request():  
 """  
 pack msg according to the protocol  
 :return:  
 """  
 return **"14"**def pack\_calendar\_name\_edit(calendar\_id, name):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param name:  
 :return:  
 """  
 return **f"20**{calendar\_id}**,**{name}**"**def pack\_event\_name\_edit(event\_id, name):  
 """  
 pack msg according to the protocol  
 :param event\_id:  
 :param name:  
 :return:  
 """  
 return **f"21**{event\_id}**,**{name}**"**def pack\_event\_time\_edit(event\_id, start, end, date):  
 """  
 pack msg according to the protocol  
 :param event\_id:  
 :param start:  
 :param end:  
 :param date:  
 :return:  
 """  
 return **f"22**{event\_id}**,**{start}**,**{end}**,**{date}**"**def pack\_event\_delete(event\_id):  
 """  
 pack msg according to the protocol  
 :param event\_id:  
 :return:  
 """  
 return **f"30**{event\_id}**"**def pack\_exit\_calendar(calendar\_id):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :return:  
 """  
 return **f"31**{calendar\_id}**"**def pack\_calendar\_ids():  
 """  
 pack msg according to the protocol  
 :return:  
 """  
 return **"40"**def pack\_day\_event(calendar\_id, date):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param date:  
 :return:  
 """  
 return **f"41**{calendar\_id}**,**{date}**"**def pack\_month\_events(calendar\_id, month, year):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param month:  
 :param year:  
 :return:  
 """  
 return **f"42**{calendar\_id}**,**{month}**,**{year}**"**def pack\_get\_calendar\_info(calendar\_id):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :return:  
 """  
 return **f"43**{calendar\_id}**"**def pack\_key(key):  
 """  
 pack msg according to the protocol  
 :param key:  
 :return:  
 """  
 return **f"50**{key}**"**

Server protocol:

# server\_protocol - responsible for the use of protocol in the communication with the clients  
  
  
def unpack(data):  
 """  
 unpack the data the server sent occording to the protocol  
 :param data: str  
 :return: opcode and list of params  
 """  
 opcode = data[:2]  
 params = data[2:].split(**","**)  
  
 return (opcode, params)  
  
def pack\_login(status):  
 """  
 pack msg according to the protocol  
 :param status:  
 :return:  
 """  
 return **f"00**{status}**"**def pack\_sign\_up(status):  
 """  
 pack msg according to the protocol  
 :param status:  
 :return:  
 """  
 return **f"01**{status}**"**def pack\_new\_calendar(status, data\_or\_not\_existing\_participants):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param data\_or\_not\_existing\_participants:  
 :return:  
 """  
 if status == **"0"** or status == **"9"**:  
 for i in range(len(data\_or\_not\_existing\_participants[3])):  
 data\_or\_not\_existing\_participants[3][i] = **"$"**.join(data\_or\_not\_existing\_participants[3][i])  
 data\_or\_not\_existing\_participants[3] = **"\*"**.join(data\_or\_not\_existing\_participants[3])  
 print(**"pack new calendar"**, data\_or\_not\_existing\_participants)  
 data\_or\_not\_existing\_participants = **"^"**.join(data\_or\_not\_existing\_participants)  
  
 return **f"02**{status}**,**{data\_or\_not\_existing\_participants}**"**def pack\_new\_event(status, id\_or\_not\_existing\_participants):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param id\_or\_not\_existing\_participants:  
 :return:  
 """  
 if status == **"1"**:  
 if id\_or\_not\_existing\_participants and id\_or\_not\_existing\_participants != **"user"**:  
 id\_or\_not\_existing\_participants = **"^"**.join(id\_or\_not\_existing\_participants)  
  
 return **f"04**{status}**,**{id\_or\_not\_existing\_participants}**"**def pack\_event\_info(date, color):  
 """  
 pack msg according to the protocol  
 :param date:  
 :param color:  
 :return:  
 """  
 return **f"05**{date}**,**{color}**"**def pack\_there\_is\_an\_invitation():  
 """  
 pack msg according to the protocol  
 :return:  
 """  
 return **f"10"**def pack\_new\_calendar\_participant(status, calendar\_id, username):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param calendar\_id:  
 :param username:  
 :return:  
 """  
 return **f"11**{status}**,**{calendar\_id}**,**{username}**"**def pack\_event\_invitation\_succeed(status):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param username:  
 :param name\_event:  
 :param name\_calendar:  
 :return:  
 """  
 return **f"13**{status}**"**def pack\_invitations(invitations):  
 """  
 pack msg according to the protocol  
 :param invitations:  
 :return:  
 """  
 if not invitations:  
 day\_events = []  
 for i in invitations:  
 i[2] = **"$"**.join(i[2])  
  
 invitations = [**"^"**.join(x) for x in invitations]  
 invitations = **"\*"**.join(invitations)  
 return **f"14**{invitations}**"**def pack\_is\_calendar\_invitation\_work(status):  
 """  
 pack msg according to the protocol  
 :param status:  
 :return:  
 """  
 return **f"15**{status}**"**def pack\_calendar\_name\_edit(status, calendar\_id, name):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param calendar\_id:  
 :param name:  
 :return:  
 """  
 return **f"20**{status}**,**{calendar\_id}**,**{name}**"**def pack\_event\_name\_edit(status, event\_id , name):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param event\_id:  
 :param name:  
 :return:  
 """  
 return **f"21**{status}**,**{event\_id}**,**{name}**"**def pack\_time\_edit(status, calendar\_id, event\_id, time\_or\_participants):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param calendar\_id:  
 :param event\_id:  
 :param time\_or\_participants:  
 :return:  
 """  
  
 time\_or\_participants = **"^"**.join(time\_or\_participants)  
 return **f"22**{status}**,**{calendar\_id}**,**{event\_id}**,**{time\_or\_participants}**"**def pack\_event\_dalete(status, event\_id):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param event\_id:  
 :return:  
 """  
 return **f"30**{status}**,**{event\_id}**"**def pack\_delete\_calendar(status, calendar\_id):  
 """  
 pack msg according to the protocol  
 :param status:  
 :param calendar\_id:  
 :return:  
 """  
 return **f"31**{status}**,**{calendar\_id}**"**def pack\_exit\_calendar(calendar\_id, username):  
 """  
 pack msg according to the protocol  
 :param calendar\_id:  
 :param username:  
 :return:  
 """  
 return **f"32**{calendar\_id}**,**{username}**"**def pack\_calendar\_ids(calendar\_ids):  
 """  
 pack msg according to the protocol  
 :param calendar\_ids:  
 :return:  
 """  
 calendar\_ids = **"^"**.join(calendar\_ids)  
 return **f"40**{calendar\_ids}**"**def pack\_day\_events(day\_events):  
 """  
 pack msg according to the protocol  
 :param day\_events:  
 :return:  
 """  
 if not day\_events:  
 day\_events = []  
 for i in day\_events:  
 i[0] = **"$"**.join(i[0])  
  
 day\_events = [**"^"**.join(x) for x in day\_events]  
 day\_events = **"\*"**.join(day\_events)  
 print(**f"hello** {day\_events}**"**)  
 return **f"41**{day\_events}**"**def pack\_month\_events(month\_events):  
 """  
 pack msg according to the protocol  
 :param month\_events:  
 :return:  
 """  
 if not month\_events:  
 month\_events = []  
 month\_events = [**"^"**.join(x) for x in month\_events]  
 month\_events = **"\*"**.join(month\_events)  
 return **f"42**{month\_events}**"**def pack\_key(key):  
 """  
 pack msg according to the protocol  
 :param key:  
 :return:  
 """  
 return **f"50**{key}**"**

Encryption:

# Encryption - responsible for the encryption and decryption of the messages between the server and the clients  
  
import base64  
from Cryptodome.Cipher import AES  
from Cryptodome import Random  
import hashlib  
  
class Encryption:  
 def \_\_init\_\_(self, key):  
 self.bs = AES.block\_size  
 self.key = hashlib.sha256(key.encode()).digest()  
  
 def encrypt(self, raw):  
 """  
 encrypt msg  
 :param raw:  
 :return:  
 """  
 raw = self.\_pad(raw)  
 iv = Random.new().read(AES.block\_size)  
 cipher = AES.new(self.key, AES.MODE\_CBC, iv)  
 return base64.b64encode(iv + cipher.encrypt(raw.encode()))  
  
 def decrypt(self, enc):  
 """  
 decrypt msg  
 :param enc:  
 :return:  
 """  
 enc = base64.b64decode(enc)  
 iv = enc[:AES.block\_size]  
 cipher = AES.new(self.key, AES.MODE\_CBC, iv)  
 return self.\_unpad(cipher.decrypt(enc[AES.block\_size:])).decode(**'utf-8'**)  
  
 def \_pad(self, s):  
 """  
 pad msg  
 :param s:  
 :return:  
 """  
 return s + (self.bs - len(s) % self.bs) \* chr(self.bs - len(s) % self.bs)  
  
  
 @staticmethod  
 def \_unpad(s):  
 """  
 unpad mag  
 :param s:  
 :return:  
 """  
 return s[:-ord(s[len(s)-1:])]  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
  
 cry = Encryption(**"yuval"**)  
 length = cry.encrypt(**"5"**)  
 print(length)  
 print(cry.decrypt(length))  
 msg = cry.encrypt(**"My name is yuval"**)  
 print(msg)  
 print(cry.decrypt(msg))

Data base:

# joined\_calendar\_db - responsible for the data base  
  
import sqlite3  
from datetime import date  
import hashlib  
  
class joined\_calendar\_db:  
  
 def \_\_init\_\_(self):  
 self.db\_conn = None  
 self.db\_cursor = None  
 self.name\_of\_db = **"joined\_calendar\_data"** self.calendars = **"calendars"** self.users = **"users"** self.calendars\_participants = **"calendars\_participants"** self.events\_participants = **"events\_participants"** self.event\_info = **"event\_info"** self.calendar\_invitations = **"calendar\_invitations"** self.event\_invitations = **"event\_invitations"** self.reminders = **"reminders"** self.colors = [**'BLUE'**, **'GREEN'**, **'PURPLE'**, **'PINK'**, **'GREY'**, **'ORANGE'**, **'BROWN'**, **'GOLD'**, **'TURQUOISE'**, **'LIGHT BLUE'**, **'LIME GREEN'**, **'SALMON'**, **'DARK OLIVE GREEN'**]  
 self.joined\_color = **'RED'** self.\_create\_db()  
  
 def \_create\_db(self):  
 """  
 connect to the db and create the tables that dont exists  
 :return: None  
 """  
 self.db\_conn = sqlite3.connect(self.name\_of\_db)  
 self.db\_cursor = self.db\_conn.cursor()  
 sql\_table = [**"CREATE TABLE IF NOT EXISTS "** + self.users +  
 **"(username VARCHAR(20), password VARCHAR(20), phone\_number VARCHAR(10), PRIMARY KEY(username));"**,  
 **"CREATE TABLE IF NOT EXISTS "** + self.calendars +  
 **" (calendar\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), manager VARCHAR(20))"**,  
  
 **"CREATE TABLE IF NOT EXISTS "** + self.calendars\_participants +  
 **" (calendar\_id VARCHAR(3), participant VARCHAR(20), color VARCHAR(20), PRIMARY KEY(calendar\_id, participant))"**,  
 **"CREATE TABLE IF NOT EXISTS "** + self.events\_participants +  
 **" (event\_id VARCHAR(4), participant VARCHAR(20), PRIMARY KEY(event\_id, participant))"**,  
 **"CREATE TABLE IF NOT EXISTS "** + self.event\_info +  
 **" (event\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, calendar\_id VARCHAR(3), name VARCHAR(30), manager VARCHAR(20), start\_hour VARCHAR(5), end\_hour VARCHAR(5), date VARCHAR(10))"**,  
 **"CREATE TABLE IF NOT EXISTS "** + self.calendar\_invitations +  
 **" (username VARCHAR(20), invited\_by VARCHAR(20), calendar\_id VARCHAR(3), PRIMARY KEY(username, invited\_by, calendar\_id))"**,  
 **"CREATE TABLE IF NOT EXISTS "** + self.event\_invitations +  
 **" (username VARCHAR(20), invited\_by VARCHAR(20), calendar\_id VARCHAR(3), event\_id VARCHAR(4), PRIMARY KEY(username, invited\_by, event\_id))"**,  
 **"CREATE TABLE IF NOT EXISTS "** + self.reminders +  
 **" (username VARCHAR(20), event\_id VARCHAR(4), date VARCHAR(10), time VARCHAR(5), PRIMARY KEY(username, event\_id))"**]  
  
  
 [self.db\_cursor.execute(x) for x in sql\_table]  
 self.db\_conn.commit()  
  
 def \_is\_calendar\_exists(self, calendar\_id):  
 """  
 check if calendar exists  
 :param calendar\_id: str  
 :return: bool  
 """  
 sql\_table = **"SELECT calendar\_id FROM "** + self.calendars + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql\_table, (calendar\_id,))  
 return self.db\_cursor.fetchone() is not None  
  
 def \_is\_event\_exists(self, id):  
 """  
 check if event exists  
 :param id: str  
 :return: bool  
 """  
 sql\_table = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql\_table, (id,))  
 return self.db\_cursor.fetchone() is not None  
  
 def is\_user\_exists(self, user):  
 """  
 check if user exists  
 :param user: str  
 :return: bool  
 """  
 sql\_table = **"SELECT username FROM "** + self.users + **" WHERE username = ?"** self.db\_cursor.execute(sql\_table, (user,))  
 user = self.db\_cursor.fetchone()  
 return user is not None  
  
 def add\_calendar(self, name, manager):  
 """  
 add calendar to calendars table and calendar\_participants table, increase last\_calendar\_id  
 :param name: str  
 :param manager: str  
 :return: id(str) or empty str if already exists  
 """  
 sql\_table = **"INSERT INTO "** + self.calendars + **"(name,manager) VALUES(?,?)"** self.db\_cursor.execute(sql\_table, (name, manager,))  
 self.db\_conn.commit()  
  
 sql = **"SELECT calendar\_id from "** + self.calendars + **" ORDER BY calendar\_id DESC LIMIT (1)"** self.db\_cursor.execute(sql)  
 id = str(self.db\_cursor.fetchone()[0])  
 self.add\_calendar\_participant(id, manager)  
 return id  
  
  
 def add\_calendar\_participant(self, id, user):  
 """  
 if calendar exists, add participant to calendar\_participants table  
 :param id: str  
 :param user: str  
 :return:  
 """  
 flag = False  
 if self.\_is\_calendar\_exists(id):  
  
 # check if participant is not already exists  
 if not self.is\_participant\_exists\_in\_calendar(id, user):  
  
 sql\_table = **"INSERT INTO "** + self.calendars\_participants + **" VALUES(?,?,?)"** self.db\_cursor.execute(sql\_table, (id, user, self.find\_color(id),))  
 self.db\_conn.commit()  
 flag = True  
 else:  
 print(**"already is a participant"**)  
 else:  
 print(**"calendar do not exists so cant add participant"**)  
 return flag  
  
 def find\_color(self, id):  
 """  
 find color that is mot occupied in the calendar  
 :param id: str  
 :return: color(str) if the calendar is not full  
 """  
 sql\_table = **"SELECT color FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql\_table, (id,))  
 current\_colors = self.db\_cursor.fetchall()  
 current\_colors = [i[0] for i in current\_colors]  
 color = **""** for c in self.colors:  
 if c not in current\_colors:  
 color = c  
 break  
 return color  
  
 def add\_user(self, username, password, phone):  
 """  
 add username to users table if user is not exists  
 :param username: str  
 :param password: str  
 :param phone: str  
 :return: if user not exists (bool)  
 """  
  
 flag = self.is\_user\_exists(username)  
  
 if not flag:  
 sql\_table = **"INSERT INTO "** + self.users + **" VALUES(?,?,?)"** self.db\_cursor.execute(sql\_table, (username, hashlib.sha256(password.encode()).digest(), phone,))  
 self.db\_conn.commit()  
  
 return not flag  
  
 def is\_participant\_exists\_in\_calendar(self, id, user):  
 """  
 check if the user alredy exists in the calendar  
 :param id:  
 :param user:  
 :return:  
 """  
 sql\_table = **"SELECT participant FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql\_table, (id,))  
 participants = self.db\_cursor.fetchall()  
 participants = [i[0] for i in participants]  
  
 return user in participants  
  
  
 def add\_calendar\_invitation(self, id, invited\_by, invited):  
 """  
 add calendar invitation to the table  
 :param id:  
 :param invited\_by:  
 :param invited:  
 :return:  
 """  
 flag = False  
 if self.\_is\_calendar\_exists(id) and self.is\_user\_exists(invited) and self.is\_user\_exists(invited\_by):  
 if not self.is\_calendar\_invitation\_exists(id, invited):  
 sql\_table = **"INSERT INTO "** + self.calendar\_invitations + **" VALUES(?,?,?)"** self.db\_cursor.execute(sql\_table, (invited, invited\_by, id,))  
 self.db\_conn.commit()  
 flag = True  
 else:  
 print(**"invitation already exists"**)  
 else:  
 print(**"the calendar doesnt exists so cant add invitation or user not exists"**)  
 return flag  
  
 def is\_calendar\_invitation\_exists(self, id, user):  
 """  
 check if invitation exists in the table  
 :param id:  
 :param user:  
 :return:  
 """  
 sql\_table = **"SELECT username FROM "** + self.calendar\_invitations + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql\_table, (id,))  
 invitations = self.db\_cursor.fetchall()  
 invitations = [i[0] for i in invitations]  
  
 return user in invitations  
  
 def add\_event(self, name, participants, calendar\_id, manager, start, end, date):  
 """  
 add event to the table and add the invitations to the invitations table if time available  
 :param name: str  
 :param participants: list of participants  
 :param calendar\_id:  
 :param manager:  
 :param start:  
 :param end:  
 :param date:  
 :return:  
 """  
 event\_id = **""** if self.\_is\_calendar\_exists(calendar\_id):  
 if self.is\_participants\_in\_calendar(participants, calendar\_id):  
 if self.check\_is\_time\_available(manager, start, end, date):  
 sql = **"INSERT INTO "** + self.event\_info + **"(calendar\_id, name, manager, start\_hour, end\_hour, date) VALUES(?,?,?,?,?,?)"** self.db\_cursor.execute(sql, (calendar\_id, name, manager, start, end, date,))  
 self.db\_conn.commit()  
 sql = **"SELECT event\_id from "** + self.event\_info + **" ORDER BY event\_id DESC LIMIT (1)"** self.db\_cursor.execute(sql)  
 event\_id = self.db\_cursor.fetchone()[0]  
 self.add\_event\_participant(event\_id, manager)  
 participants = participants[1:]  
 for user in participants:  
 self.add\_event\_invitation(event\_id, calendar\_id, user, manager)  
 else:  
 print(**"time is not available"**)  
 else:  
 print(**"errorrr event participants are not calendar participants"**)  
 else:  
 print(**"calendar do not exists so cant add event"**)  
 return event\_id  
  
 def add\_event\_participant(self, id, participant):  
 """  
 add participant to event in events\_participants table  
 :param id:  
 :param participant:  
 :return:  
 """  
 flag = False  
 if self.\_is\_event\_exists(id):  
  
 # check if participant is not already exists  
 if not self.is\_participant\_exists\_in\_event(id, participant):  
  
 sql\_table = **"INSERT INTO "** + self.events\_participants + **" VALUES(?,?)"** self.db\_cursor.execute(sql\_table, (id, participant,))  
 self.db\_conn.commit()  
 flag = True  
 else:  
 print(**"errorrr already is a participant"**)  
 else:  
 print(**"event do not exists so cant add participant"**)  
 return flag  
  
 def is\_participant\_exists\_in\_event(self, id, username):  
 """  
 check if user is a participant in the event  
 :param id:  
 :param username:  
 :return:  
 """  
 sql\_table = **"SELECT participant FROM "** + self.events\_participants + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql\_table, (id,))  
 participants = self.db\_cursor.fetchall()  
 participants = [i[0] for i in participants]  
  
 return username in participants  
  
 def get\_calendar\_participants(self, id):  
 """  
 return calendar participants  
 :param id:  
 :return: list  
 """  
 sql = **"SELECT participant FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 participants = self.db\_cursor.fetchall()  
 participants = [i[0] for i in participants]  
 return participants  
  
 def get\_calendar\_participants\_plus\_colors(self, id):  
 """  
 return calendar participants with their colors  
 :param id:  
 :return: list  
 """  
 sql = **"SELECT participant FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 participants = self.db\_cursor.fetchall()  
 participants = [i[0] for i in participants]  
 p = []  
 for i in participants:  
 p.append([i, self.get\_color(i, id)])  
 return p  
  
 def is\_participants\_in\_calendar(self, participants, id):  
 """  
 check if the usernames in the list are participants in the calendar  
 :param participants:  
 :param id:  
 :return: bool  
 """  
 flag = True  
 calendar\_participants = self.get\_calendar\_participants(id)  
 for x in participants:  
 if not x in calendar\_participants:  
 flag = False  
 break  
 return flag  
  
 def check\_is\_time\_available(self, username, start, end, date):  
 """  
 check if the username has free time  
 :param username:  
 :param start:  
 :param end:  
 :param date:  
 :return: bool  
 """  
 flag = False  
 if self.is\_user\_exists(username):  
 sql = **"SELECT start\_hour, end\_hour FROM "** + self.event\_info + **" WHERE manager = ? AND date = ?"** self.db\_cursor.execute(sql, (username, date,))  
 hours = self.db\_cursor.fetchall()  
 hours = [i for i in hours]  
 start = int(start.replace(**":"**, **""**))  
 end = int(end.replace(**":"**, **""**))  
 for x in hours:  
 if (start >= int(x[0].replace(**":"**, **""**)) and start < int(x[1].replace(**":"**, **""**))) or (end > int(x[0].replace(**":"**, **""**)) and end <= int(x[1].replace(**":"**, **""**))):  
 break  
 else:  
 flag = True  
 else:  
 print(**"user do not exists so cant check time availability"**)  
 return flag  
  
 def events\_in\_time(self, username, start, end, date):  
 """  
 return events in time  
 :param username:  
 :param start:  
 :param end:  
 :param date:  
 :return:  
 """  
 events = []  
 if self.is\_user\_exists(username):  
 sql = **"SELECT start\_hour, end\_hour, event\_id FROM "** + self.event\_info + **" WHERE manager = ? AND date = ?"** self.db\_cursor.execute(sql, (username, date,))  
 hours = self.db\_cursor.fetchall()  
 hours = [i for i in hours]  
 start = int(start.replace(**":"**, **""**))  
 end = int(end.replace(**":"**, **""**))  
 for x in hours:  
 if (start >= int(x[0].replace(**":"**, **""**)) and start < int(x[1].replace(**":"**, **""**))) or (  
 end > int(x[0].replace(**":"**, **""**)) and end <= int(x[1].replace(**":"**, **""**))):  
 events.append(str(x[2]))  
  
 else:  
 print(**"user do not exists so cant check time availability"**)  
 return events  
  
 def add\_event\_invitation(self, event\_id, calendar\_id, invited, invited\_by):  
 """  
  
 :param event\_id:  
 :param calendar\_id:  
 :param invited:  
 :param invited\_by:  
 :return:  
 """  
 flag = False  
 if self.\_is\_event\_exists(event\_id) and self.is\_user\_exists(invited) and self.is\_user\_exists(invited\_by):  
 if not self.is\_event\_invitation\_exists(event\_id, invited) and self.is\_participant\_exists\_in\_calendar(calendar\_id, invited):  
 sql\_table = **"INSERT INTO "** + self.event\_invitations + **" VALUES(?,?,?,?)"** self.db\_cursor.execute(sql\_table, (invited, invited\_by, calendar\_id, event\_id,))  
 self.db\_conn.commit()  
 flag = True  
 else:  
 print(**"invitation already exists"**)  
 else:  
 print(**"the event doesnt exists so cant add invitation or user not exists"**)  
 return flag  
  
 def is\_event\_invitation\_exists(self, event\_id, invited):  
 """  
 check if invitation already exists  
 :param event\_id:  
 :param invited:  
 :return:  
 """  
 sql\_table = **"SELECT username FROM "** + self.event\_invitations + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql\_table, (event\_id,))  
 invitations = self.db\_cursor.fetchall()  
 invitations = [i[0] for i in invitations]  
  
 return invited in invitations  
  
 def is\_manager\_calander(self, id, user):  
 """  
 check if user is manager  
 :param id:  
 :param user:  
 :return:  
 """  
 sql = **"SELECT manager FROM "** + self.calendars + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 manager = self.db\_cursor.fetchone()[0]  
  
 return manager == user  
  
 def is\_manager\_event(self, id, user):  
 """  
 check if user is manager  
 :param id:  
 :param user:  
 :return:  
 """  
 print(id)  
 sql = **"SELECT manager FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 manager = self.db\_cursor.fetchone()[0]  
  
 return manager == user  
  
 def get\_password(self, username):  
 """  
 return password  
 :param username:  
 :return:  
 """  
  
 sql = **"SELECT password FROM "** + self.users + **" WHERE username = ?"** self.db\_cursor.execute(sql, (username,))  
 password = self.db\_cursor.fetchone()[0]  
 if password == None:  
 password = **"-1"** return password  
  
 def change\_calendar\_name(self, id, name):  
 """  
 change the calendar name  
 :param id:  
 :param name:  
 :return:  
 """  
 sql = **"UPDATE "** + self.calendars + **" SET name = ?"** + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (name, id,))  
 self.db\_conn.commit()  
  
  
 def change\_event\_name(self, id, name):  
 """  
 change the event name  
 :param id:  
 :param name:  
 :return:  
 """  
 sql = **"UPDATE "** + self.event\_info + **" SET name = ? WHERE event\_id = ?"** self.db\_cursor.execute(sql, (name, id,))  
 self.db\_conn.commit()  
  
  
 def change\_event\_time(self, id, start, end, date):  
 """  
 change event time  
 :param id:  
 :param start:  
 :param end:  
 :param date:  
 :return:  
 """  
 sql = **"UPDATE "** + self.event\_info + **" SET start\_hour = ? , end\_hour = ? , date = ? WHERE event\_id = ?"** self.db\_cursor.execute(sql, (start, end, date, id,))  
 self.db\_conn.commit()  
  
  
 def add\_reminder(self, time, date, event\_id, username):  
 """  
 add reminder to reminder table  
 :param time:  
 :param date:  
 :param event\_id:  
 :param username:  
 :return:  
 """  
 sql\_table = **"INSERT INTO "** + self.reminders + **" VALUES(?,?,?,?)"** self.db\_cursor.execute(sql\_table, (username, event\_id, date, time,))  
 self.db\_conn.commit()  
  
 def get\_today\_reminders(self):  
 """  
 return list of all the reminders at the same day and delete them from the reminders table  
 :return: list of all the reminders at the same day  
 """  
 today = date.today()  
 today = today.strftime(**"%d.%m.%Y"**)  
 sql = **"SELECT username, event\_id, time FROM "** + self.reminders + **" WHERE date = ?"** self.db\_cursor.execute(sql, (today,))  
 reminders = self.db\_cursor.fetchall()  
 sql = **"DELETE FROM "** + self.reminders + **" WHERE date = ?"** self.db\_cursor.execute(sql, (today,))  
 self.db\_conn.commit()  
  
  
 return reminders  
  
 def delete\_reminder\_for\_user(self, username, event\_id):  
 """  
 delete the reminder from reminders table  
 :param username:  
 :param event\_id:  
 :return:  
 """  
 sql = **"DELETE FROM "** + self.reminders + **" WHERE username = ? And event\_id = ?"** self.db\_cursor.execute(sql, (username, event\_id,))  
 self.db\_conn.commit()  
  
 def delete\_reminder\_for\_all(self, event\_id):  
 """  
 delete all reminders for this event from reminders table  
 :param event\_id:  
 :return:  
 """  
 sql = **"DELETE FROM "** + self.reminders + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (event\_id,))  
 self.db\_conn.commit()  
  
 def get\_event\_participants(self, id):  
 """  
 return event participants  
 :param id:  
 :return: list  
 """  
 sql = **"SELECT participant FROM "** + self.events\_participants + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 participants = self.db\_cursor.fetchall()  
 participants = [i[0] for i in participants]  
 return participants  
  
 def exit\_calendar(self, calendar\_id, username):  
 """  
 remove user from calendar  
 :param calendar\_id:  
 :param username:  
 :return: status of what happened  
 """  
 status = 4  
 if self.\_is\_calendar\_exists(calendar\_id) and self.is\_user\_exists(username) and self.is\_participant\_exists\_in\_calendar(calendar\_id, username):  
 if self.is\_manager\_calander(calendar\_id, username):  
 status = 1  
 # delete calendar  
 sql = [**"DELETE FROM "** + self.calendar\_invitations + **" WHERE calendar\_id = ?"**,  
 **"DELETE FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ?"**,  
 **"DELETE FROM "** + self.calendars + **" WHERE calendar\_id = ?"**]  
 [self.db\_cursor.execute(x, (calendar\_id,)) for x in sql]  
 self.db\_conn.commit()  
  
 sql = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (calendar\_id,))  
 events = self.db\_cursor.fetchall()  
 events = [x[0] for x in events]  
 for x in events:  
 sql = [**"DELETE FROM "** + self.event\_invitations + **" WHERE event\_id = ?"**,  
 **"DELETE FROM "** + self.events\_participants + **" WHERE event\_id = ?"**,  
 **"DELETE FROM "** + self.reminders + **" WHERE event\_id = ?"**]  
 [self.db\_cursor.execute(s, (x,)) for s in sql]  
 self.db\_conn.commit()  
  
 sql = **"DELETE FROM "** + self.event\_info + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (calendar\_id,))  
 self.db\_conn.commit()  
  
 else:  
 status = 2  
 # remove from calendar  
  
 sql = **"DELETE FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ? AND participant = ?"** self.db\_cursor.execute(sql, (calendar\_id, username, ))  
 self.db\_conn.commit()  
  
 sql = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE calendar\_id = ? AND manager = ?"** self.db\_cursor.execute(sql, (calendar\_id, username,))  
 events = self.db\_cursor.fetchall()  
 events = [x[0] for x in events]  
 if len(events) != 0:  
 status = 3  
 for x in events:  
 sql = [**"DELETE FROM "** + self.events\_participants + **" WHERE event\_id = ? AND participant = ?"**,  
 **"DELETE FROM "** + self.reminders + **" WHERE event\_id = ? AND username = ?"**]  
 [self.db\_cursor.execute(s, (x, username,)) for s in sql]  
 self.db\_conn.commit()  
 sql = **"DELETE FROM "** + self.event\_invitations + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (calendar\_id,))  
 self.db\_conn.commit()  
  
 sql = **"DELETE FROM "** + self.event\_info + **" WHERE calendar\_id = ? AND manager = ?"** self.db\_cursor.execute(sql, (calendar\_id, username,))  
 self.db\_conn.commit()  
 return status  
  
 def delete\_event(self, event\_id, username):  
 """  
 delete event  
 :param event\_id:  
 :param username:  
 :return:  
 """  
 flag = self.is\_manager\_event(event\_id, username)  
 if flag:  
 sql = [**"DELETE FROM "** + self.event\_info + **" WHERE event\_id = ?"**,  
 **"DELETE FROM "** + self.event\_invitations + **" WHERE event\_id = ?"**,  
 **"DELETE FROM "** + self.events\_participants + **" WHERE event\_id = ?"**]  
 [self.db\_cursor.execute(s, (event\_id,)) for s in sql]  
 self.db\_conn.commit()  
 return flag  
  
 def get\_manager\_calander(self, id):  
 """  
 get the manager of the calendar  
 :param id:  
 :return: manager  
 """  
 sql = **"SELECT manager FROM "** + self.calendars + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 manager = self.db\_cursor.fetchone()[0]  
  
 return manager  
  
 def get\_manager\_event(self, id):  
 """  
 get the manager of the event  
 :param id:  
 :return: manager  
 """  
 sql = **"SELECT manager FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (id,))  
 manager = self.db\_cursor.fetchone()[0]  
  
 return manager  
  
 def get\_calendar\_invitations(self, username):  
 """  
 get all the user's calendar invitations  
 :param username:  
 :return: list of all the invitations  
 """  
  
 sql = **"SELECT invited\_by, calendar\_id FROM "** + self.calendar\_invitations + **" WHERE username = ?"** self.db\_cursor.execute(sql, (username,))  
 invitations = self.db\_cursor.fetchall()  
 invitations = [x for x in invitations]  
 final = []  
 participants = []  
 sql2 = **"SELECT name FROM "** + self.calendars + **" WHERE calendar\_id = ?"** sql = **"SELECT participant FROM "** + self.calendars\_participants + **" WHERE calendar\_id = ?"** for i in invitations:  
 self.db\_cursor.execute(sql, (i[1],))  
 participants = self.db\_cursor.fetchall()  
 participants = [x[0] for x in participants]  
 self.db\_cursor.execute(sql2, (i[1],))  
 name = self.db\_cursor.fetchone()[0]  
 final.append([name, i[0], participants, i[1]])  
  
 return final  
  
 def get\_event\_invitations(self, username):  
 """  
 get all the user's event invitations  
 :param username:  
 :return: list of all the invitations  
 """  
  
 sql = **"SELECT invited\_by, event\_id, calendar\_id FROM "** + self.event\_invitations + **" WHERE username = ?"** self.db\_cursor.execute(sql, (username,))  
 invitations = self.db\_cursor.fetchall()  
 invitations = [x for x in invitations]  
 final = []  
 participants = []  
 sql2 = **"SELECT name, date, start\_hour, end\_hour FROM "** + self.event\_info + **" WHERE event\_id = ?"** sql = **"SELECT participant FROM "** + self.events\_participants + **" WHERE event\_id = ?"** for i in invitations:  
 self.db\_cursor.execute(sql, (i[1],))  
 participants = self.db\_cursor.fetchall()  
 participants = [x[0] for x in participants]  
 self.db\_cursor.execute(sql2, (i[1],))  
 name, date, start, end = self.db\_cursor.fetchone()  
 final.append([name, i[0], participants, date, start, end, i[1]])  
 return final  
  
 def delete\_calendar\_invitation(self, username, id):  
 """  
 delete calendar invitation  
 :param username:  
 :param id:  
 :return:  
 """  
 if self.is\_calendar\_invitation\_exists(id, username):  
 sql = **"DELETE FROM "** + self.calendar\_invitations + **" WHERE username = ? AND calendar\_id = ?"** self.db\_cursor.execute(sql, (username, id,))  
 self.db\_conn.commit()  
 else:  
 print(**"error - calendar invitation doesnt exists"**)  
  
 def get\_personal\_calendar(self, username):  
 """  
 return calendar\_id of personal calendar  
 :param username:  
 :return:  
 """  
 calendar\_id = **""** if self.is\_user\_exists(username):  
 sql = **"SELECT calendar\_id FROM "** + self.calendars + **" WHERE manager = ? AND name = ?"** self.db\_cursor.execute(sql, (username, **"personal"**))  
 calendar\_id = self.db\_cursor.fetchone()[0]  
 return calendar\_id  
  
  
 def delete\_event\_invitation(self, username, id):  
 """  
 delete calendar invitation  
 :param username:  
 :param id:  
 :return:  
 """  
 if self.is\_event\_invitation\_exists(id, username):  
 sql = **"DELETE FROM "** + self.event\_invitations + **" WHERE username = ? AND event\_id = ?"** self.db\_cursor.execute(sql, (username, id,))  
 self.db\_conn.commit()  
 else:  
 print(**"error - event invitation doesnt exists"**)  
  
 def get\_user\_calendars(self, username):  
 """  
 get list of username calendars ids  
 :param username:  
 :return:  
 """  
 sql = **"SELECT calendar\_id FROM "** + self.calendars\_participants + **" WHERE participant = ?"** self.db\_cursor.execute(sql, (username,))  
 calendars = self.db\_cursor.fetchall()  
 calendars = [x[0] for x in calendars]  
 personal\_id = str(self.get\_personal\_calendar(username))  
 calendars.remove(personal\_id)  
 calendars = [personal\_id] + calendars  
 return calendars  
  
 def get\_events\_of\_calendar(self, calendar\_id, month, year):  
 """  
 get list that contains date and color of all the events in this month that users from the calendar are participating  
 :param calendar\_id:  
 :param month:  
 :param year:  
 :return:  
 """  
 events = []  
 if self.\_is\_calendar\_exists(calendar\_id):  
 participants = self.get\_calendar\_participants(calendar\_id)  
 month\_year = str(month) + **"."** + str(year)  
 sql = **f"SELECT event\_id, date FROM** {self.event\_info} **WHERE date LIKE ?"** wildcard\_month\_year = **'%'** + month\_year  
 self.db\_cursor.execute(sql, (wildcard\_month\_year,))  
 temp = self.db\_cursor.fetchall()  
 ids = []  
 dates = []  
 set\_participants = set(participants)  
 for t in temp:  
 ids += [t[0]]  
 dates += [t[1]]  
 for id in ids:  
 p = self.get\_event\_participants(id)  
 set\_p = set(p)  
 both = set\_p & set\_participants  
 if both:  
 date = dates[ids.index(id)]  
 if len(both) > 1:  
 temp = [date, self.joined\_color]  
 else:  
 both = list(both)  
 temp = [date, self.get\_color(both[0], calendar\_id)]  
  
 for event in events:  
 if event[0] == date:  
 if temp[1] != event[1]:  
 events[events.index(event)] = [event[0], self.joined\_color]  
 else:  
 events[events.index(event)] = [event[0], event[1]]  
  
 break  
 else:  
 events.append(temp)  
  
 return events  
  
 def get\_color(self, username, calendar\_id):  
 """  
 get color of username in calendar  
 :param username:  
 :param calendar\_id:  
 :return:  
 """  
 sql = **"SELECT color FROM "** + self.calendars\_participants + **" WHERE participant = ? AND calendar\_id = ?"** self.db\_cursor.execute(sql, (username, calendar\_id,))  
 return self.db\_cursor.fetchone()[0]  
  
 def get\_day\_events(self, username, calendar\_id, date):  
 """  
 get event info for all the events of the day  
 :param username:  
 :param calendar\_id:  
 :param date:  
 :return:  
 """  
 day, month, year = date.split(**'.'**)  
 day\_events = None  
 day\_ids = []  
 if self.\_is\_calendar\_exists(calendar\_id):  
 participants = self.get\_calendar\_participants(calendar\_id)  
 for p in participants:  
 day\_ids += self.get\_day\_ids(p, date)  
 day\_ids = list(set(day\_ids))  
 day\_events = []  
 for id in day\_ids:  
 day\_events += [self.get\_event\_info(id, username, calendar\_id)]  
 day\_events[-1][-1] = str(day\_events[-1][-1])  
 return day\_events  
  
 def get\_day\_ids(self, participant, date):  
 """  
 get list of this day calendars events ids  
 :param participant:  
 :param date:  
 :return:  
 """  
 sql = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE date = ?"** self.db\_cursor.execute(sql, (date, ))  
 event\_ids = self.db\_cursor.fetchall()  
 event\_ids = set([str(i[0]) for i in event\_ids])  
 sql = **"SELECT event\_id FROM "** + self.events\_participants + **" WHERE participant = ?"** self.db\_cursor.execute(sql, (participant,))  
 event\_ids2 = self.db\_cursor.fetchall()  
 event\_ids2 = set([i[0] for i in event\_ids2])  
 final = list(event\_ids & event\_ids2)  
 return final  
  
 def get\_calendar\_info(self, calendar\_id):  
 """  
 return name, manager and participants  
 :param calendar\_id:  
 :return:  
 """  
 info = []  
 if self.\_is\_calendar\_exists(calendar\_id):  
  
 participants = self.get\_calendar\_participants(calendar\_id)  
 sql = **"SELECT name, manager FROM "** + self.calendars + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (calendar\_id,))  
 name, manager = self.db\_cursor.fetchone()  
  
 info = [name, manager, participants]  
 return info  
  
 def get\_calendar\_info\_with\_colors(self, calendar\_id):  
 """  
 return name, manager and participants with colors  
 :param calendar\_id:  
 :return:  
 """  
 info = []  
 if self.\_is\_calendar\_exists(calendar\_id):  
 participants = self.get\_calendar\_participants\_plus\_colors(calendar\_id)  
 sql = **"SELECT name, manager FROM "** + self.calendars + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (calendar\_id,))  
 name, manager = self.db\_cursor.fetchone()  
  
 info = [name, manager, participants]  
 return info  
  
 def get\_calendar\_name(self, calendar\_id):  
 """  
 return calendar\_id  
 :param calendar\_id:  
 :return:  
 """  
 name = **""** if self.\_is\_calendar\_exists(calendar\_id):  
 sql = **"SELECT name FROM "** + self.calendars + **" WHERE calendar\_id = ?"** self.db\_cursor.execute(sql, (calendar\_id,))  
 name = self.db\_cursor.fetchone()[0]  
 return name  
  
 def get\_day\_color(self, date, calendar\_id):  
 """  
 get day color  
 :param date:  
 :param calendar\_id:  
 :return:  
 """  
 day\_color = []  
 sql = **"SELECT event\_id FROM "** + self.event\_info + **" WHERE date = ?"** self.db\_cursor.execute(sql, (date,))  
 event\_ids = self.db\_cursor.fetchall()  
 event\_ids = [i[0] for i in event\_ids]  
 for id in event\_ids:  
 temp = self.get\_some\_event\_info(id, calendar\_id)  
 if len(temp) != 0:  
 if len(day\_color) == 0:  
 day\_color = temp  
 if temp[1] == **'RED'**:  
 break  
 else:  
 if temp[1] != day\_color[1]:  
 day\_color[1] = **'RED'** break  
 return day\_color  
  
 def get\_event\_date(self, event\_id):  
 """  
 get date of event  
 :param event\_id:  
 :return:  
 """  
 date = **""** if self.\_is\_event\_exists(event\_id):  
 sql = **"SELECT date FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (event\_id,))  
 date = self.db\_cursor.fetchone()[0]  
 return date  
  
 def get\_event\_info(self, event\_id, username, calendar\_id):  
 """  
 get event info - participants, name, manager, start hour, end hour, date  
 or participants, start, end, date  
 :param event\_id:  
 :param username  
 :param calendar\_id:  
 :return:  
 """  
 info = []  
 if self.\_is\_event\_exists(event\_id):  
 participants = list(set(self.get\_event\_participants(event\_id)) & set(self.get\_calendar\_participants(calendar\_id)))  
 if username in participants:  
 sql = **"SELECT name, start\_hour, end\_hour, date, manager, event\_id FROM "** + self.event\_info + **" WHERE event\_id = ?"** else:  
 sql = **"SELECT start\_hour, end\_hour, date, event\_id FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (event\_id,))  
 info = self.db\_cursor.fetchone()  
  
 info = [participants] + list(info)  
  
 return info  
  
 def get\_some\_event\_info(self, event\_id, calendar\_id):  
 """  
 get event color (by participants color) and date  
 :param event\_id:  
 :param calendar\_id  
 :return:  
 """  
 info = []  
 if self.\_is\_calendar\_exists(calendar\_id) and self.\_is\_event\_exists(event\_id):  
 participants = self.get\_calendar\_participants(calendar\_id)  
  
 set\_participants = set(participants)  
 p = self.get\_event\_participants(event\_id)  
 set\_p = set(p)  
 both = set\_p & set\_participants  
 sql = **"SELECT date FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (event\_id,))  
 date = self.db\_cursor.fetchone()[0]  
 if both:  
 if len(both) > 1:  
 info = [date, self.joined\_color]  
 else:  
 both = list(both)  
 info = [date, self.get\_color(both[0], calendar\_id)]  
 return info  
  
  
 def get\_calendar\_from\_event(self, event\_id):  
 """  
 return id of the calendar that the event was created in  
 :param event\_id:  
 :return:  
 """  
 sql = **"SELECT calendar\_id FROM "** + self.event\_info + **" WHERE event\_id = ?"** self.db\_cursor.execute(sql, (event\_id,))  
 id = self.db\_cursor.fetchone()  
 if not id:  
 id = [**""**]  
 return id[0]  
  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 db = joined\_calendar\_db()  
 print(db.add\_user(**'test1'**, **'1234'**, **"1111111111"**))  
 print(db.add\_calendar(**"test"**, **"test1"**))  
 print(db.add\_calendar(**"test"**, **"test1"**))  
 print(db.add\_user(**'test2'**, **'1234'**, **"2222222222"**))  
 print(db.add\_user(**'test3'**, **'1234'**, **"3333333333"**))  
 print(db.add\_calendar\_participant(**"1"**, **"test2"**))  
 print(db.add\_calendar\_participant(**"1"**, **"test2"**))  
 print(db.add\_calendar\_invitation(**"1"**, **"test1"**, **"test2"**))  
 print(db.add\_calendar\_invitation(**"2"**, **"test1"**, **"test2"**))  
 print(db.add\_calendar\_invitation(**"1"**, **"fake"**, **"test2"**))  
 print(db.add\_event(**"test name"**, [**"test1"**, **"test2"**], **"1"**, **"test1"**, **"14:00"**, **"16:00"**, **"12.02.2024"**))  
 print(db.add\_calendar\_participant(**"2"**, **"test2"**))  
 print(db.add\_event(**"test name"**, [**"test1"**, **"test2"**], **"2"**, **"test1"**, **"11:00"**, **"13:00"**, **"12.02.2024"**))  
 print(db.add\_event(**"test name"**, [**"test1"**], **"2"**, **"test1"**, **"11:00"**, **"13:00"**, **"13.02.2024"**))  
 print(db.add\_event(**"test name"**, [**"test1"**], **"2"**, **"test1"**, **"11:00"**, **"13:00"**, **"13.03.2024"**))  
 print(db.add\_event(**"test name"**, [**"test1"**, **"test2"**], **"1"**, **"test1"**, **"08:00"**, **"09:00"**, **"12.02.2024"**))  
 print(db.add\_event\_participant(**"1"**, **"test1"**))  
 print(db.add\_event\_participant(**"1"**, **"test2"**))  
 print(db.add\_event\_participant(**"700"**, **"test3"**))  
 db.add\_reminder(**"18:00"**, **"15.02.2024"**, **'1'**, **"test1"**)  
 db.add\_reminder(**"17:00"**, **"15.02.2024"**, **'1'**, **"test2"**)  
 db.add\_reminder(**"18:00"**, **"15.03.2024"**, **'2'**, **"test1"**)  
 db.add\_reminder(**"18:00"**, **"15.03.2024"**, **'2'**, **"test2"**)  
 db.delete\_reminder\_for\_user(**"test2"**, **"2"**)  
 db.delete\_reminder\_for\_user(**"test1"**, **"2"**)  
 db.delete\_reminder\_for\_all(**"2"**)  
  
 print(db.get\_today\_reminders())  
 print(db.get\_calendar\_invitations(**"test2"**))  
 print(db.get\_event\_invitations(**"test2"**))  
 db.delete\_calendar\_invitation(**"test2"**, **'1'**)  
 db.delete\_event\_invitation(**"test2"**, **'1'**)  
 print(db.get\_calendar\_invitations(**"test2"**))  
 print(db.get\_event\_invitations(**"test2"**))  
 print(db.get\_user\_calendars(**"test1"**))  
 print(db.delete\_event(**"2"**, **"test1"**))  
 print(db.get\_events\_of\_calendar(**"2"**, **"02"**, **"2024"**))  
 print(db.get\_day\_events(**"test2"**, **"1"**, **"12.02.2024"**))  
 print(db.get\_calendar\_info(**"1"**))  
 print(db.get\_some\_event\_info(**"5"**, **"1"**))  
 print(**f"**{db.get\_some\_event\_info(**'1'**, **'2'**)} **here"**)  
 print(db.get\_day\_color(**"12.02.2024"**, **"1"**))  
 print(db.get\_day\_color(**"12.02.2024"**, **"2"**))  
 print(db.get\_day\_color(**"10.02.2024"**, **"1"**))  
 print(db.get\_day\_color(**"13.02.2024"**, **"2"**))

Client\_logic:

# client\_logic - responsible for the client's logic  
  
from client\_comm import ClientComm  
import queue  
import threading  
import client\_protocol as protocol  
from datetime import datetime  
import time  
from pubsub import pub  
import wx  
from graphics import MyFrame  
  
class Params:  
  
 def \_\_init\_\_(self):  
 self.user\_calendars = []  
 self.current\_day = [] # full date  
 self.day\_events = []  
 self.current\_month = [] # month and year  
 self.invitations = []  
 self.current\_invitation = 0  
 self.current\_calendar = [] # [id, participants, name, manager]  
 self.current\_event = 0  
  
  
def handle\_login(params):  
 """  
 show if and why couldnt log in  
 :param params: status  
 :return:  
 """  
 msg\_to\_send = **"succeed"** if params[0] == **"1"**:  
 msg\_to\_send = **"incorrect password"** call\_error(**"incorrect password"**)  
 elif params[0] == **"2"**:  
 msg\_to\_send = **"incorrect username"** call\_error(**"incorrect username"**)  
 elif params[0] == **"3"**:  
 msg\_to\_send = **"username already open on other computer"** call\_error(**"username already open on other computer"**)  
 elif params[0] == **"0"**:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"login"**)  
  
 return msg\_to\_send  
  
  
  
def login(username, password):  
 """  
 try to log in  
 :param username:  
 :param password:  
 :return:  
 """  
 if **"^"** in username or **"\*"** in username or **","** in username or **"$"** in username or **"^"** in password or **"\*"** in password or **","** in password or **"$"** in password or len(password) < 5:  
 call\_error(**"not valid input"**)  
 else:  
 comm.send(protocol.pack\_login(username, password))  
  
  
def handle\_sign\_up(params):  
 """  
 show if and why couldnt sign up  
 :param params:  
 :return:  
 """  
 msg\_to\_send = **"succeed"** if params[0] == **"1"**:  
 msg\_to\_send = **"username already taken"** call\_error(**"username already taken"**)  
 elif params[0] == **"0"**:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"register"**)  
  
 return msg\_to\_send  
  
  
def sign\_up(username, password, phone\_number):  
 """  
 try to sign up  
 :param username:  
 :param password:  
 :param phone\_number:  
 :return:  
 """  
 if **"^"** in username or **"\*"** in username or **","** in username or **"$"** in username or **"^"** in password or **"\*"** in password or **","** in password or **"$"** in password or len(password) < 5 or not phone\_number.isdigit() or len(phone\_number) != 10:  
 call\_error(**"not valid input"**)  
 else:  
 comm.send(protocol.pack\_signup(username, password, phone\_number))  
  
def call\_error(error):  
 """  
 call error  
 :param error:  
 :return:  
 """  
 wx.CallAfter(pub.sendMessage, **"error"**, error=error)  
  
  
def handle\_new\_calendar(params):  
 """  
 add id to user\_calendars and show the calendar, or show why couldnt create calendar  
 :param params:  
 :return:  
 """  
  
 status, data\_or\_not\_existing\_participants = params  
  
 if status == **"0"**:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"new cal"**)  
 calendar\_id, name, manager, participants = data\_or\_not\_existing\_participants.split(**"^"**)  
 participants = participants.split(**"\*"**)  
 for i in range(len(participants)):  
 participants[i] = participants[i].split(**"$"**)  
 global\_params.current\_calendar = [calendar\_id, participants, name, manager]  
 global\_params.user\_calendars.append(calendar\_id)  
 wx.CallAfter(pub.sendMessage, **"show calendar"**, name=name, manager=manager, participants=participants)  
 elif status == **"9"**:  
 calendar\_id, name, manager, participants = data\_or\_not\_existing\_participants.split(**"^"**)  
 participants = participants.split(**"\*"**)  
 for i in range(len(participants)):  
 participants[i] = participants[i].split(**"$"**)  
 global\_params.current\_calendar = [calendar\_id, participants, name, manager]  
 wx.CallAfter(pub.sendMessage, **"show calendar"**, name=name, manager=manager, participants=participants)  
 else:  
 error = **", "**.join(data\_or\_not\_existing\_participants.split(**"^"**)) + **"do not exist"** call\_error(error)  
  
  
def new\_calendar(name, participants):  
 """  
 send to server to add a new calendar  
 :param name:  
 :param participants:  
 :return:  
 """  
 if name == **"personal"** or **"^"** in name or **"\*"** in name or **","** in name or **"$"** in name:  
 call\_error(**"name can't be 'personal' or contain '^', '\*', '$', ','"**)  
 else:  
 comm.send(protocol.pack\_new\_calendar(name, participants))  
  
  
def handle\_new\_event(params):  
 """  
 if succeed add to screen, and if not show why.  
 :param params: status, id\_or\_not\_existing participants  
 :return:  
 """  
 status, id\_or\_not\_existing\_participants = params  
 if status == **"1"**:  
 if id\_or\_not\_existing\_participants == **"user"**:  
 call\_error(**"can't invite yourself"**)  
 elif id\_or\_not\_existing\_participants != **""** and id\_or\_not\_existing\_participants != **"[]"**:  
 id\_or\_not\_existing\_participants = id\_or\_not\_existing\_participants.split(**"^"**)  
 call\_error(**", "**.join(id\_or\_not\_existing\_participants) + **" do not exist in calendar"**)  
 else:  
 call\_error(**"time isn't available"**)  
 else:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"new event"**)  
  
  
  
def new\_event(calendar\_id, name, participants, start, end, date):  
 """  
 send new event  
 :return:  
 """  
 flag = True  
 if **"^"** in name or **"\*"** in name or **","** in name or **"$"** in name:  
 call\_error(**"name can't contain '^', '\*', '$', ','"**)  
 else:  
  
 time\_format = **"%H:%M"** try:  
 time.strptime(start, time\_format)  
 time.strptime(end, time\_format)  
 except:  
 flag = False  
  
 if flag:  
 flag = len(start) == 5 and len(end) == 5 and (int(start[:2]) < int(end[:2]) or (int(start[:2]) == int(end[:2]) and int(start[3:]) < int(end[3:])))  
  
 if not flag:  
 call\_error(**"time not valid"**)  
 else:  
 comm.send(protocol.pack\_new\_event(calendar\_id, name, start, end, date, participants))  
  
  
def handle\_event\_info(params):  
 """  
 put on screen dot that represent the event. add event to month\_events and if already exists, replace it.  
 :param params: date, color  
 :return:  
 """  
 date, color = params  
 if date in month\_events.keys():  
 del month\_events[date]  
  
 if color:  
 month = date[3: 5]  
 if month == global\_params.current\_month:  
 month\_events[date] = color  
 if color:  
 wx.CallAfter(pub.sendMessage, **"mark"**, dates=[[int(date[:2]), color]])  
 else:  
 wx.CallAfter(pub.sendMessage, **"unmark"**, dates=[int(date[:2])])  
  
  
  
def handle\_is\_calendar\_invitation\_work(params):  
 """  
 show if invitation succeed and if not why  
 :param params: status  
 :return:  
 """  
 error = **""** status = params[0]  
 if status == **"0"**:  
 print(**"invitation succeed"**)  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"new cal parti"**)  
 elif status == **"1"**:  
 error = **"couldnt add invitation because calendar do not exists"** elif status == **"2"**:  
 error = **"couldnt add invitation because you are not the manager"** elif status == **"3"**:  
 error = **"couldnt add invitation because user do not exists"** elif status == **"4"**:  
 error = **"couldnt add invitation because username already exists in calendar"** elif status == **"5"**:  
 error = **"couldnt add invitation because invitation already exists"** if status != **"0"**:  
 call\_error(error)  
  
  
def invite\_to\_calendar(username, calendar\_id):  
 """  
 send invitation to server  
 :param username:  
 :param calendar\_id:  
 :return:  
 """  
 if global\_params.current\_calendar[2] == **"personal"**:  
 call\_error(**"can't invite to personal calendar"**)  
 else:  
 comm.send(protocol.pack\_calendar\_invitation(calendar\_id, username))  
  
  
def handle\_add\_participant\_to\_calendar(params):  
 """  
 if the calendar is the current calendar on screen, add participant to participant list and show on screen  
 :param params:  
 :return:  
 """  
 status, calendar\_id, username = params  
 if global\_params.current\_calendar[0] == calendar\_id:  
 global\_params.current\_calendar[1].append(username)  
  
  
def response\_to\_calendar\_invitation(status, calendar\_id):  
 """  
 send the response  
 :param status:  
 :param calendar\_id:  
 :return:  
 """  
 comm.send((protocol.pack\_calendar\_response(status, calendar\_id)))  
  
  
def handle\_there\_is\_an\_invitation(params):  
 """  
 show that there is invitations  
 :param params:  
 :return:  
 """  
 wx.CallAfter(pub.sendMessage, **"new invitation"**)  
  
  
def invite\_to\_event(username, calendar\_id, event\_id):  
 """  
 send invitation to server  
 :param username:  
 :param calendar\_id:  
 :param event\_id:  
 :return:  
 """  
 comm.send(protocol.pack\_event\_invitation(calendar\_id, event\_id, username))  
  
  
def handle\_is\_event\_invitation\_work(params):  
 """  
 show if invitation succeed and if not why  
 :param params: status  
 :return:  
 """  
 error = **""** status = params[0]  
 if status == **"0"**:  
 print(**"invitation succeed"**)  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"new evt parti"**)  
 elif status == **"1"**:  
 error = **"couldnt add invitation because calendar do not exists"** elif status == **"2"**:  
 error = **"couldnt add invitation because you are not the manager"** elif status == **"3"**:  
 error = **"couldnt add invitation because user do not exists"** elif status == **"4"**:  
 error = **"couldnt add invitation because username do not exists in calendar"** elif status == **"5"**:  
 error = **"couldnt add invitation because invitation already exists"** elif status == **"6"**:  
 error = **"couldnt add invitation because username already exists in event"** if status != **'0'**:  
 call\_error(error)  
  
  
def response\_event\_invitation(status, event\_id):  
 """  
 send the response  
 :param status:  
 :param event\_id:  
 :return:  
 """  
 comm.send((protocol.pack\_event\_response(status, event\_id)))  
  
  
def change\_name\_of\_calendar(name, calendar\_id):  
 """  
 change name of calendar  
 :param name:  
 :param calendar\_id:  
 :return:  
 """  
 if name == **"personal"** or **"^"** in name or **"\*"** in name or **","** in name or **"$"** in name:  
 call\_error(**"name can't be 'personal' or contain '^', '\*', '$', ','"**)  
 elif global\_params.current\_calendar[2] == **"personal"**:  
 call\_error(**"can't change name of personal calendar"**)  
 else:  
 comm.send(protocol.pack\_calendar\_name\_edit(calendar\_id, name))  
  
  
def handle\_calendar\_name\_change(params):  
 """  
 change name of calendar on screen  
 :param params: status, calendar\_id, name  
 :return:  
 """  
 status, calendar\_id, name = params  
 if status == **"0"**:  
 if global\_params.current\_calendar[0] == calendar\_id:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"cal name"**)  
 global\_params.current\_calendar[2] = name  
 wx.CallAfter(pub.sendMessage, **"show calendar"**, name=name, manager=global\_params.current\_calendar[3], participants=global\_params.current\_calendar[1])  
 dates = []  
 for i in month\_events.keys():  
 dates += [[int(i[:2]), month\_events[i]]]  
 wx.CallAfter(pub.sendMessage, **"mark"**, dates=dates)  
 else:  
 call\_error(**"couldnt change name of calendar because you are not the manager"**)  
  
  
def change\_name\_of\_event(name, event\_id):  
 """  
 change name of event  
 :param name:  
 :param event\_id:  
 :return:  
 """  
 if **"^"** in name or **"\*"** in name or **","** in name or **"$"** in name:  
 call\_error(**"name can't contain '^', '\*', '$', ','"**)  
 else:  
 comm.send(protocol.pack\_event\_name\_edit(event\_id, name))  
  
  
def handle\_event\_name\_change(params):  
 """  
 tell the user if succeed or not  
 :param params: status, calendar\_id, event\_id, name  
 :return:  
 """  
 status, event\_id, name = params  
 if status == **'0'**:  
 current\_event = [i for i in range(len(global\_params.day\_events)) if global\_params.day\_events[i][-1] == str(event\_id)][-1]  
 global\_params.day\_events[current\_event][1] = name  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"evt name"**)  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.day\_events[global\_params.current\_event])  
 else:  
 call\_error(**"you are not the manager so couldnt change name of event"**)  
  
  
def change\_time(event\_id, start, end, date):  
 """  
 change time of event  
 :param event\_id:  
 :param start:  
 :param end:  
 :param date:  
 :return:  
 """  
 date\_format = **"%d.%m.%Y"** time\_format = **"%H:%M"** try:  
 time.strptime(start, time\_format)  
 time.strptime(end, time\_format)  
 flag = bool(datetime.strptime(date, date\_format))  
 except:  
 flag = False  
  
 if flag:  
 flag = len(start) == 5 and len(end) == 5 and len(date) == 10 and (int(start[:2]) < int(end[:2]) or (int(start[:2]) == int(end[:2])and int(start[3:]) < int(end[3:])))  
  
 if flag:  
 comm.send(protocol.pack\_event\_time\_edit(event\_id, start, end, date))  
 else:  
 call\_error(**"not valid input"**)  
  
  
def handle\_time\_change(params):  
 """  
 show change of time or show why couldnt change  
 :param params: status, calendar\_id, event\_id, time\_or\_participants  
 :return:  
 """  
 status, calendar\_id, event\_id, time\_or\_participants = params  
 if status == **"1"**:  
 participants = time\_or\_participants.split(**"^"**)  
 call\_error(**f"couldnt change time because** {**', '**.join(participants)} **are not free at this time"**)  
 elif status == **"0"**:  
 start, end, date = time\_or\_participants.split(**"^"**)  
 for i in range(len(global\_params.day\_events)):  
 if global\_params.day\_events[i][-1] == event\_id:  
 global\_params.day\_events[i][4] = date  
 global\_params.day\_events[i][2] = start  
 global\_params.day\_events[i][3] = end  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"evt time"**)  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.day\_events[global\_params.current\_event])  
 elif status == **"2"**:  
 call\_error(**"couldnt change time because you are not the manager"**)  
 else:  
 call\_error(**"event do not exist so cant change its time"**)  
  
  
def delete\_event(event\_id):  
 """  
 try to delete the event  
 :param event\_id:  
 :return:  
 """  
 comm.send(protocol.pack\_event\_delete(event\_id))  
  
  
def handle\_delete\_event(params):  
 """  
 show if succeed and if not why  
 :param params: status, event\_id  
 :return:  
 """  
 error = **""** status, event\_id = params  
 if status == **"0"**:  
 global\_params.day\_events.remove(global\_params.day\_events[global\_params.current\_event])  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"event"**)  
 if len(global\_params.day\_events) == 0:  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.current\_day)  
 elif global\_params.current\_event == len(global\_params.day\_events):  
 global\_params.current\_event = global\_params.current\_event - 1  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.day\_events[global\_params.current\_event])  
 else:  
 global\_params.current\_event = global\_params.current\_event  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.day\_events[global\_params.current\_event])  
 elif status == **"1"**:  
 error = **"cant delete event because you are not the manager"** else:  
 error = **"cant delete event because event do not exist"** if status != **"0"**:  
 call\_error(error)  
  
  
def exit\_calendar(calendar\_id):  
 """  
 try to exit a calendar  
 :param calendar\_id:  
 :return:  
 """  
 if global\_params.current\_calendar[2] == **"personal"**:  
 call\_error(**"cant delete calendar because its a personal one"**)  
 else:  
 comm.send(protocol.pack\_exit\_calendar(calendar\_id))  
  
  
def handle\_exit\_calendar(params):  
 """  
 delete calendar participant  
 :param params: username, calendar\_id  
 :return:  
 """  
 username, calendar\_id = params  
 if calendar\_id == global\_params.current\_calendar[0]:  
 global\_params.current\_calendar[1].remove(username)  
  
  
def handle\_delete\_calendar(params):  
 """  
 delete caledar from screen and from list of ids  
 :param params: status, calendar\_id  
 :return:  
 """  
 status, calendar\_id = params  
 if status == **"0"**:  
 if global\_params.current\_calendar[0] == calendar\_id:  
  
 if global\_params.user\_calendars.index(calendar\_id) == (len(global\_params.user\_calendars)-1):  
 get\_calendar\_info(global\_params.user\_calendars[-2])  
  
  
 else:  
 get\_calendar\_info(global\_params.user\_calendars[global\_params.user\_calendars.index(calendar\_id)+1])  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"calendar"**)  
  
 global\_params.user\_calendars.remove(calendar\_id)  
 else:  
 call\_error(**"couldnt delete calendar, please try again"**)  
  
  
def get\_calendar\_ids():  
 """  
 try to get calendar ids  
 :return:  
 """  
 comm.send(protocol.pack\_calendar\_ids())  
  
  
def handle\_calendar\_ids(params):  
 """  
 save calendar ids in user\_calendars  
 :param params:  
 :return:  
 """  
 calendar\_ids = params[0]  
 calendar\_ids = calendar\_ids.split(**"^"**)  
 global\_params.user\_calendars = calendar\_ids  
  
  
  
def get\_day\_events(calendar\_id, date):  
 """  
 try to get day events of calendar  
 :param calendar\_id:  
 :param date:  
 :return:  
 """  
 date\_format = **"%d.%m.%Y"** try:  
 flag = bool(datetime.strptime(date, date\_format))  
 except:  
 flag = False  
  
 if flag:  
 comm.send(protocol.pack\_day\_event(calendar\_id, date))  
 global\_params.current\_day = date  
 else:  
 print(**"something went wrong - not valid date"**)  
  
  
def handle\_day\_events(params):  
 """  
 show on screen the events, save day events in day\_events  
 :param params: day\_events  
 :return:  
 """  
 if params[0] != **""**:  
 events = params[0].split(**"\*"**)  
 for i in range(len(events)):  
 events[i] = events[i].split(**"^"**)  
 events[i][0] = events[i][0].split(**"$"**)  
 global\_params.day\_events = events  
 global\_params.current\_event = 0  
 event = events[0]  
 else:  
 global\_params.day\_events = []  
 event = [global\_params.current\_day][0]  
  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=event)  
  
  
def get\_month\_events(calendar\_id, month, year):  
 """  
 try to get month events of calendar  
 :param calendar\_id:  
 :param month:  
 :param year:  
 :return:  
 """  
 if month.isnumeric() and year.isnumeric() and len(month) == 2 and len(year) == 4:  
 comm.send(protocol.pack\_month\_events(calendar\_id, month, year))  
 global\_params.current\_month = month + **"."** + year  
 else:  
 print(**"something went wrong - not valid month and year"**)  
  
  
def handle\_month\_events(params):  
 """  
 show month events on screen and save it on the dictionary  
 :param params:  
 :return:  
 """  
 events = params[0].split(**"\*"**)  
 month\_events.clear()  
 if len(events) > 0 and events[0] != **""**:  
 for i in events:  
 date, color = i.split(**"^"**)  
 month\_events[date] = color  
 dates = []  
 for i in month\_events.keys():  
 dates += [[int(i[:2]), month\_events[i]]]  
 wx.CallAfter(pub.sendMessage, **"mark"**, dates=dates)  
  
  
def get\_calendar\_info(calendar\_id):  
 """  
 try to get calendar info  
 :param calendar\_id:  
 :return:  
 """  
 comm.send(protocol.pack\_get\_calendar\_info(calendar\_id))  
  
  
def get\_invitations():  
 """  
 try to get invitations  
 :return:  
 """  
 comm.send(protocol.pack\_invitations\_request())  
  
  
def handle\_invitations(params):  
 """  
 get invitations and shoe them on screen  
 :param params: invitations  
 :return:  
 """  
 if params == [**""**]:  
 wx.CallAfter(pub.sendMessage, **"show invitation"**, params=[])  
 else:  
 invitations = params[0].split(**"\*"**)  
 for i in range(len(invitations)):  
 invitations[i] = invitations[i].split(**"^"**)  
 invitations[i][2] = invitations[i][2].split(**"$"**)  
 global\_params.invitations = invitations  
 wx.CallAfter(pub.sendMessage, **"show invitation"**, params=invitations[0])  
 global\_params.current\_invitation = 0  
  
  
def handle\_recv(msg\_q):  
 """  
 handle msgs from server  
 :param msg\_q:  
 :return:  
 """  
 while True:  
 msg = msg\_q.get()  
 opcode, params = protocol.unpack(msg)  
 print(**f'got from server:** {msg}**'**)  
 if opcode in opcodes:  
 print(opcodes[opcode](params))  
 else:  
 print(**f'command number** {opcode} **not in dictionary'**)  
  
  
def handle\_graphics(graphics\_q):  
 """  
 handle msgs from graphics  
 :param graphics\_q:  
 :return:  
 """  
 while True:  
 msg = graphics\_q.get()  
 opcode, params = msg  
 print(**f'got from graphics:** {msg}**'**)  
 if opcode == **"register"**:  
 username, password, phone = params  
 sign\_up(username, password, phone)  
 elif opcode == **"login"**:  
 username, password = params  
 login(username, password)  
 elif opcode == **"right"**:  
 current = global\_params.user\_calendars.index(global\_params.current\_calendar[0])  
 if len(global\_params.user\_calendars) == (current+1):  
 call\_error(**"can't go more right"**)  
 else:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"calendar"**)  
 get\_calendar\_info(global\_params.user\_calendars[current+1])  
  
 elif opcode == **"left"**:  
 current = global\_params.user\_calendars.index(global\_params.current\_calendar[0])  
 if current == 0:  
 call\_error(**"can't go more left"**)  
 else:  
 get\_calendar\_info(global\_params.user\_calendars[current-1])  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"calendar"**)  
 elif opcode == **"month"**:  
 month, year = params  
 global\_params.current\_month = [month, year]  
 dates = []  
 for i in month\_events.keys():  
 dates += [int(i[:2])]  
 wx.CallAfter(pub.sendMessage, **"unmark"**, dates=dates)  
 month\_events.clear()  
 get\_month\_events(global\_params.current\_calendar[0], month, year)  
 elif opcode == **"day"**:  
 date = params  
 get\_day\_events(global\_params.current\_calendar[0], date)  
 elif opcode == **"left event"**:  
 if global\_params.current\_event == 0:  
 call\_error(**"can't go more left"**)  
 else:  
 global\_params.current\_event = global\_params.current\_event - 1  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"event"**)  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.day\_events[global\_params.current\_event])  
 elif opcode == **"right event"**:  
 if (global\_params.current\_event + 1) == len(global\_params.day\_events):  
 call\_error(**"can't go more right"**)  
 else:  
 global\_params.current\_event = global\_params.current\_event + 1  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"event"**)  
 wx.CallAfter(pub.sendMessage, **"show day"**, event=global\_params.day\_events[global\_params.current\_event])  
 elif opcode == **"invitations"**:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"calendar"**)  
 get\_invitations()  
 elif opcode == **"left invitation"**:  
 if global\_params.current\_invitation == 0:  
 call\_error(**"can't go more left"**)  
 else:  
 global\_params.current\_invitation = global\_params.current\_invitation - 1  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"invitation"**)  
 wx.CallAfter(pub.sendMessage, **"show invitation"**, params=global\_params.invitations[global\_params.current\_invitation])  
 elif opcode == **"right invitation"**:  
 if (global\_params.current\_invitation + 1) == len(global\_params.invitations):  
 call\_error(**"can't go more right"**)  
 else:  
 global\_params.current\_invitation = global\_params.current\_invitation + 1  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"invitation"**)  
 wx.CallAfter(pub.sendMessage, **"show invitation"**, params=global\_params.invitations[global\_params.current\_invitation])  
 elif opcode == **"response"**:  
 wx.CallAfter(pub.sendMessage, **"hide"**, panel=**"invitation"**)  
 if len(global\_params.invitations[global\_params.current\_invitation]) > 5:  
 response\_event\_invitation(params[0], global\_params.invitations[global\_params.current\_invitation][-1])  
 else:  
 response\_to\_calendar\_invitation(params[0], global\_params.invitations[global\_params.current\_invitation][-1])  
 global\_params.invitations.remove(global\_params.invitations[global\_params.current\_invitation])  
  
 if len(global\_params.invitations) != 0:  
 if not (params[0] == **"0"** and len(global\_params.invitations[global\_params.current\_invitation]) < 5):  
 if len(global\_params.invitations) == global\_params.current\_invitation - 1:  
 global\_params.current\_invitation = 0  
 wx.CallAfter(pub.sendMessage, **"show invitation"**, params=global\_params.invitations[global\_params.current\_invitation])  
 else:  
 wx.CallAfter(pub.sendMessage, **"show invitation"**, params=[])  
 elif opcode == **"new cal parti"**:  
 for i in params:  
 invite\_to\_calendar(i, global\_params.current\_calendar[0])  
 elif opcode == **"new cal"**:  
 name, users = params  
 new\_calendar(name, users)  
 elif opcode == **"new evt parti"**:  
 for i in params:  
 invite\_to\_event(i, global\_params.current\_calendar[0], global\_params.day\_events[global\_params.current\_event][-1])  
 elif opcode == **"new event"**:  
 new\_event(global\_params.current\_calendar[0], params[0], params[1], params[2], params[3], params[4])  
 elif opcode == **"del evt"**:  
 delete\_event(global\_params.day\_events[global\_params.current\_event][-1])  
 elif opcode == **"exit cal"**:  
 exit\_calendar(global\_params.current\_calendar[0])  
 elif opcode == **"cal name"**:  
 change\_name\_of\_calendar(params, global\_params.current\_calendar[0])  
 elif opcode == **"evt name"**:  
 change\_name\_of\_event(params, global\_params.day\_events[global\_params.current\_event][-1])  
 elif opcode == **"evt time"**:  
 change\_time(global\_params.day\_events[global\_params.current\_event][-1], params[1], params[2], params[0])  
 else:  
 print(**f'command** {opcode} **not valid'**)  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 msg\_q = queue.Queue()  
 graphics\_q = queue.Queue()  
 comm = ClientComm(**'192.168.4.82'**, 4500, msg\_q)  
 opcodes = {**"00"**: handle\_login, **"01"**: handle\_sign\_up, **"02"**: handle\_new\_calendar, **"04"**: handle\_new\_event,  
 **"05"**: handle\_event\_info, **"10"**: handle\_there\_is\_an\_invitation, **"11"**: handle\_add\_participant\_to\_calendar,  
 **"13"**: handle\_is\_event\_invitation\_work, **"14"**: handle\_invitations, **"15"**: handle\_is\_calendar\_invitation\_work,  
 **"20"**: handle\_calendar\_name\_change, **"21"**: handle\_event\_name\_change, **"22"**: handle\_time\_change,  
 **"30"**: handle\_delete\_event, **"31"**: handle\_delete\_calendar, **"32"**: handle\_exit\_calendar, **"40"**: handle\_calendar\_ids,  
 **"41"**: handle\_day\_events, **"42"**: handle\_month\_events}  
 month\_events = {} # date: color  
 global\_params = Params()  
 global\_params.day\_events = [[**"1"**, **"01.01.2024"**, **"hello"**, **"18:00"**, **"20:00"**, **"amit"**, [**"amit"**, **"alon"**, **"yuval"**]]] # [id, date, name, start, end, manager, [participants]]  
  
  
 # tests i did  
 # login("test2", "1234")  
 # sign\_up("test4", "1234", "4444444444")  
 # new\_calendar("hello", ["test3"])  
 # new\_event("1", "hello", ["test2"], "20:00", "21:00", "18.03.2024")  
 # invite\_to\_calendar("test1", '21')  
 # invite\_to\_calendar("test1", '1')  
 # response\_to\_calendar\_invitation("0", '21')  
 # response\_to\_calendar\_invitation("1", '21')  
 # invite\_to\_event("test3", "1", "1")  
 # response\_event\_invitation("0", "9")  
 # change\_name\_of\_calendar("new name", "1")  
 # change\_name\_of\_event("new name", "1")  
 # change\_time("10", "18:00", "20:00", "21.03.2024")  
 # change\_time("9", "18:00", "21:00", "17.03.2024")  
 # change\_time("9", "18:00", "21:00", "20.03.2024")  
 # delete\_event("10")  
 # delete\_event("9")  
 # exit\_calendar("58")  
 # get\_calendar\_ids()  
 # get\_day\_events("1", "12.02.2024")  
 # get\_month\_events("1", "03", "2024")  
 # get\_invitations()  
  
  
 threading.Thread(target=handle\_recv, args=(msg\_q,)).start()  
 threading.Thread(target=handle\_graphics, args=(graphics\_q,)).start()  
  
  
 app = wx.App(False)  
 frame = MyFrame(graphics\_q)  
 frame.Show()  
 app.MainLoop()

Server logic:

# server\_logic - responsible for the server's logic  
  
  
from server\_comm import ServerComm  
import queue  
import server\_protocol as protocol  
from joined\_calendar\_db import joined\_calendar\_db as data\_base  
import hashlib  
from datetime import datetime  
  
def handle\_login(ip, params):  
 """  
 check if password and username correct and return msg accordingly. if succeed - send to the client his calendar ids and his invitations.  
 :param ip:  
 :param params: username, password  
 :return:  
 """  
 username, password = params  
 status = 0  
 msgs\_to\_send = []  
 if not db.is\_user\_exists(username):  
 status = 2  
 elif db.get\_password(username) != hashlib.sha256(password.encode()).digest():  
 status = 1  
 elif username in current\_users.keys():  
 status = 3  
 else:  
 ids = db.get\_user\_calendars(username)  
 msgs\_to\_send += [protocol.pack\_calendar\_ids(ids)]  
 data = [ids[0]] + db.get\_calendar\_info\_with\_colors(ids[0])  
 msgs\_to\_send += [protocol.pack\_new\_calendar(**"9"**, data)]  
 today = datetime.now()  
 current\_open\_calendars[username] = [ids[0], str(today.month).zfill(2)+**"."**+str(today.year)] # user: [calendar\_id, month and year]  
 msgs\_to\_send += [protocol.pack\_month\_events(db.get\_events\_of\_calendar(ids[0], today.month, today.year))]  
  
 if db.get\_calendar\_invitations(username) or db.get\_event\_invitations(username):  
 msgs\_to\_send += [protocol.pack\_there\_is\_an\_invitation()]  
 current\_users[username] = ip  
 comm.send(ip, protocol.pack\_login(status))  
 for msg in msgs\_to\_send:  
 comm.send(ip, msg)  
  
  
def handle\_sign\_up(ip, params):  
 """  
 try to open a username and return msg if available or if taken. if available, add user and add him a private calendar and send it to him  
 :param ip:  
 :param params: username, password, phone  
 :return:  
 """  
 username, password, phone\_number = params  
 status = 1  
 msg = **""** msgs\_to\_send = []  
 if not db.is\_user\_exists(username):  
 status = 0  
  
 db.add\_user(username, password, phone\_number)  
 db.add\_calendar(**"personal"**, username)  
 ids = db.get\_user\_calendars(username)  
 msgs\_to\_send += [protocol.pack\_calendar\_ids(ids)]  
 data = [ids[0]] + db.get\_calendar\_info\_with\_colors(ids[0])  
 msgs\_to\_send += [protocol.pack\_new\_calendar(**"9"**, data)]  
 today = datetime.now()  
 current\_open\_calendars[username] = [ids[0], str(today.month).zfill(2)+**"."**+str(today.year)] # user: [calendar\_id, month and year]  
 msgs\_to\_send += [protocol.pack\_month\_events(db.get\_events\_of\_calendar(ids[0], today.month, today.year))]  
 current\_users[username] = ip  
 comm.send(ip, protocol.pack\_sign\_up(status))  
 for msg in msgs\_to\_send:  
 comm.send(ip, msg)  
  
def handle\_new\_calendar(ip, params):  
 """  
 check if every participant is an existing user and if so, create the calendar and add invitations. send status if succeed. if succeed send id and if not send why  
 :param ip:  
 :param params: name, participants  
 :return:  
 """  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 name, participants = params  
 participants = participants.split(**"^"**)  
 not\_existing\_participants = [i for i in participants if not db.is\_user\_exists(i)]  
 if not not\_existing\_participants:  
 id = db.add\_calendar(name, username)  
 comm.send(ip, protocol.pack\_new\_calendar(**"0"**, [id, name, username, [username, **"BLUE"**]]))  
 today = datetime.now()  
 current\_open\_calendars[username] = [id, str(today.month).zfill(2)+**"."**+str(today.year)] # user: [calendar\_id, month and year]  
 comm.send(ip, protocol.pack\_month\_events(db.get\_events\_of\_calendar(id, today.month, today.year)))  
 for i in participants:  
 db.add\_calendar\_invitation(id, username, i)  
 if i in current\_users:  
 comm.send(current\_users[i], protocol.pack\_there\_is\_an\_invitation())  
 else:  
 comm.send(ip, protocol.pack\_new\_calendar(**"1"**, not\_existing\_participants))  
  
  
def handle\_new\_event(ip, params):  
 """  
 check if time is available and if so add event. add invitations if participants in the calendar. send to participants that the calendar is open on their screen the info(color, date)  
 :param ip:  
 :param params: calendar\_id, name, start, end, date, participants  
 :return:  
 """  
 calendar\_id, name, start, end, date, participants = params  
 print(**"params"**, calendar\_id, name, start, end, date, participants)  
 participants = participants.split(**"^"**)  
 print(participants)  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 not\_existing\_participants = [i for i in participants if not db.is\_user\_exists(i)]  
 not\_in\_calendar = []  
 not\_in\_calendar = [i for i in participants if not db.is\_participant\_exists\_in\_calendar(calendar\_id, i)]  
 not\_existing\_participants = list(set(not\_existing\_participants).union(set(not\_in\_calendar)))  
 if username in participants:  
 comm.send(ip, protocol.pack\_new\_event(**"1"**, **"user"**))  
 elif not\_existing\_participants and not\_existing\_participants != [**""**]:  
 comm.send(ip, protocol.pack\_new\_event(**"1"**, not\_existing\_participants))  
 else:  
 if db.check\_is\_time\_available(username, start, end, date):  
 if participants == [**""**]:  
 participants = []  
 event\_id = db.add\_event(name, participants, calendar\_id, username, start, end, date)  
 comm.send(ip, protocol.pack\_new\_event(**"0"**, event\_id))  
 for p in participants:  
 db.add\_event\_invitation(event\_id, calendar\_id, p, username)  
 if p in current\_users:  
 comm.send(current\_users[p], protocol.pack\_there\_is\_an\_invitation())  
 date, color = get\_event\_info(date, calendar\_id)  
 comm.send(ip, protocol.pack\_event\_info(date, color))  
 for user in current\_open\_calendars:  
 month = date[3:]  
 if month == current\_open\_calendars[user][1] and db.is\_participant\_exists\_in\_calendar(current\_open\_calendars[user][0], username) and user != username:  
 date, color = get\_event\_info(event\_id, current\_open\_calendars[user][0])  
 comm.send(current\_users[user], protocol.pack\_event\_info(date, color))  
  
 else:  
 comm.send(ip, protocol.pack\_new\_event(**"1"**, **""**))  
  
  
def get\_event\_info(date, calendar\_id):  
 """  
 return the color of the event, the id and the date.  
 :param date:  
 :param calendar\_id:  
 :return:  
 """  
 return db.get\_day\_color(date, calendar\_id)  
  
  
def handle\_calendar\_invitation(ip, params):  
 """  
 if calendar exists, invited\_by is manager, username exists and not already in calendar or invitation already exists, add invitation to table and if username is in current\_users send him the invitation  
 :param params: calendar\_id, username, invited\_by  
 :return:  
 """  
 calendar\_id, username = params  
 invited\_by = [i for i in current\_users if current\_users[i] == ip][0]  
 if not db.\_is\_calendar\_exists(calendar\_id):  
 comm.send(ip, protocol.pack\_is\_calendar\_invitation\_work(**"1"**))  
 elif not db.is\_manager\_calander(calendar\_id, invited\_by):  
 comm.send(ip, protocol.pack\_is\_calendar\_invitation\_work(**"2"**))  
 elif not db.is\_user\_exists(username):  
 comm.send(ip, protocol.pack\_is\_calendar\_invitation\_work(**"3"**))  
 elif db.is\_participant\_exists\_in\_calendar(calendar\_id, username):  
 comm.send(ip, protocol.pack\_is\_calendar\_invitation\_work(**"4"**))  
 elif db.is\_calendar\_invitation\_exists(calendar\_id, username):  
 comm.send(ip, protocol.pack\_is\_calendar\_invitation\_work(**"5"**))  
 else:  
 comm.send(ip, protocol.pack\_is\_calendar\_invitation\_work(**"0"**))  
 db.add\_calendar\_invitation(calendar\_id, invited\_by, username)  
 if username in current\_users.keys():  
 comm.send(current\_users[username], protocol.pack\_there\_is\_an\_invitation())  
  
  
def handle\_is\_calendar\_accepted(ip, params):  
 """  
 if accepted - add participant to table, send calendar in current month to user, send users that are currently looking on the calendar the new participant, remove invitation  
 if declined - remove invitation from table  
 :param ip:  
 :param params: status, calendar\_id  
 :return:  
 """  
 status, calendar\_id = params  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 db.delete\_calendar\_invitation(username, calendar\_id)  
 if status == **"0"**:  
 db.add\_calendar\_participant(calendar\_id, username)  
 info = db.get\_calendar\_info\_with\_colors(calendar\_id)  
 comm.send(ip, protocol.pack\_new\_calendar(status, [calendar\_id, info[0], info[1], info[2]]))  
 today = datetime.now()  
 current\_open\_calendars[username] = [calendar\_id, str(today.month).zfill(2)+**"."**+str(today.year)] # user: [calendar\_id, month and year]  
 for user in current\_open\_calendars:  
 if current\_open\_calendars[user][0] == calendar\_id:  
 if user != username:  
 comm.send(current\_users[user], protocol.pack\_new\_calendar\_participant(status, calendar\_id, username))  
 comm.send(current\_users[user], protocol.pack\_month\_events(db.get\_events\_of\_calendar(calendar\_id, current\_open\_calendars[user][1][:2], current\_open\_calendars[user][1][3:])))  
  
def handle\_event\_invitation(ip, params):  
 """  
 check if username is manager and if username, calendar and event exist and if username is participant in the calendar and if invitation not already exists and username not already participant.  
 if so, add invitation to table and if username in current usernames, send the invitation.  
 :param ip:  
 :param params: calendar\_id, event\_id, username  
 :return:  
 """  
  
 calendar\_id, event\_id, username = params  
 invited\_by = [i for i in current\_users if current\_users[i] == ip][0]  
 if not db.\_is\_calendar\_exists(calendar\_id):  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"1"**))  
 elif not db.is\_manager\_event(event\_id, invited\_by):  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"2"**))  
 elif not db.is\_user\_exists(username):  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"3"**))  
 elif not db.is\_participant\_exists\_in\_calendar(calendar\_id, username):  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"4"**))  
 elif db.is\_event\_invitation\_exists(event\_id, username):  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"5"**))  
 elif db.is\_participant\_exists\_in\_event(event\_id, username):  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"6"**))  
 else:  
 comm.send(ip, protocol.pack\_event\_invitation\_succeed(**"0"**))  
 db.add\_event\_invitation(event\_id, calendar\_id, username, invited\_by)  
 if username in current\_users.keys():  
 comm.send(current\_users[username],  
 protocol.pack\_there\_is\_an\_invitation())  
  
  
def handle\_is\_event\_accepted(ip, params):  
 """  
 if accepted - add participant to table, send event to user, send users that are currently looking on the calendar the new color, remove invitation  
 if declined - remove invitation from table  
 :param ip:  
 :param params: status, event\_id  
 :return:  
 """  
 status, event\_id = params  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 db.delete\_event\_invitation(username, event\_id)  
 if status == **"0"**:  
 db.add\_event\_participant(event\_id, username)  
 calendar\_id = db.get\_calendar\_from\_event(event\_id)  
 date = db.get\_event\_date(event\_id)  
 info = db.get\_day\_color(date, calendar\_id)  
 if len(info) > 0 and info != [**""**]:  
 for user in current\_open\_calendars:  
 if current\_open\_calendars[user][1] == info[0][3:] and db.is\_participant\_exists\_in\_calendar(current\_open\_calendars[user][0], username):  
 date, color = db.get\_day\_color(date, current\_open\_calendars[user][0])  
 comm.send(current\_users[user], protocol.pack\_event\_info(date, color))  
  
  
def handle\_change\_name\_of\_calendar(ip, params):  
 """  
 if manager - change the name of the calendar on the table.  
 send new name for users who currently watch the calendar  
 :param ip:  
 :param params: calendar\_id, name  
 :return:  
 """  
 calendar\_id, name = params  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 if db.is\_manager\_calander(calendar\_id, username):  
 db.change\_calendar\_name(calendar\_id, name)  
 comm.send(ip, protocol.pack\_calendar\_name\_edit(**"0"**, calendar\_id, name))  
 for user in current\_open\_calendars:  
 if current\_open\_calendars[user][0] == calendar\_id:  
 comm.send(current\_users[user], protocol.pack\_calendar\_name\_edit(**"0"**, calendar\_id, name))  
 else:  
 comm.send(ip, protocol.pack\_calendar\_name\_edit(**"1"**, calendar\_id, **""**))  
  
  
def handle\_change\_name\_of\_event(ip, params):  
 """  
 if manager - change the name of the calendar on the table.  
 send new name for users who currently watch the calendar  
 :param ip:  
 :param params: calendar\_id, name  
 :return:  
 """  
 event\_id, name = params  
 calendar\_id = db.get\_calendar\_from\_event(event\_id)  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 if db.is\_manager\_event(event\_id, username):  
 db.change\_event\_name(event\_id, name)  
 comm.send(ip, protocol.pack\_event\_name\_edit(**"0"**, event\_id, name))  
 else:  
 comm.send(ip, protocol.pack\_event\_name\_edit(**"1"**, event\_id, **""**))  
  
  
def handle\_time\_change(ip, params):  
 """  
 if all participants are free and user is manager - change time in table  
 if not, send why couldnt change  
 :param ip:  
 :param params: event\_id, start, end, date  
 :return:  
 """  
 event\_id, start, end, date = params  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 if db.\_is\_event\_exists(event\_id):  
 calendar\_id = db.get\_calendar\_from\_event(event\_id)  
 participants = db.get\_event\_participants(event\_id)  
 users\_do\_not\_free = []  
 for p in participants:  
 if not db.check\_is\_time\_available(p, start, end, date) and db.events\_in\_time(p, start, end, date) != [event\_id]:  
 users\_do\_not\_free.append(p)  
 if len(users\_do\_not\_free) > 0:  
 comm.send(ip, protocol.pack\_time\_edit(**"1"**, calendar\_id, event\_id, users\_do\_not\_free))  
 elif not db.is\_manager\_event(event\_id, username):  
 comm.send(ip, protocol.pack\_time\_edit(**"2"**, calendar\_id, event\_id, []))  
 else:  
 old\_date = db.get\_event\_date(event\_id)  
 db.change\_event\_time(event\_id, start, end, date)  
 if date != old\_date:  
 participants = set(participants)  
 for user in current\_users:  
 if current\_open\_calendars[user][1] == date[3:]:  
 p = set(db.get\_calendar\_participants(current\_open\_calendars[user][0]))  
 both = list(p & participants)  
 if len(both) > 0:  
  
 date, color = db.get\_day\_color(date, current\_open\_calendars[user][0])  
 comm.send(current\_users[user], protocol.pack\_event\_info(date, color))  
 if current\_open\_calendars[user][1] == old\_date[3:]:  
 p = set(db.get\_calendar\_participants(current\_open\_calendars[user][0]))  
 both = list(p & participants)  
 if len(both) > 0:  
 old\_date\_and\_color = db.get\_day\_color(old\_date, current\_open\_calendars[user][0])  
 if len(old\_date\_and\_color) > 0:  
 old\_date, color = old\_date\_and\_color  
 comm.send(current\_users[user], protocol.pack\_event\_info(old\_date, color))  
 else:  
 comm.send(current\_users[user], protocol.pack\_event\_info(old\_date, **""**))  
  
 comm.send(ip, protocol.pack\_time\_edit(**"0"**, current\_open\_calendars[username][0], event\_id, [start, end, date]))  
 else:  
 comm.send(ip, protocol.pack\_time\_edit(**"3"**, **""**, event\_id, []))  
  
  
def handle\_delete\_event(ip, params):  
 """  
 delete event from table and send it to the users who are currently looking on the month of the event and the event is on their calendar  
 :param params: event\_id  
 :return:  
 """  
 event\_id = params[0]  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 if db.\_is\_event\_exists(event\_id) and db.is\_manager\_event(event\_id, username):  
 date\_of\_event = db.get\_event\_date(event\_id)  
 month\_year = date\_of\_event[3:]  
 participants = set(db.get\_event\_participants(event\_id))  
 db.delete\_event(event\_id, username)  
 comm.send(ip, protocol.pack\_event\_dalete(**"0"**, event\_id))  
 for user in current\_users:  
 if current\_users[user][1] == month\_year:  
 calendar\_id = current\_users[user][0]  
 p = set(db.get\_calendar\_participants(calendar\_id))  
 both = list(p & participants)  
 if len(both) > 0:  
 info = db.get\_some\_event\_info(event\_id, calendar\_id)  
 if len(info) > 0:  
 date, color = info  
 else:  
 date = date\_of\_event  
 color = **""** comm.send(current\_users[user], protocol.pack\_event\_info(date, color))  
  
 elif db.\_is\_event\_exists(event\_id) and not db.is\_manager\_event(event\_id, username):  
 comm.send(ip, protocol.pack\_event\_dalete(**"1"**, event\_id))  
 else:  
 comm.send(ip, protocol.pack\_event\_dalete(**"2"**, event\_id))  
  
  
def handle\_get\_invitations(ip, params):  
 """  
 send calendar's and event's invitations to user  
 :param ip:  
 :param params:  
 :return:  
 """  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 comm.send(ip, protocol.pack\_invitations(db.get\_calendar\_invitations(username) + db.get\_event\_invitations(username)))  
  
  
def handle\_exit\_calendar(ip, params):  
 """  
 if calendar exist:  
 if username is manager - delete calendar and all its participants, delete events that were created in the calendar and tell the clients who are on the calendar that it was deleted  
 otherwise - delete participant from table, delete participant from events that were created in the calendar and if he is their manager - deletes them  
 send the users that are currently on the calendar that the participant has exit  
 in both cases - if the user that exits is manager of some events so they get deleted - send users who are currently on a calendar that some of its participants are in the calendar that someone exits  
 otherwise and if user is participant in some events on the calendar - send users who are currently on a calendar that the user that exits the calendar was in is a participant there.  
 otherwise - tell the user why couldnt exit  
 :param params: calendar\_id  
 :return:  
 """  
 calendar\_id = params[0]  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 info = db.get\_calendar\_info(calendar\_id)  
 status = db.exit\_calendar(calendar\_id, username)  
 if status == 4:  
 comm.send(ip, protocol.pack\_delete\_calendar(**"1"**, calendar\_id))  
 else:  
 name, manager, participants = info  
  
 if status == 3:  
 comm.send(ip, protocol.pack\_delete\_calendar(**"0"**, calendar\_id))  
 personal\_id = db.get\_personal\_calendar(username)  
  
 for user in current\_users:  
 if current\_open\_calendars[user][0] == calendar\_id and user != username:  
 comm.send(current\_users[user], protocol.pack\_exit\_calendar(calendar\_id, username))  
 comm.send(current\_users[user], protocol.pack\_month\_events(  
 db.get\_events\_of\_calendar(current\_open\_calendars[user][0], current\_open\_calendars[user][1][:2],  
 current\_open\_calendars[user][1][3:])))  
 elif user != username:  
  
 comm.send(current\_users[user], protocol.pack\_month\_events(db.get\_events\_of\_calendar(current\_open\_calendars[user][0], current\_open\_calendars[user][1][:2], current\_open\_calendars[user][1][3:])))  
  
 elif status == 2:  
 comm.send(ip, protocol.pack\_delete\_calendar(**"0"**, calendar\_id))  
  
 for user in current\_users:  
 if current\_open\_calendars[user][0] == calendar\_id and user != username:  
 comm.send(current\_users[user], protocol.pack\_exit\_calendar(calendar\_id, username))  
 comm.send(current\_users[user], protocol.pack\_month\_events(  
 db.get\_events\_of\_calendar(current\_open\_calendars[user][0], current\_open\_calendars[user][1][:2],  
 current\_open\_calendars[user][1][3:])))  
 elif user != username:  
 if username in db.get\_calendar\_participants(current\_open\_calendars[user][0]):  
 comm.send(current\_users[user], protocol.pack\_month\_events(  
 db.get\_events\_of\_calendar(current\_open\_calendars[user][0], current\_open\_calendars[user][1][:2],  
 current\_open\_calendars[user][1][3:])))  
 else:  
 comm.send(ip, protocol.pack\_delete\_calendar(**"0"**, calendar\_id))  
 today = datetime.now()  
 for user in current\_users:  
 if user in participants and user != username:  
 comm.send(current\_users[user], protocol.pack\_delete\_calendar(**"0"**, calendar\_id))  
  
 else:  
  
 comm.send(current\_users[user], protocol.pack\_month\_events(  
 db.get\_events\_of\_calendar(current\_open\_calendars[user][0], current\_open\_calendars[user][1][:2],  
 current\_open\_calendars[user][1][3:])))  
  
  
def handle\_calendar\_ids(ip, params):  
 """  
 if user exists, send him his calendar ids  
 :param params:  
 :return:  
 """  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 comm.send(ip, protocol.pack\_calendar\_ids(db.get\_user\_calendars(username)))  
  
  
def handle\_day\_events(ip, params):  
 """  
  
 :param ip:  
 :param params: calendar\_id, date  
 :return:  
 """  
 calendar\_id, date = params  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 comm.send(ip, protocol.pack\_day\_events(db.get\_day\_events(username, calendar\_id, date)))  
  
  
def handle\_month\_events(ip, params):  
 """  
 get month events of calendar (color and date)  
 :param ip:  
 :param params: calendar\_id, month, year  
 :return:  
 """  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 calendar\_id, month, year = params  
  
 current\_open\_calendars[username] = [calendar\_id,  
 str(month).zfill(2) + **"."** + str(year)] # user: [calendar\_id, month and year]  
  
 comm.send(ip, protocol.pack\_month\_events(db.get\_events\_of\_calendar(calendar\_id, month, year)))  
  
  
def handle\_get\_calendar\_info(ip, params):  
 """  
 get calendar info and send to user  
 :param ip:  
 :param params:  
 :return:  
 """  
 username = [i for i in current\_users if current\_users[i] == ip][0]  
 info = db.get\_calendar\_info\_with\_colors(params[0])  
 if info != []:  
 data = [params[0]] + info  
 comm.send(ip, protocol.pack\_new\_calendar(**"9"**, data))  
 today = datetime.now()  
 current\_open\_calendars[username] = [params[0], str(today.month).zfill(2) + **"."** + str(today.year)] # user: [calendar\_id, month and year]  
 comm.send(ip, protocol.pack\_month\_events(db.get\_events\_of\_calendar(params[0], today.month, today.year)))  
  
  
def disconnect\_client(ip, params):  
 """  
 delete ip from current\_users and user from current\_open\_calendars  
 :param ip:  
 :param params:  
 :return:  
 """  
 # username = next(filter(lambda item: item[1] == ip, current\_users.items()), None)[0]  
 username = [i for i in current\_users if current\_users[i] == ip]  
 if len(username) != 0:  
 username = username[0]  
 del current\_users[username]  
 if username in current\_open\_calendars:  
 del current\_open\_calendars[username]  
  
  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 msg\_q = queue.Queue()  
 comm = ServerComm(4500, msg\_q)  
 db = data\_base()  
 opcodes = {**"00"**: handle\_login, **"01"**: handle\_sign\_up, **"02"**: handle\_new\_calendar, **"04"**: handle\_new\_event,  
 **"10"**: handle\_calendar\_invitation, **"11"**: handle\_is\_calendar\_accepted, **"12"**: handle\_event\_invitation,  
 **"13"**: handle\_is\_event\_accepted, **"14"**: handle\_get\_invitations, **"20"**: handle\_change\_name\_of\_calendar,  
 **"21"**: handle\_change\_name\_of\_event, **"22"**: handle\_time\_change, **"30"**: handle\_delete\_event,  
 **"31"**: handle\_exit\_calendar, **"40"**: handle\_calendar\_ids, **"41"**: handle\_day\_events, **"42"**: handle\_month\_events,  
 **"43"**: handle\_get\_calendar\_info, **"99"**: disconnect\_client}  
 current\_users = {} # username: ip  
 current\_open\_calendars = {} # user: [calendar\_id, month and year, day]  
 while True:  
 # handle msgs  
 ip, msg = msg\_q.get()  
 opcode, params = protocol.unpack(msg)  
 print(**f'got from** {ip}**:** {msg}**'**)  
 if opcode in opcodes:  
 print(opcodes[opcode](ip, params))  
 else:  
 print(**f'command number** {opcode} **not in dictionary'**)

Graphics classes:

# graphics - responsible for the graphics  
  
import wx  
from pubsub import pub  
import wx.adv  
import wx.lib.scrolledpanel  
import time  
from datetime import datetime  
from wx.lib.calendar import Calendar  
from wx.adv import CalendarCtrl, GenericCalendarCtrl, CalendarDateAttr  
  
  
# MyFrame - the frame of the graphics  
class MyFrame(wx.Frame):  
 def \_\_init\_\_(self, graphics\_q, parent=None):  
 super(MyFrame, self).\_\_init\_\_(parent, title=**"joined calendar"**, size=(900, 700))  
 self.SetMinSize(self.GetSize())  
 self.SetMaxSize(self.GetSize())  
 self.graphics\_q = graphics\_q  
 # create status bar  
 self.CreateStatusBar(1)  
  
 # create main panel - to put on the others panels  
 main\_panel = MainPanel(self)  
 box = wx.BoxSizer(wx.VERTICAL)  
 box.Add(main\_panel, 1, wx.EXPAND)  
  
 # arrange the frame  
 self.SetSizer(box)  
 self.Layout()  
 self.Show()  
  
  
   
# MainPanel - the main panel of the graphics which all the other panels rely on  
class MainPanel(wx.Panel):  
 def \_\_init\_\_(self, parent):  
 wx.Panel.\_\_init\_\_(self, parent)  
 pub.subscribe(self.show\_error\_notification, **"error"**)  
 pub.subscribe(self.show\_calendar, **"show calendar"**)  
 pub.subscribe(self.show\_invitation, **"show invitation"**)  
 pub.subscribe(self.mark\_dates, **"mark"**)  
 pub.subscribe(self.unmark\_dates, **"unmark"**)  
 pub.subscribe(self.hide\_panel, **"hide"**)  
 pub.subscribe(self.show\_day, **"show day"**)  
 pub.subscribe(self.color\_invitation, **"new invitation"**)  
 self.frame = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.v\_box = wx.BoxSizer(wx.VERTICAL)  
 # create object for each panel  
 self.login = LoginPanel(self, self.frame)  
 self.first = FirstPanel(self, self.frame)  
 self.registration = RegistrationPanel(self, self.frame)  
 self.calendar = CalendarPanel(self, self.frame)  
 self.new\_cal = newCalendarPanel(self, self.frame)  
 self.event = EventPanel(self, self.frame)  
 self.new\_cal\_parti = newCalendarParticipantPanel(self, self.frame)  
 self.new\_evt\_parti = newEventParticipantPanel(self, self.frame)  
 self.cal\_name = calNamePanel(self, self.frame)  
 self.evt\_name = evtNamePanel(self, self.frame)  
 self.new\_event = newEventPanel(self, self.frame)  
 self.evt\_time = evtTimePanel(self, self.frame)  
 self.invitations = invitationsPanel(self, self.frame)  
 self.v\_box.Add(self.login, 1, wx.EXPAND)  
 self.v\_box.Add(self.registration, 1, wx.EXPAND)  
 self.v\_box.Add(self.calendar, 1, wx.EXPAND)  
 self.v\_box.Add(self.first, 1, wx.EXPAND)  
 self.v\_box.Add(self.event, 1, wx.EXPAND)  
 self.v\_box.Add(self.new\_cal, 1, wx.EXPAND)  
 self.v\_box.Add(self.new\_cal\_parti, 1, wx.EXPAND)  
 self.v\_box.Add(self.new\_evt\_parti, 1, wx.EXPAND)  
 self.v\_box.Add(self.invitations, 1, wx.EXPAND)  
 self.v\_box.Add(self.new\_event, 1, wx.EXPAND)  
 self.v\_box.Add(self.cal\_name, 1, wx.EXPAND)  
 self.v\_box.Add(self.evt\_name, 1, wx.EXPAND)  
 self.v\_box.Add(self.evt\_time, 1, wx.EXPAND)  
  
  
 # The first panel to show  
 self.SetSizer(self.v\_box)  
 self.Layout()  
 self.first.Show()  
  
  
  
 def show\_error\_notification(self, error):  
 """  
 show notification  
 :return:  
 """  
 wx.MessageBox(error, **"Error"**, wx.OK | wx.ICON\_ERROR)  
  
  
 def show\_calendar(self, name, manager, participants):  
 """  
 show calendar  
 :param name:  
 :param manager:  
 :param participants:  
 :return:  
 """  
 self.v\_box.Detach(self.calendar)  
 self.calendar = CalendarPanel(self, self.frame, name, participants, manager)  
 self.v\_box.Add(self.calendar, 1, wx.EXPAND)  
  
 self.SetSizer(self.v\_box)  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.calendar.Show()  
  
 def show\_invitation(self, params):  
 """  
 show invitations  
 :param params: name, manager, start, end, date, participants, id or name, manager, participants, id  
 :return:  
 """  
 self.v\_box.Detach(self.invitations)  
 if len(params) == 7:  
 self.invitations = invitationsPanel(self, self.frame, params[0], params[2], params[1], params[3], params[4], params[5])  
 elif len(params) == 4:  
 self.invitations = invitationsPanel(self, self.frame, name=params[0], manager=params[1], participants=params[-2])  
 else:  
 self.invitations = invitationsPanel(self, self.frame)  
 self.v\_box.Add(self.invitations, 1, wx.EXPAND)  
 self.SetSizer(self.v\_box)  
 self.invitations.Show()  
  
 def show\_new\_cal\_parti(self, name):  
 """  
 change new calendar participant panel  
 :param name:  
 :return:  
 """  
 self.v\_box.Detach(self.new\_cal\_parti)  
 self.new\_cal\_parti = newCalendarParticipantPanel(self, self.frame, name)  
 self.v\_box.Add(self.new\_cal\_parti, 1, wx.EXPAND)  
 self.SetSizer(self.v\_box)  
  
 def hide\_panel(self, panel):  
 """  
 hide panel  
 :param panel:  
 :return:  
 """  
 if panel == **"login"**:  
 self.login.Hide()  
 elif panel == **"register"**:  
 self.registration.Hide()  
 elif panel == **"calendar"**:  
 self.frame.SetStatusText(**""**)  
 self.calendar.Hide()  
 elif panel == **"event"**:  
 self.event.Hide()  
 elif panel == **"new cal"**:  
 self.frame.SetStatusText(**""**)  
 self.new\_cal.Hide()  
 elif panel == **"new cal parti"**:  
 self.frame.SetStatusText(**""**)  
 self.new\_cal\_parti.Hide()  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.calendar.Show()  
 elif panel == **"invitation"**:  
 self.invitations.Hide()  
 elif panel == **"new evt parti"**:  
 self.frame.SetStatusText(**""**)  
 self.new\_evt\_parti.Hide()  
 self.event.Show()  
 elif panel == **"new event"**:  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.new\_event.Hide()  
 self.calendar.Show()  
 elif panel == **"cal name"**:  
 self.cal\_name.Hide()  
 elif panel == **"evt name"**:  
 self.evt\_name.Hide()  
 elif panel == **"evt time"**:  
 self.evt\_time.Hide()  
  
 def mark\_dates(self, dates):  
 """  
 mark dates with events  
 :param dates:  
 :return:  
 """  
 self.calendar.mark\_dates(dates)  
  
  
 def unmark\_dates(self, dates):  
 """  
 unmark dates with events  
 :param dates:  
 :return:  
 """  
 self.calendar.unmark\_dates(dates)  
  
  
 def show\_day(self, event):  
 """  
 show day events  
 :param events:  
 :return:  
 """  
 self.calendar.Hide()  
 self.v\_box.Detach(self.event)  
 self.event.events = event # [id, date, name, start, end, manager, [participants]]  
 if type(event) is list and len(event) == 7:  
 self.event = EventPanel(self, self.frame, event[1], event[0], event[-2], event[-3], event[2], event[3])  
 elif type(event) is list and len(event) == 5:  
 self.event = EventPanel(self, self.frame, participants=event[0], start=event[1], end=event[2], date=event[3])  
 else:  
 self.event = EventPanel(self, self.frame, date=event)  
  
 self.v\_box.Add(self.event, 1, wx.EXPAND)  
 self.SetSizer(self.v\_box)  
 self.event.Show()  
  
  
 def show\_new\_event(self, date):  
 """  
 show events on screen  
 :param date:  
 :return:  
 """  
 self.v\_box.Detach(self.new\_event)  
 self.new\_event = newEventPanel(self, self.frame, date)  
 self.v\_box.Add(self.new\_event, 1, wx.EXPAND)  
 self.SetSizer(self.v\_box)  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
 self.new\_event.Show()  
  
 def color\_invitation(self):  
 """  
 color the invitation button green  
 :return:  
 """  
 self.calendar.invitationBtn.SetForegroundColour(wx.GREEN)  
 self.Refresh()  
  
 def uncolor\_invitation(self):  
 """  
 color the invitation button grey  
 :return:  
 """  
 self.calendar.invitationBtn.SetForegroundColour(wx.BLACK)  
 self.Refresh()  
  
  
# LoginPanel - responsible for the log in  
class LoginPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 title = wx.StaticText(self,-1, label=**"Login"**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # username  
 nameBox = wx.BoxSizer(wx.HORIZONTAL)  
 nameText = wx.StaticText(self, 1, label=**"Username: "**)  
 nameText.SetFont(smallfont)  
 self.nameField = wx.TextCtrl(self, -1, name=**"username"**, size=(300, 50))  
 self.nameField.SetFont(smallfont)  
 nameBox.Add(nameText, 0, wx.ALL, 5)  
 nameBox.Add(self.nameField, 0, wx.ALL, 5)  
 # password  
 passBox = wx.BoxSizer(wx.HORIZONTAL)  
 passText = wx.StaticText(self, 1, label=**"Password: "**)  
 passText.SetFont(smallfont)  
 self.passField = wx.TextCtrl(self, -1, name=**"password"**, style=wx.TE\_PASSWORD, size=(300, 50))  
 self.passField.SetFont(smallfont)  
 passBox.Add(passText, 0, wx.ALL, 5)  
 passBox.Add(self.passField, 0, wx.ALL, 5)  
 login = wx.Image(**"pics**\\**done.png"**)  
 login.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 login = wx.Bitmap(login)  
 logBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=login)  
 logBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 logBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 logBtn.Bind(wx.EVT\_BUTTON, self.handle\_login)  
 btnBox.Add(logBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(40)  
 sizer.Add(title, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(60)  
 sizer.Add(nameBox,0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(20)  
 sizer.Add(passBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(30)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
  
  
 def handle\_login(self, event):  
 """  
 try to login  
 :param event:  
 :return:  
 """  
 username = self.nameField.GetValue()  
 password = self.passField.GetValue()  
 if not username or not password :  
 self.frame.SetStatusText(**"Must enter name and password"**)  
 else:  
 self.frame.graphics\_q.put((**"login"**, (username, password)))  
  
 def handle\_back(self, event):  
 """  
 go back to first panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.first.Show()  
  
  
# RegisterPanel - responsible for the registration  
class RegistrationPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 title = wx.StaticText(self, -1, label=**"Register"**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 # username  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 nameBox = wx.BoxSizer(wx.HORIZONTAL)  
 nameText = wx.StaticText(self, 1, label=**"Username: "**)  
 nameText.SetForegroundColour(wx.BLACK)  
 nameText.SetFont(smallfont)  
 self.nameField = wx.TextCtrl(self, -1, name=**"username"**, size=(300,50))  
 self.nameField.SetFont(smallfont)  
 nameBox.Add(nameText, 0, wx.ALL, 5)  
 nameBox.Add(self.nameField, 0, wx.ALL, 5)  
 # password  
 passBox = wx.BoxSizer(wx.HORIZONTAL)  
 passText = wx.StaticText(self, 1, label=**"Password: "**)  
 passText.SetForegroundColour(wx.BLACK)  
 passText.SetFont(smallfont)  
 self.passField = wx.TextCtrl(self, -1, name=**"password"**, style=wx.TE\_PASSWORD, size=(300, 50))  
 self.passField.SetFont(smallfont)  
 passBox.Add(passText, 0, wx.ALL, 5)  
 passBox.Add(self.passField, 0, wx.ALL, 5)  
 # phone number  
 phoneBox = wx.BoxSizer(wx.HORIZONTAL)  
 phoneText = wx.StaticText(self, 1, label=**"Phone: "**)  
 phoneText.SetForegroundColour(wx.BLACK)  
 phoneText.SetFont(smallfont)  
 self.phoneField = wx.TextCtrl(self, -1, name=**"Phone"**, size=(300, 50))  
 self.phoneField.SetFont(smallfont)  
 phoneBox.Add(phoneText, 0, wx.ALL, 5)  
 phoneBox.Add(self.phoneField, 0, wx.ALL, 5)  
 register = wx.Image(**"pics**\\**done.png"**)  
 register.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 register = wx.Bitmap(register)  
 regBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=register)  
 regBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 regBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 regBtn.Bind(wx.EVT\_BUTTON, self.handle\_registration)  
 btnBox.Add(regBtn, 0, wx.ALL, 5)  
 # add all elements to sizer  
 sizer.AddSpacer(40)  
 sizer.Add(title, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(60)  
 sizer.Add(nameBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(20)  
  
 sizer.Add(passBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(20)  
  
 sizer.Add(phoneBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(30)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
  
 def handle\_registration(self, event):  
 """  
 try to register  
 :param event:  
 :return:  
 """  
 username = self.nameField.GetValue()  
 password = self.passField.GetValue()  
 phone = self.phoneField.GetValue()  
 if not username or not password or not phone:  
 self.frame.SetStatusText(**"Must enter name, password and phone"**)  
 else:  
 self.frame.graphics\_q.put((**"register"**, (username, password, phone)))  
  
 def handle\_back(self, event):  
 """  
 go bacl to first screen  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.first.Show()  
  
  
# FirstPanel - the first panel the user sees. in it you choose if you want to register or to log in  
class FirstPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 title = wx.StaticText(self, -1, label=**"joined calendar"**)  
 titlefont = wx.Font(50, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 # login & registration buttons  
 btnfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 loginBtn = wx.Button(self, wx.ID\_ANY, label=**"login"**, size=(300, 300))  
 loginBtn.SetFont(btnfont)  
 loginBtn.Bind(wx.EVT\_BUTTON, self.handle\_login)  
 btnBox.Add(loginBtn, 0, wx.ALL, 5)  
 btnBox.AddSpacer(50)  
 # add all elements to sizer  
 sizer.AddSpacer(50)  
 sizer.Add(title, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(100)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
  
 regBtn = wx.Button(self, wx.ID\_ANY, label=**"register"**, size=(300, 300))  
 regBtn.SetFont(btnfont)  
 regBtn.Bind(wx.EVT\_BUTTON, self.handle\_reg)  
 btnBox.Add(regBtn, 1, wx.ALL, 5)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_login(self, event):  
 """  
 open login panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**''**)  
 self.Hide()  
 self.parent.login.Show()  
  
 def handle\_reg(self, event):  
 """  
 open registration panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**''**)  
 self.Hide()   
 self.parent.registration.Show()  
  
  
  
# CalendarPanel - shows the calendar and connects to all of it's abilities  
class CalendarPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, name=**"personal"**, participants=None, manager=**"1"**):  
 wx.Panel.\_\_init\_\_(self, parent, pos=parent.GetPosition(), size=parent.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.events = []  
 self.name = name  
  
 self.title\_row = wx.BoxSizer(wx.HORIZONTAL)  
 # title  
 self.title = wx.StaticText(self, -1, label=name)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 calfont = wx.Font(35, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_NORMAL)  
 self.title.SetForegroundColour(wx.BLACK)  
 self.title.SetFont(titlefont)  
 #arrrow button  
 left = wx.Image(**"pics**\\**left.png"**)  
 left.Rescale(100, 80)  
 right = wx.Image(**"pics**\\**right.png"**)  
 right.Rescale(100, 80)  
 left = wx.Bitmap(left)  
 right = wx.Bitmap(right)  
 self.leftBtn = wx.BitmapButton(self,  
 size=(100, 80), pos=(100,50), name=**"button"**, bitmap=left)  
 self.leftBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.leftBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 self.rightBtn = wx.BitmapButton(self,  
 size=(100, 80), pos=(660, 50), name=**"button"**, bitmap=right)  
 self.rightBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.rightBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 self.leftBtn.Bind(wx.EVT\_BUTTON, self.left\_cal)  
 self.rightBtn.Bind(wx.EVT\_BUTTON, self.right\_cal)  
  
 # set bmp as bitmap for button  
  
 self.title\_row.Add(self.leftBtn, 1, wx.ALL, 5)  
 self.title\_row.Add(self.rightBtn, 1, wx.ALL, 5)  
 btnBox2 = wx.BoxSizer(wx.HORIZONTAL)  
 changeBtn = wx.Button(self, wx.ID\_ANY, label=**"change name"**, size=(100, 40), pos=(765,95))  
 changeBtn.Bind(wx.EVT\_BUTTON, self.change\_calendar\_name)  
 btnBox2.Add(changeBtn, 0, wx.ALL, 5)  
 addBtn = wx.Button(self, wx.ID\_ANY, label=**"add participant"**, size=(100, 40), pos=(765,50))  
 addBtn.Bind(wx.EVT\_BUTTON, self.add\_calendar\_participant)  
 btnBox2.Add(addBtn, 0, wx.ALL, 5)  
  
  
  
 self.title\_row.Add(self.title, 1, wx.ALL, 5)  
 self.title\_row.Add(btnBox2, 0, wx.ALL, 5)  
  
  
 self.mainbox = wx.BoxSizer(wx.VERTICAL)  
  
  
 self.btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 self.partBtn = wx.Button(self, wx.ID\_ANY, label=**"participants"**, size=(100, 40))  
 self.partBtn.Bind(wx.EVT\_BUTTON, self.display)  
 self.btnBox.Add(self.partBtn, 1, wx.ALL, 5)  
 self.invitationBtn = wx.Button(self, wx.ID\_ANY, label=**"invitations"**, size=(100, 40))  
 self.invitationBtn.Bind(wx.EVT\_BUTTON, self.show\_invitations)  
 self.btnBox.Add(self.invitationBtn, 1, wx.ALL, 5)  
 self.newBtn = wx.Button(self, wx.ID\_ANY, label=**"New calendar"**, size=(100, 40))  
 self.newBtn.Bind(wx.EVT\_BUTTON, self.new\_calendar)  
 self.btnBox.Add(self.newBtn, 1, wx.ALL, 5)  
 self.btnBox.AddSpacer(430)  
 delBtn = wx.Button(self, wx.ID\_ANY, label=**"Exit calendar"**, size=(100, 40))  
 delBtn.Bind(wx.EVT\_BUTTON, self.exit\_calendar)  
 self.btnBox.Add(delBtn, 0, wx.ALL, 5)  
 self.myScrolled = ParticipantsPanel(self, self.frame,self.partBtn, participants)  
  
  
 self.mainbox.Add(self.btnBox)  
  
 self.mainbox.AddSpacer(20)  
  
  
 self.mainbox.Add(self.title, 0, wx.CENTER)  
  
  
 calSizer = wx.BoxSizer(wx.VERTICAL)  
  
  
 self.cal = GenericCalendarCtrl(self, -1, wx.DateTime().Today(),  
 style = wx.adv.CAL\_SUNDAY\_FIRST  
 | wx.adv.CAL\_SEQUENTIAL\_MONTH\_SELECTION,name=**"cal-2"**,size=(720,450))  
  
 self.cal.SetFont(calfont)  
  
  
  
  
  
 self.OnChangeMonth()  
 # bind some event handlers to each calendar  
 self.cal.Bind(wx.adv.EVT\_CALENDAR, self.OnCalSelected)  
 self.cal.Bind(wx.adv.EVT\_CALENDAR\_MONTH, self.OnChangeMonth)  
 self.cal.Bind(wx.adv.EVT\_CALENDAR\_SEL\_CHANGED, self.OnCalSelChanged)  
 self.cal.Bind(wx.adv.EVT\_CALENDAR\_WEEKDAY\_CLICKED, self.OnCalWeekdayClicked)  
 today = datetime.now()  
 self.current\_month = str(today.month).zfill(2)  
  
 # create some sizers for layout  
 self.mainbox.AddSpacer(20)  
 self.mainbox.Add(self.cal,0, wx.ALL | wx.CENTER,5)  
  
  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.SetSizer(self.mainbox)  
 # arrange the screen  
 self.Layout()  
 self.Hide()  
  
  
  
 def OnCalSelected(self, evt):  
 print(**'OnCalSelected: %s**\n**'** % evt.Date)  
 cal = evt.GetEventObject()  
 print(**'name: %s**\n**'** % cal.GetName())  
  
 def OnCalWeekdayClicked(self, evt):  
 print(**'OnCalWeekdayClicked: %s**\n**'** % evt.GetWeekDay())  
  
 cal = evt.GetEventObject()  
 print(**'name: %s**\n**'** % cal.GetName())  
  
 def OnCalSelChanged(self, evt):  
 """  
 open day or change month (depends on where the user clicked)  
 :param evt:  
 :return:  
 """  
 cal = evt.GetEventObject()  
 month = str(cal.GetDate())[4:7]  
 monthes = [**"Jan"**, **"Feb"**, **"Mar"**, **"Apr"**, **"May"**, **"Jun"**, **"Jul"**, **"Aug"**, **"Sep"**, **"Oct"**, **"Nov"**, **"Dec"**]  
 month\_num = str(monthes.index(month) + 1).zfill(2)  
 year = str(cal.GetDate())[-4:]  
 print(month\_num)  
 print(self.current\_month)  
 if month\_num != self.current\_month: # month change  
 self.current\_month = month\_num  
 self.frame.graphics\_q.put((**"month"**, (month\_num, year)))  
 else: # day sellected  
 day = str(cal.GetDate())[8:10]  
 day = day.replace(**" "**, **"0"**)  
 self.frame.graphics\_q.put((**"day"**, (day+**"."**+month\_num+**"."**+year)))  
 self.Hide()  
  
  
  
 def OnChangeMonth(self, evt=None):  
 if evt is None:  
 cal = self.cal  
 else:  
 cal = evt.GetEventObject()  
  
 def display(self, evt):  
 """  
 display participants scrolled panel  
 :param evt:  
 :return:  
 """  
 self.partBtn.Hide()  
 self.myScrolled.Show()  
  
 def new\_calendar(self, evt):  
 """  
 open new calendar panel  
 :param evt:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.new\_cal.Show()  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
  
  
  
 def left\_cal(self, evt):  
 """  
 scroll to left calendar if possible  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"left"**, ()))  
  
 def right\_cal(self, evt):  
 """  
 scroll to right calendar if possible  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"right"**, ()))  
  
  
 def change\_calendar\_name(self, evt):  
 """  
 open calendar name change panel  
 :param evt:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.cal\_name.Show()  
  
 def add\_calendar\_participant(self, evt):  
 """  
 open add calendar participants panel  
 :param evt:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.show\_new\_cal\_parti(self.name)  
 self.parent.new\_cal\_parti.Show()  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
  
  
 def show\_invitations(self, evt):  
 """  
 open invitation panel  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"invitations"**, ()))  
  
 def exit\_calendar(self, evt):  
 """  
 exit calendar  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"exit cal"**, ()))  
  
  
 def mark\_dates(self, dates):  
 """  
 mark dates in calendar  
 :param dates:  
 :return:  
 """  
 for i in dates:  
 self.cal.SetAttr(i[0], wx.adv.CalendarDateAttr(colBack=i[1]))  
  
 self.Refresh()  
  
 def unmark\_dates(self, dates):  
 """  
 unmark dates in calendar  
 :param dates:  
 :return:  
 """  
 for i in dates:  
 self.cal.ResetAttr(i)  
 self.Layout()  
  
  
  
# ParticipantsPanel - responsible for showing the calendar's participants on the calendar panel when the participants button is clicked  
class ParticipantsPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, btn, participants):  
 wx.Panel.\_\_init\_\_(self, parent,size=(100,140),pos=(0,0))  
 self.frame = frame  
 self.parent = parent  
 if participants:  
 self.participants = participants  
 else:  
 self.participants = [**"1"**, **"2"**, **"3"**]  
 self.participantsBTN = btn  
  
 self.main\_box = wx.BoxSizer(wx.VERTICAL)  
  
  
 # title  
 title = wx.StaticText(self, -1, label=**"participants"**)  
 titlefont = wx.Font(15, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
  
  
 self.main\_box.Add(title, 0, wx.LEFT)  
  
  
  
  
 self.scrollP = wx.lib.scrolledpanel.ScrolledPanel(self,-1,style=wx.TAB\_TRAVERSAL | wx.SUNKEN\_BORDER, size=(100,80))  
 self.scrollP.SetAutoLayout(1)  
 self.scrollP.SetupScrolling()  
  
 self.main\_box.Add(self.scrollP,0, wx.EXPAND)  
  
  
 # button  
 self.closeBtn = wx.Button(self, wx.ID\_ANY, label=**"close"**, size=(100, 20))  
 self.closeBtn.Bind(wx.EVT\_BUTTON, self.close)  
  
 self.display\_participants()  
  
 self.SetSizer(self.main\_box)  
  
 self.Hide()  
  
  
  
  
 def close(self, evt):  
 """  
 close scrolled panel and return the button  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.participantsBTN.Show()  
  
  
 def display\_participants(self):  
 """  
 display scrolled panel of participants in calendar  
 :return:  
 """  
 self.spSizer = wx.BoxSizer(wx.VERTICAL)  
 if self.participants != [**"1"**, **"2"**, **"3"**]:  
 for parti in self.participants:  
 print(**"parti"**, parti)  
 user = wx.TextCtrl(self.scrollP, value=parti[0])  
 user.SetForegroundColour(parti[1])  
 self.spSizer.Add(user)  
 self.scrollP.SetSizer(self.spSizer)  
  
 if self.closeBtn:  
 self.closeBtn.Destroy()  
  
  
 self.closeBtn = wx.Button(self, wx.ID\_ANY, label=**"close"**, size=(100, 20))  
 self.closeBtn.Bind(wx.EVT\_BUTTON, self.close)  
 self.main\_box.Add(self.closeBtn, 0, wx.ALL, 5)  
  
  
 self.SetSizer(self.main\_box)  
 self.Layout()  
  
  
  
# EventPanel - show the events of the chosen day and connect to it's abilities  
class EventPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, name=**"&&&"**, participants=None, manager=**"$$$"**, date=**"01.01.2024"**, start = **"16:00"**, end=**"18:00"**):  
 wx.Panel.\_\_init\_\_(self, parent, pos=parent.GetPosition(), size=parent.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.events = []  
 self.date = date  
 mainbox = wx.BoxSizer(wx.VERTICAL)  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 self.backBtn = wx.Button(self, wx.ID\_ANY, label=**"back"**, size=(100, 40))  
 self.backBtn.Bind(wx.EVT\_BUTTON, self.go\_back)  
 btnBox.Add(self.backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(650)  
 newBtn = wx.Button(self, wx.ID\_ANY, label=**"new event"**, size=(100, 40))  
 newBtn.Bind(wx.EVT\_BUTTON, self.new\_event)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
 mainbox.AddSpacer(10)  
 mainbox.Add(btnBox)  
 if participants:  
 title\_row = wx.BoxSizer(wx.HORIZONTAL)  
 # title  
 title = wx.StaticText(self, -1, label=date)  
 titlefont = wx.Font(45, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 eventfont = wx.Font(25, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 smallTitleFont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.BOLD)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 #arrrow button  
 left = wx.Image(**"pics**\\**left.png"**)  
 left.Rescale(100, 80)  
 right = wx.Image(**"pics**\\**right.png"**)  
 right.Rescale(100, 80)  
 left = wx.Bitmap(left)  
 right = wx.Bitmap(right)  
 self.leftBtn = wx.BitmapButton(self,  
 size=(100, 80), pos=(100,50), name=**"button"**, bitmap=left)  
 self.leftBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.leftBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 self.rightBtn = wx.BitmapButton(self,  
 size=(100, 80), pos=(660, 50), name=**"button"**, bitmap=right)  
 self.rightBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.rightBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 self.leftBtn.Bind(wx.EVT\_BUTTON, self.left\_event)  
 self.rightBtn.Bind(wx.EVT\_BUTTON, self.right\_event)  
  
 # set bmp as bitmap for button  
 title\_row.Add(self.leftBtn, 1, wx.ALL, 5)  
 title\_row.AddSpacer(200)  
 title\_row.Add(title, 1, wx.ALL, 5)  
 title\_row.AddSpacer(170)  
 title\_row.Add(self.rightBtn, 1, wx.ALL, 5)  
  
 if manager != **"&&&"** and name != **"&&&"**:  
 manager\_row = wx.BoxSizer(wx.HORIZONTAL)  
 manager = wx.StaticText(self, -1, label=manager)  
 manager.SetForegroundColour(wx.BLACK)  
 manager.SetFont(eventfont)  
 manager\_title = wx.StaticText(self, -1, label=**"Manager: "**)  
 manager\_title.SetForegroundColour(wx.BLACK)  
 manager\_title.SetFont(smallTitleFont)  
 manager\_row.Add(manager\_title)  
 manager\_row.Add(manager)  
 name\_row = wx.BoxSizer(wx.HORIZONTAL)  
 name = wx.StaticText(self, -1, label=name)  
 name.SetForegroundColour(wx.BLACK)  
 name.SetFont(eventfont)  
 name\_title = wx.StaticText(self, -1, label=**"Name: "**)  
 name\_title.SetForegroundColour(wx.BLACK)  
 name\_title.SetFont(smallTitleFont)  
 name\_row.Add(name\_title)  
 name\_row.Add(name)  
 participants\_str = **", "**.join(participants)  
 participants\_row = wx.BoxSizer(wx.HORIZONTAL)  
 parti = wx.StaticText(self, -1, label=participants\_str)  
 parti.SetForegroundColour(wx.BLACK)  
 parti.SetFont(eventfont)  
 parti\_title = wx.StaticText(self, -1, label=**"Participants: "**)  
 parti\_title.SetForegroundColour(wx.BLACK)  
 parti\_title.SetFont(smallTitleFont)  
 participants\_row.Add(parti\_title)  
 participants\_row.Add(parti)  
 time\_str = start + **" - "** + end  
 time\_row = wx.BoxSizer(wx.HORIZONTAL)  
 time = wx.StaticText(self, -1, label=time\_str)  
 time.SetForegroundColour(wx.BLACK)  
 time.SetFont(eventfont)  
 time\_title = wx.StaticText(self, -1, label=**"Time: "**)  
 time\_title.SetForegroundColour(wx.BLACK)  
 time\_title.SetFont(smallTitleFont)  
 time\_row.Add(time\_title)  
 time\_row.Add(time)  
  
 buttons = wx.BoxSizer(wx.HORIZONTAL)  
 self.editTimeBtn = wx.Button(self, wx.ID\_ANY, label=**"change time"**, size=(100, 40))  
 self.editTimeBtn.Bind(wx.EVT\_BUTTON, self.change\_time)  
 buttons.Add(self.editTimeBtn, 1, wx.ALL, 5)  
 buttons.AddSpacer(10)  
 editNameBtn = wx.Button(self, wx.ID\_ANY, label=**"change name"**, size=(100, 40))  
 editNameBtn.Bind(wx.EVT\_BUTTON, self.change\_name)  
 buttons.Add(editNameBtn, 0, wx.ALL, 5)  
 buttons.AddSpacer(10)  
 delBtn = wx.Button(self, wx.ID\_ANY, label=**"delete event"**, size=(100, 40))  
 delBtn.Bind(wx.EVT\_BUTTON, self.del\_event)  
 buttons.Add(delBtn, 0, wx.ALL, 5)  
 buttons.AddSpacer(10)  
 addBtn = wx.Button(self, wx.ID\_ANY, label=**"Add participants"**, size=(100, 40))  
 addBtn.Bind(wx.EVT\_BUTTON, self.add\_evt\_parti)  
 buttons.Add(addBtn, 0, wx.ALL, 5)  
  
 mainbox.AddSpacer(40)  
 mainbox.Add(title\_row, 0, wx.CENTER)  
 if manager != **"&&&"** and name != **"&&&"**:  
 mainbox.AddSpacer(40)  
 mainbox.Add(name\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(20)  
 mainbox.Add(participants\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(20)  
 mainbox.Add(time\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(20)  
 mainbox.Add(manager\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(50)  
 else:  
 mainbox.AddSpacer(50)  
 mainbox.Add(participants\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(30)  
 mainbox.Add(time\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(60)  
  
 mainbox.Add(buttons, 0, wx.CENTER)  
  
 else:  
 # title  
 words = wx.BoxSizer(wx.VERTICAL)  
 title = wx.StaticText(self, -1, label=**"There are no events on"**)  
 titlefont = wx.Font(45, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 date = wx.StaticText(self, -1, label=date)  
 date.SetForegroundColour(wx.BLACK)  
 date.SetFont(titlefont)  
 words.Add(title, 0, wx.CENTER)  
 words.AddSpacer(30)  
 words.Add(date, 0, wx.CENTER)  
 mainbox.AddSpacer(200)  
 mainbox.Add(words, 1, wx.CENTER)  
  
 self.SetSizer(mainbox)  
 self.Layout()  
 self.Hide()  
  
  
 def left\_event(self, evt):  
 """  
 scroll to left event if possible  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"left event"**, ()))  
  
 def right\_event(self, evt):  
 """  
 scroll to right event if possible  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"right event"**, ()))  
  
 def go\_back(self, evt):  
 """  
 go back to calendar panel  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.parent.calendar.Show()  
  
 def new\_event(self, evt):  
 """  
 go to new event panel  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
 self.parent.show\_new\_event(self.date)  
  
  
 def del\_event(self, evt):  
 """  
 try to delete event  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"del evt"**, ()))  
  
 def change\_name(self, evt):  
 """  
 ope change event name panel  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.parent.evt\_name.Show()  
  
 def change\_time(self, evt):  
 """  
 open change event time panel  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.parent.evt\_time.Show()  
  
 def add\_evt\_parti(self, evt):  
 """  
 open add event participants panel  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
 self.parent.new\_evt\_parti.Show()  
  
  
  
# newCalendarPanel - responsible for making new calendars  
class newCalendarPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 title = wx.StaticText(self, -1, label=**"New calendar"**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # name  
 nameBox = wx.BoxSizer(wx.HORIZONTAL)  
 nameText = wx.StaticText(self, 1, label=**"Name of calendar: "**)  
 nameText.SetFont(smallfont)  
 self.nameField = wx.TextCtrl(self, -1, name=**"name"**, size=(300, 50))  
 self.nameField.SetFont(smallfont)  
 nameBox.Add(nameText, 0, wx.ALL, 5)  
 nameBox.Add(self.nameField, 0, wx.ALL, 5)  
 # participants  
 partiBox = wx.BoxSizer(wx.HORIZONTAL)  
 partiText = wx.StaticText(self, 1, label=**"Participants to add: "**)  
 partiText.SetFont(smallfont)  
 self.partiField = wx.TextCtrl(self, -1, name=**"participants"**, size=(300, 50))  
 self.partiField.SetFont(smallfont)  
 partiBox.Add(partiText, 0, wx.ALL, 5)  
 partiBox.Add(self.partiField, 0, wx.ALL, 5)  
  
 newC = wx.Image(**"pics**\\**done.png"**)  
 newC.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newC = wx.Bitmap(newC)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newC)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_calendar)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(40)  
 sizer.Add(title, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(60)  
 sizer.Add(nameBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(20)  
 sizer.Add(partiBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(30)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_calendar(self, event):  
 """  
 try to open new calendar  
 :param event:  
 :return:  
 """  
 name = self.nameField.GetValue()  
 participants = self.partiField.GetValue()  
 if not name or not participants:  
 self.frame.SetStatusText(**"Must enter name and participants"**)  
 else:  
 participants = participants.split(**", "**)  
 self.frame.graphics\_q.put((**"new cal"**, (name, participants)))  
  
 def handle\_back(self, event):  
 """  
 go back to event panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.parent.calendar.Show()  
  
  
  
# newCalendarParticipantPanel - responsible for addind participants to calendar  
class newCalendarParticipantPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, calendar\_name=**"personal"**):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 title\_row = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=calendar\_name)  
 title.SetForegroundColour(wx.BLACK)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 titleStart = wx.StaticText(self, -1, label=**"Add participants to "**)  
 titleStart.SetForegroundColour(wx.BLACK)  
 titleStart.SetFont(titlefont)  
 title\_row.Add(titleStart)  
 title\_row.Add(title)  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # participants  
 partiBox = wx.BoxSizer(wx.HORIZONTAL)  
 partiText = wx.StaticText(self, 1, label=**"Participants to add: "**)  
 partiText.SetFont(smallfont)  
 self.partiField = wx.TextCtrl(self, -1, name=**"participants"**, size=(300, 50))  
 self.partiField.SetFont(smallfont)  
 partiBox.Add(partiText, 0, wx.ALL, 5)  
 partiBox.Add(self.partiField, 0, wx.ALL, 5)  
  
 newP = wx.Image(**"pics**\\**done.png"**)  
 newP.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newP = wx.Bitmap(newP)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newP)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_calendar\_parti)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(60)  
 sizer.Add(title\_row, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(100)  
 sizer.Add(partiBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(50)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
 self.frame.SetStatusText(**"participants names must be seperated with ' ,'"**)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_calendar\_parti(self, event):  
 """  
 try to add participants to calendar  
 :param event:  
 :return:  
 """  
 participants = self.partiField.GetValue()  
 if not participants :  
 self.frame.SetStatusText(**"Must enter participants"**)  
 else:  
 print(participants)  
 participants = participants.split(**", "**)  
 self.frame.graphics\_q.put((**"new cal parti"**, (participants)))  
  
 def handle\_back(self, event):  
 """  
 go back to calendar panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.parent.calendar.Show()  
  
  
  
# invitationsPanel - responsible for showing invitations and accept / decline them  
class invitationsPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, name=**"test"**, participants=None, manager=**"1"**, date=**"&&&"**, start = **"16:00"**, end=**"18:00"**):  
 wx.Panel.\_\_init\_\_(self, parent, pos=parent.GetPosition(), size=parent.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.events = []  
 mainbox = wx.BoxSizer(wx.VERTICAL)  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 self.backBtn = wx.Button(self, wx.ID\_ANY, label=**"back"**, size=(100, 40))  
 self.backBtn.Bind(wx.EVT\_BUTTON, self.go\_back)  
 btnBox.Add(self.backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(650)  
 mainbox.Add(btnBox)  
 mainbox.AddSpacer(10)  
  
 if participants:  
 # title  
 title\_row = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=name)  
 titlefont = wx.Font(45, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 eventfont = wx.Font(25, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 smallTitleFont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.BOLD)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 #arrrow button  
 left = wx.Image(**"pics**\\**left.png"**)  
 left.Rescale(100, 80)  
 right = wx.Image(**"pics**\\**right.png"**)  
 right.Rescale(100, 80)  
 left = wx.Bitmap(left)  
 right = wx.Bitmap(right)  
 # create button at point (20, 20)  
 self.leftBtn = wx.BitmapButton(self,  
 size=(100, 80), pos=(100,50), name=**"button"**, bitmap=left)  
 self.leftBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.leftBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 self.rightBtn = wx.BitmapButton(self,  
 size=(100, 80), pos=(660, 50), name=**"button"**, bitmap=right)  
 self.rightBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.rightBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 self.leftBtn.Bind(wx.EVT\_BUTTON, self.left\_invitation)  
 self.rightBtn.Bind(wx.EVT\_BUTTON, self.right\_invitation)  
  
 # set bmp as bitmap for button  
 title\_row.Add(self.leftBtn, 1, wx.ALL, 5)  
 title\_row.AddSpacer(200)  
 title\_row.Add(title, 1, wx.ALL, 5)  
 title\_row.AddSpacer(200)  
 title\_row.Add(self.rightBtn, 1, wx.ALL, 5)  
  
 if date != **"&&&"**:  
 date\_row = wx.BoxSizer(wx.HORIZONTAL)  
 date = wx.StaticText(self, -1, label=date)  
 date.SetForegroundColour(wx.BLACK)  
 date.SetFont(eventfont)  
 date\_title = wx.StaticText(self, -1, label=**"Date: "**)  
 date\_title.SetForegroundColour(wx.BLACK)  
 date\_title.SetFont(smallTitleFont)  
 date\_row.Add(date\_title)  
 date\_row.Add(date)  
  
 time\_str = start + **" - "** + end  
 time\_row = wx.BoxSizer(wx.HORIZONTAL)  
 time = wx.StaticText(self, -1, label=time\_str)  
 time.SetForegroundColour(wx.BLACK)  
 time.SetFont(eventfont)  
 time\_title = wx.StaticText(self, -1, label=**"Time: "**)  
 time\_title.SetForegroundColour(wx.BLACK)  
 time\_title.SetFont(smallTitleFont)  
 time\_row.Add(time\_title)  
 time\_row.Add(time)  
  
 participants\_str = **", "**.join(participants)  
 participants\_row = wx.BoxSizer(wx.HORIZONTAL)  
 parti = wx.StaticText(self, -1, label=participants\_str)  
 parti.SetForegroundColour(wx.BLACK)  
 parti.SetFont(eventfont)  
 parti\_title = wx.StaticText(self, -1, label=**"Participants: "**)  
 parti\_title.SetForegroundColour(wx.BLACK)  
 parti\_title.SetFont(smallTitleFont)  
 participants\_row.Add(parti\_title)  
 participants\_row.Add(parti)  
  
 manager\_row = wx.BoxSizer(wx.HORIZONTAL)  
 manager = wx.StaticText(self, -1, label=manager)  
 manager.SetForegroundColour(wx.BLACK)  
 manager.SetFont(eventfont)  
 manager\_title = wx.StaticText(self, -1, label=**"Invited by: "**)  
 manager\_title.SetForegroundColour(wx.BLACK)  
 manager\_title.SetFont(smallTitleFont)  
 manager\_row.Add(manager\_title)  
 manager\_row.Add(manager)  
  
 buttons = wx.BoxSizer(wx.HORIZONTAL)  
 acceptBtn = wx.Button(self, wx.ID\_ANY, label=**"accept"**, size=(100, 40))  
 acceptBtn.Bind(wx.EVT\_BUTTON, self.accept)  
 buttons.Add(acceptBtn, 0, wx.ALL, 5)  
 buttons.AddSpacer(30)  
 declineBtn = wx.Button(self, wx.ID\_ANY, label=**"decline"**, size=(100, 40))  
 declineBtn.Bind(wx.EVT\_BUTTON, self.decline)  
 buttons.Add(declineBtn, 0, wx.ALL, 5)  
  
 mainbox.AddSpacer(40)  
 mainbox.Add(title\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(40)  
 if date != **"&&&"**:  
 mainbox.Add(date\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(20)  
 mainbox.Add(time\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(20)  
 mainbox.Add(participants\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(20)  
 mainbox.Add(manager\_row, 0, wx.CENTER)  
 mainbox.AddSpacer(50)  
 mainbox.Add(buttons, 0, wx.CENTER)  
  
 else:  
 # title  
 print(**"hereeeee"**)  
 print(date)  
 mainbox.AddSpacer(200)  
 title = wx.StaticText(self, -1, label=**"There are no invitations"**)  
 titlefont = wx.Font(45, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 mainbox.Add(title, 0, wx.CENTER)  
 self.parent.uncolor\_invitation()  
  
 self.SetSizer(mainbox)  
 self.Layout()  
 self.Hide()  
  
  
 def left\_invitation(self, evt):  
 """  
 scroll to left invitation if possible  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"left invitation"**, ()))  
  
 def right\_invitation(self, evt):  
 """  
 scroll to right invitation if possible  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"right invitation"**, ()))  
  
 def go\_back(self, evt):  
 """  
 go back to calendar panel  
 :param evt:  
 :return:  
 """  
 self.Hide()  
 self.frame.SetStatusText(**"Red is the joined color"**)  
 self.parent.calendar.Show()  
  
 def accept(self, evt):  
 """  
 accept invitation  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"response"**, (**"0"**)))  
  
 def decline(self, evt):  
 """  
 decline invitation  
 :param evt:  
 :return:  
 """  
 self.frame.graphics\_q.put((**"response"**, (**"1"**)))  
  
  
  
# newEventParticipantPanel - responsible for adding participants to event  
class newEventParticipantPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, event\_name=**"test"**):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 title\_row = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=event\_name)  
 title.SetForegroundColour(wx.BLACK)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 titleStart = wx.StaticText(self, -1, label=**"Add participants to "**)  
 titleStart.SetForegroundColour(wx.BLACK)  
 titleStart.SetFont(titlefont)  
 title\_row.Add(titleStart)  
 title\_row.Add(title)  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # participants  
 partiBox = wx.BoxSizer(wx.HORIZONTAL)  
 partiText = wx.StaticText(self, 1, label=**"Participants to add: "**)  
 partiText.SetFont(smallfont)  
 self.partiField = wx.TextCtrl(self, -1, name=**"participants"**, size=(300, 50))  
 self.partiField.SetFont(smallfont)  
 partiBox.Add(partiText, 0, wx.ALL, 5)  
 partiBox.Add(self.partiField, 0, wx.ALL, 5)  
  
 newP = wx.Image(**"pics**\\**done.png"**)  
 newP.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newP = wx.Bitmap(newP)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newP)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_event\_parti)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(60)  
 sizer.Add(title\_row, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(100)  
 sizer.Add(partiBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(50)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
 self.frame.SetStatusText(**"participants names must be seperated with ' ,'"**)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_event\_parti(self, event):  
 """  
 try to add participants to event  
 :param event:  
 :return:  
 """  
 participants = self.partiField.GetValue()  
 if not participants :  
 self.frame.SetStatusText(**"Must enter participants"**)  
 else:  
 print(participants)  
 participants = participants.split(**", "**)  
 self.frame.graphics\_q.put((**"new evt parti"**, (participants)))  
  
 def handle\_back(self, event):  
 """  
 go back to event panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.event.Show()  
  
  
  
# newEventPanel - responsible for making new event  
class newEventPanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame, date=**"test date"**):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 self.date = date  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 titleBox = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=**"New event on "**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 title\_date = wx.StaticText(self, -1, label=date)  
 title\_date.SetForegroundColour(wx.BLACK)  
 title\_date.SetFont(titlefont)  
 titleBox.Add(title)  
 titleBox.Add(title\_date)  
  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # name  
 nameBox = wx.BoxSizer(wx.HORIZONTAL)  
 nameText = wx.StaticText(self, 1, label=**"Name of event: "**)  
 nameText.SetFont(smallfont)  
 self.nameField = wx.TextCtrl(self, -1, name=**"name"**, size=(300, 50))  
 self.nameField.SetFont(smallfont)  
 nameBox.Add(nameText, 0, wx.ALL, 5)  
 nameBox.Add(self.nameField, 0, wx.ALL, 5)  
 # participants  
 partiBox = wx.BoxSizer(wx.HORIZONTAL)  
 partiText = wx.StaticText(self, 1, label=**"Participants to add: "**)  
 partiText.SetFont(smallfont)  
 self.partiField = wx.TextCtrl(self, -1, name=**"participants"**, size=(300, 50))  
 self.partiField.SetFont(smallfont)  
 partiBox.Add(partiText, 0, wx.ALL, 5)  
 partiBox.Add(self.partiField, 0, wx.ALL, 5)  
 # start hour  
 startBox = wx.BoxSizer(wx.HORIZONTAL)  
 startText = wx.StaticText(self, 1, label=**"Start hour: "**)  
 startText.SetFont(smallfont)  
 self.startField = wx.TextCtrl(self, -1, name=**"start"**, size=(300, 50))  
 self.startField.SetFont(smallfont)  
 startBox.Add(startText, 0, wx.ALL, 5)  
 startBox.Add(self.startField, 0, wx.ALL, 5)  
 # end hour  
 endBox = wx.BoxSizer(wx.HORIZONTAL)  
 endText = wx.StaticText(self, 1, label=**"End hour: "**)  
 endText.SetFont(smallfont)  
 self.endField = wx.TextCtrl(self, -1, name=**"end"**, size=(300, 50))  
 self.endField.SetFont(smallfont)  
 endBox.Add(endText, 0, wx.ALL, 5)  
 endBox.Add(self.endField, 0, wx.ALL, 5)  
  
 newE = wx.Image(**"pics**\\**done.png"**)  
 newE.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newE = wx.Bitmap(newE)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newE)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_event)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(40)  
 sizer.Add(titleBox, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(50)  
 sizer.Add(nameBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(10)  
 sizer.Add(partiBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(10)  
 sizer.Add(startBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(10)  
 sizer.Add(endBox, -1, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(30)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
 self.frame.SetStatusText(**"participants names must be separated with ', '"**)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_event(self, event):  
 """  
 try to add event  
 :param event:  
 :return:  
 """  
 name = self.nameField.GetValue()  
 participants = self.partiField.GetValue()  
 start = self.startField.GetValue()  
 end = self.endField.GetValue()  
 if not name or not start or not end:  
 self.frame.SetStatusText(**"Must enter name, start and end"**)  
 else:  
 if not participants:  
 participants = **""** participants = participants.split(**", "**)  
 self.frame.graphics\_q.put((**"new event"**, (name, participants, start, end, self.date)))  
  
 def handle\_back(self, event):  
 """  
 go back to event panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.event.Show()  
  
  
# calNamePanel - responsible for changing the name of calendar  
class calNamePanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 titleBox = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=**"Change calendar name "**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 titleBox.Add(title)  
  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # name  
 nameBox = wx.BoxSizer(wx.HORIZONTAL)  
 nameText = wx.StaticText(self, 1, label=**"New name: "**)  
 nameText.SetFont(smallfont)  
 self.nameField = wx.TextCtrl(self, -1, name=**"name"**, size=(300, 50))  
 self.nameField.SetFont(smallfont)  
 nameBox.Add(nameText, 0, wx.ALL, 5)  
 nameBox.Add(self.nameField, 0, wx.ALL, 5)  
  
 newN = wx.Image(**"pics**\\**done.png"**)  
 newN.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newN = wx.Bitmap(newN)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newN)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_name)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(60)  
 sizer.Add(titleBox, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(60)  
 sizer.Add(nameBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(40)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_name(self, event):  
 """  
 try to change calendar name  
 :param event:  
 :return:  
 """  
 name = self.nameField.GetValue()  
  
 if not name:  
 self.frame.SetStatusText(**"Must enter name"**)  
 else:  
 self.frame.graphics\_q.put((**"cal name"**, (name)))  
  
 def handle\_back(self, event):  
 """  
 go back to calendar panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.calendar.Show()  
  
  
  
# evtNamePanel - responsible for changing the name of event  
class evtNamePanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 titleBox = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=**"Change event name "**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 titleBox.Add(title)  
  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # name  
 nameBox = wx.BoxSizer(wx.HORIZONTAL)  
 nameText = wx.StaticText(self, 1, label=**"New name: "**)  
 nameText.SetFont(smallfont)  
 self.nameField = wx.TextCtrl(self, -1, name=**"name"**, size=(300, 50))  
 self.nameField.SetFont(smallfont)  
 nameBox.Add(nameText, 0, wx.ALL, 5)  
 nameBox.Add(self.nameField, 0, wx.ALL, 5)  
  
 newN = wx.Image(**"pics**\\**done.png"**)  
 newN.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newN = wx.Bitmap(newN)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newN)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_name)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(60)  
 sizer.Add(titleBox, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(60)  
 sizer.Add(nameBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(40)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_name(self, event):  
 """  
 try to change event name  
 :param event:  
 :return:  
 """  
 name = self.nameField.GetValue()  
  
 if not name:  
 self.frame.SetStatusText(**"Must enter name"**)  
 else:  
 self.frame.graphics\_q.put((**"evt name"**, (name)))  
  
 def handle\_back(self, event):  
 """  
 go back to event panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.event.Show()  
  
  
  
# evtTimePanel - responsible for changing the time of event  
class evtTimePanel(wx.Panel):  
 def \_\_init\_\_(self, parent, frame):  
 wx.Panel.\_\_init\_\_(self, parent, pos=wx.DefaultPosition, size=frame.GetSize())  
 self.frame = frame  
 self.parent = parent  
 self.SetBackgroundColour(wx.LIGHT\_GREY)  
 sizer = wx.BoxSizer(wx.VERTICAL)  
 # title  
 titleBox = wx.BoxSizer(wx.HORIZONTAL)  
 title = wx.StaticText(self, -1, label=**"Change event time "**)  
 titlefont = wx.Font(40, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
 title.SetForegroundColour(wx.BLACK)  
 title.SetFont(titlefont)  
 titleBox.Add(title)  
  
 smallfont = wx.Font(20, wx.DECORATIVE, wx.NORMAL, wx.NORMAL)  
  
 # date  
 dateBox = wx.BoxSizer(wx.HORIZONTAL)  
 dateText = wx.StaticText(self, 1, label=**"New date: "**)  
 dateText.SetFont(smallfont)  
 self.dateField = wx.TextCtrl(self, -1, name=**"date"**, size=(300, 50))  
 self.dateField.SetFont(smallfont)  
 dateBox.Add(dateText, 0, wx.ALL, 5)  
 dateBox.Add(self.dateField, 0, wx.ALL, 5)  
  
 # start  
 startBox = wx.BoxSizer(wx.HORIZONTAL)  
 startText = wx.StaticText(self, 1, label=**"New start hour: "**)  
 startText.SetFont(smallfont)  
 self.startField = wx.TextCtrl(self, -1, name=**"date"**, size=(300, 50))  
 self.startField.SetFont(smallfont)  
 startBox.Add(startText, 0, wx.ALL, 5)  
 startBox.Add(self.startField, 0, wx.ALL, 5)  
  
 # end  
 endBox = wx.BoxSizer(wx.HORIZONTAL)  
 endText = wx.StaticText(self, 1, label=**"New end hour: "**)  
 endText.SetFont(smallfont)  
 self.endField = wx.TextCtrl(self, -1, name=**"date"**, size=(300, 50))  
 self.endField.SetFont(smallfont)  
 endBox.Add(endText, 0, wx.ALL, 5)  
 endBox.Add(self.endField, 0, wx.ALL, 5)  
  
 newT = wx.Image(**"pics**\\**done.png"**)  
 newT.Rescale(250, 100)  
 back = wx.Image(**"pics**\\**back.png"**)  
 back.Rescale(200, 80)  
 back = wx.Bitmap(back)  
 newT = wx.Bitmap(newT)  
 newBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=newT)  
 newBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 newBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
 backBtn = wx.BitmapButton(self,  
 size=(200, 80), name=**"button"**, bitmap=back)  
 backBtn.SetBackgroundColour(wx.LIGHT\_GREY)  
 backBtn.SetWindowStyleFlag(wx.NO\_BORDER)  
  
 btnBox = wx.BoxSizer(wx.HORIZONTAL)  
 backBtn.Bind(wx.EVT\_BUTTON, self.handle\_back)  
 btnBox.Add(backBtn, 1, wx.ALL, 5)  
 btnBox.AddSpacer(40)  
 newBtn.Bind(wx.EVT\_BUTTON, self.handle\_new\_time)  
 btnBox.Add(newBtn, 0, wx.ALL, 5)  
  
 # add all elements to sizer  
 sizer.AddSpacer(40)  
 sizer.Add(titleBox, 0, wx.CENTER | wx.TOP, 5)  
 sizer.AddSpacer(40)  
 sizer.Add(dateBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(20)  
 sizer.Add(startBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(20)  
 sizer.Add(endBox, 0, wx.CENTER | wx.ALL, 5)  
 sizer.AddSpacer(40)  
 sizer.Add(btnBox, wx.CENTER | wx.ALL, 5)  
  
 # arrange the screen  
 self.SetSizer(sizer)  
 self.Layout()  
 self.Hide()  
  
 def handle\_new\_time(self, event):  
 """  
 try to change time of event  
 :param event:  
 :return:  
 """  
 date = self.dateField.GetValue()  
 start = self.startField.GetValue()  
 end = self.endField.GetValue()  
  
  
 if not date or not start or not end:  
 self.frame.SetStatusText(**"Must enter date, start hour and end hour"**)  
 else:  
 self.frame.graphics\_q.put((**"evt time"**, (date, start, end)))  
  
 def handle\_back(self, event):  
 """  
 go back to event panel  
 :param event:  
 :return:  
 """  
 self.frame.SetStatusText(**""**)  
 self.Hide()  
 self.parent.event.Show()  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = wx.App(False)  
 frame = MyFrame()  
 frame.Show()  
 app.MainLoop()