פרויקט גמר

**הנדסאי שם המגמה**

בנושא

תפריט חכם

מגישים: ג'וליאן תותרי

מנחה: רון להב

2025

תארי ך: 18.01.2025

חוזר מנהל מה"ט – 51-4-11 נספח מס( 1 'הצעה לפרויקט גמר)

**א. פרטי הסטודנט**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **תאריך סיום**  **הלימודים** | **טלפון נייד** | **כתובת** | **ת.ז. 9 ספרות** | **שם הסטודנט** |
|  | 0504575435 | 6 רח' 5056 | 214659385 | ג'וליאן תותרי |

**שם המכללה** המכללה הטכנולוגית נוף הגליל  **סמל המכללה :** 72209

**מסלול ההכשרה:** הנדסאים.

**מגמת לימוד:** הנדסת תוכנה-סייבר

**מקום ביצוע הפרויקט:** המכללה הטכנולוגית נוף הגליל

**ב. פרטי המנחה האישי**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **מקום עבודה/תפקיד** | **תואר** | **טלפון נייד** | **כתובת** | **שם המנחה \*** |
| ראש המגמה | Bsc / Mba | 0523203282 | יקינטון 23 חיפה | רון להב |



\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

חתימת הסטודנט חתימת המנחה האישי חתימת הגורם המקצועי מטעם מה"ט

# שם הפרויקט

תפריט חכם (smart menu). "ביקור חכם במסעדה ללא המתנה למלצר"

# רקע

## **תיאור ורקע כללי**

מערכת לניהול עבודת המלצרים והמטבח מול הלקוח במסעדה.

המערכת מציגה תפריט של מסעדה ומאפשרת ללקוח להזמין אוכל\שתיה מהתפריט, ומאפשרת גם לבקש מהמלצר שירות לפי הצורך למשל "לבקש חשבון", ובנוסף לזה תאפשר ללקוח לשלם את החשבון דרך כרטיס אשראי. הרעיון הוא שהכול מתנהל דרך האפליקציה בלי להמתין שהמלצר יגיע לשולחן על מנת לקבל את בקשות הלקוח

## **מטרות המערכת**

**1**. מטרת המערכת לשפר את עבודת המלצר במסעדה ולקצר את זמן ההמתנה למלצר. בכך מזרז את ביצועו שגורם להקטנת זמני המתנה של הלקוחות במסעדה.

**2**. המטרה היא להפוך את תפריט המסעדה לתפריט דיגיטלי, אבל לא רק אלא גם רוב פעולות המסעדה לפעולות דיגיטלית וטכנולוגית יותר מתקדמות במקום פעולות ידניות, בעיקר פעולות המלצר.

# סקירת מצב קיים בשוק, אלו בעיות קימות

* [Must Have Menu](https://www.musthavemenus.com/category/Restaurant-Menu.html)
* [*Web Restaurant*](https://www.webstaurantstore.com/guide/813/menu-buying-guide.html)- היום קיים מגוון רחב של מערכות כמו *(MSUT HAVE MENU && WEB RESTURANT)* השימושיות בתוך המסעדות מערכות אלו לא ניתן לבציע הזמנה של אוכל\שתיה דרכן אלא צריך לבצע מול המלצר, ובנוסף לזה לא ניתן לסגור את החשבון ולבצע תשלום דרך כרטיס אשראי.
* [Menu](https://www.webstaurantstore.com/guide/813/menu-buying-guide.html) - מערכת "Menu" לא מאפשרת ללקוחות להזמין אוכל דרכה אלא רק לצפות בתפריט של המסעדה.
* נכון להיום קיים בעיה נפוצה ברוב המסעדות היא איטיות השירות של המסעדה שזה בגלל מערכת ניהולית לא יעילה, גורם ללקוחות לצאת מהמסעדה ללא חזרה.
* רוב המסעדות עדיין עובדים עם תפריט נייר שקשה לבעל המסעדה להוסיף או לשנות פריט בתוכו.

# מה הפרויקט אמור לחדש או לשפר

הפרויקט משנה את סגנון עבודת המסעדות לסגנון דיגיטלי וטכנולוגי מתקדם יותר והפרויקט מאפשר למסעדות להיות מעודכנים תמיד, לטובת המסעדנים והלקוחות. כמו כן הפרויקט משפר את עבודת ה online והתאום בין הגורמים השונים שבמסעדה. המערכת מאפשרת למצע צ'ט בין הסועד לעובד ייעודי במסעדה לצורך התייעצות על מנות וסיוע בבחירת המנות.

# דרישות מערכת ופונקציונאליות

## **דרישות מערכת** – סביבת הטמעה ושימוש. שרידות, ביצועים\ התמודדות עם עומסים.

* המערכת תתנהל בצורה מקוונת, כלומר מתחברים לאתר האינטרנט של המערכת לכן חובה להיות מחובר לאינטרנט.
* הנתונים שנשמרים בענן המערכת עושה להם גיבוי בזמנים קבועים כך ניתן לשמור על הנתונים מהאבוד.
* המערכת תופעל בענן בעל כוח רב לארח מספר ענק של משתמשים בו זמני, בנוסף מאגרי הנתונים יתארחו על גבי ענן בעל קיבולת זיכרון גבוהה וגמישה.

## **דרישות פונקציונאליות – רשימת דרישות המשתמש מהמערכת, מהן הפעולות בהן נדרשת המערכת לתמוך.**

|  |  |  |  |
| --- | --- | --- | --- |
| **מס** | **תיאור דרישה** | **FR** | **NFR** |
| **1** | **משתמשים** |  |  |
| 1.1 | המערכת תאפשר קליטת משתמש חדש באמצעות ממשק טלפוני . | V |  |
| 1.1.1 | בזמן קליטת משתמש חדש  המערכת תקלוט הפרטים הבאים  1) שם פרטי  2) מספר טלפון  3) מייל  4) סיסמה לחשבון הלקוח | V |  |
| 1.1.2 | המערכת תאפשר ללקוח קיים לראות ולערוך את החשבון שלו במערכת. | V |  |
| 1.2 | המערכת תצפין את סיסמת המשתמש דרך שימוש באלגוריתם Hashing של BCrypt | V |  |
| 1.3 | המערכת תאמת בקשות משתמש לשרת דרך   JWT (JSON Web Token | V |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **מערכת** |  |  |
| 2.1 | לאחר כניסה למערכת, המערכת תחשוף בפני המשתמש את שולחנות המסעדה והסטטוס שלהם  **(Occupied\Empty)** | V |  |
| 2.2 | המערכת תאפשר למשתמש לסמן את עצמו כיושב על שולחן דרך כפתור באפליקציה | V |  |
| 2.3 | במידה ונמצא שולחן פנוי עבור הסועד / סועדים ולחצן "סמן כיושב" המערכת תעביר את המשתמש לתפריט המסעדה לשם בקשת האוכל | V |  |
| 2.4 | אחרי הזמנת ארוחות מהתפריט המשתמש יועבר לעמוד תשלום | V |  |
| 2.5 | המערכת תסמן שולחן כפנוי לאחר גמר תשלום ואישור המשלם שהוא פינה את השולחן | V |  |
| 2.6 | בתום ביצוע ההזמנה , המלצרים והשף של המסעדה יכולים לראות את ההזמנה והיא תצא לביצוע במטבח. | V |  |
| 2.7 | המערכת תאפשר למטבח לסמן את ההזמנה כמוכנה לשילוח לשולחן בתום ההכנה שלה. | V |  |
| 2.8 | המערכת תשתמש בשקעים (Sockets) לשם עדכון סטטוס הכנת הארוחה בזמן אמת כשמלצר או שף מעדכן שהארוחה מוכנה היא תעודכן בזמן אמת דרך פונקציה בצד הלקוח שמאזינה לתרחיש | V |  |
| 2.9 | המערכת תשתמש בשקעים (Sockets) לשם עדכון סטטוס ישיבה על שולחן בזמן אמת למשל אם לקוח עוזב את השולחן שהוא עליו השולחן שלו יסומן כ"פנוי" לשאר הלקוחות בזמן אמת דרך תרחיש אשר נמצא באפליקציה של כל לקוח | V |  |
| 2.10 | המערכת תאפשר לבטל הזמנה כל עוד ההזמנה לא הוגשה לשולחן | V |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **3** | **מלצר** |  |  |
| 3.1 | המערכת תאפשר לבצע קריאה מהירה למלצר בעזרת פונקציות מוגדרות מראש   1. להחליף סכו"ם – להביא סכו"ם חדש. 2. להביא קנקן מים. 3. להביא כיסא תינוק 4. לנקות את השולחן. 5. להביא מפיות נוספות. 6. הבאת רוטב נוסף. (הקלדת שם הרוטב) 7. הבאת תבלינים (הקלדת שם התבלין). 8. להחליף מנה – במקרה של טעות במנה או אי-שביעות רצון. 9. להביא חשבון – אם התשלום מתבצע מול המלצר. 10. להזיז מאוורר/מזגן – אם ניתן לשליטה ידנית. 11. **טעות בחשבון** – אפשר להפנות את הלקוח לשירות תמיכה דרך האפליקציה. 12. **לדווח על אלרגיות** – ניתן להציג מידע תזונתי על המנות ישירות באפליקציה. 13. **להזמין אריזת טייק-אווי** – אפשר לאפשר ללקוח להזמין אריזה ישירות באפליקציה ולגשת לעמדת האיסוף. 14. **להזמין נר/עוגה לאירוע** – ניתן להוסיף אופציה לחגיגות וליידע את הצוות ישירות 15. **לבקש להעביר שולחן** – אם יש שולחן פנוי, ניתן לבצע החלפה דרך המערכת ללא צורך במלצר. 16. **לבקש חשבון נפרד** – המערכת תאפשר חלוקת תשלום דיגיטלית לכל סועד בנפרד | V |  |
| 4 | **בסיס נתונים** |  |  |
| 4.1 | במערכת קיים בסיס נתונים השומר את נתונים הבאים:   1. נתונים של המשתמשים. 2. נתונים של המסעדה. 3. הזמנות קודמות של משתמשים. 4. פרטי שולחנות המסעדה (כמה אנשים יכולים לשבת וכו..) |  | V |
| 4.2 | פרטי משתמשים רגישים כמו סיסמאות, מספרי כרטיסי אשראי, מס' ת"ז של בעלי כרטיסי יוצפנו ע"י ספריית הצפנה | V |  |
| 5 | **בעל מסעדה** |  |  |
| 5.1 | לבעל מסעדה יהיו יכולות הוספה ומחיקה של עובדים, שולחנות, ומנות | V |  |
| 5.2 | בעל מסעדה יכול להוסיף עובד כל עוד שאותו עובד לא נמצא בבסיס הנתונים (המייל של העובד צריך להיות ייחודי) | V |  |
| 5.3 | בעל מסעדה יכול למחוק עובד כל עוד אותו עובד אינו מחובר לטרמינל שלא (כלומר כל עוד שהעובד אינו במשמרת) | V |  |
| 5.4 | כשמנהל מוחק שולחן מספר השולחנות משתני לשקף את כמות השולחנות האמיתי שנמצאים במסעדה. הסבר:  נניח שהמסעדה כוללת 5 שולחנות ממוספרות מ 1-5 אם בעל בא למחוק את שולחן 2 שולחן 2 נמחק שולחן 3 הופך לשולחן 2 שולחן 4 ל-3 וכו... | V |  |
| 5.5 | בשל הפונקציה בסעיף 5.4 ולשם מחיקת שולחן בצורה מוגנת כל השולחנות אחרי אותו שולחן צריכים להיות רכים כך שאין בהם אף לקוח ואף מלצר אחרת המחיקה תצור התנהגות לא צפויה שח התוכנה | V |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | תפריט מוצר |  |  |
| 6.1 | מוצרים במסעדה יחולקו לפי קטגוריות 1. מנות ראשונות  2. מנות עיקריות  3. קינוחים.  4. משקאות קרים\חמים  5. אלכוהול |  | V |
| 6.2 | כל מוצר יכלול   1. את מרכיבי המנה 2. מחיר 3. קטיגורית מנה 4. המלצות כלליות |  | V |

# בעיות צפויות במהלך הפיתוח ופתרונות:

|  |  |
| --- | --- |
| **תיאור הבעיה(6.1)** | **פתרון אפשרי(6.2)** |
| אינטגרציה של אוסף של APIs בעיקר Mailjet לשליחת מייל וAbstractAPI על מנת לבדוק אם כתובת מייל נמצאת | כמוס פונקציונאליות של שליחת מיילים דרך שימוש במחלקה המשלבת את פונקציונאליות שתי השירותים יחד |
| כישלון חיבור ל-MongoDB אחרי שלקוח נכנס יכול להיות בעיתי לפונקציונאליות של התוכנה | יישום caching לנתונים אשר חשובים ללקוח בעיקר הארוחות בתפריט וה-State האחרון של שולחנות המסעדה. בנוסף לכך השארת אותו State בצד שרת חשוב כי לפעמים כישלון חיבור ל-MongoDB לא יהיה תוצאה משגיאה בשרת |
| סוג החיבור בין צד הלקוח לצד השרת הוא HTTP כברירת מחדל | שימוש בתעודות TLS/SSL על מנת לשמור על הצפנת מידע שיוחלף בין שרת ללקוח |
| גישה ל-APIs דורשת מפתחות API שהתוכנה צריכה להכיל. אבל העלאת מפתחות עם הקוד ל-Github עלולה לחשוף את אותם מפתחות לשאר העולם | לשים את כלל המפתחות בקובץ JSON אחד אשר נקרא Secrets.json ואי העלאתו ל-Github דרך הוספת שמו לקובץ ה-.gitignore שלנו |

# פתרון טכנולוגי נבחר:

## **טופולוגית הפתרון:**

****

## **טכנולוגיות בשימוש**:

* Visual Studio Code תוכנה לעריכת קוד
* MongoDB Compass תוכנה לוויזווליזאציאת בסיס נתונים
* MailjetAPI ספריית עזר לשם שליחת מיילים למשתמשים

## **שפות הפיתוח:**

**צד לקוח:**

* **TypeScript**: שפת תכנות המוסיפה טיפוסיות ל-JavaScript ומאפשרת כתיבת קוד אמין ומובנה.
* **React Native**: ספריה לתכנות ממשקי משתמש לאיפון ולאנדרואיד בחרתי אותה בגלל פילוסופיאת ה-Write once, run everywhere אם לא בחרתי אותה אז היה נדרש ממני לכתוב את ה-UI לכל סוג פלפון בשפה לחוד
* **CSS**: לעיצוב ועיצוב דפי אינטרנט.

**צד שרת:**

* **C#**: שפת תכנות מונחית עצמים המשמשת לפיתוח צד שרת. השתמשתי בשפה זו בשל מהירותה
* **MongoDB**: מערכת לניהול מסדי נתונים גמישה ומודולרית, שמתאימה לעבודה עם כמויות גדולות של נתונים.

## **תיאור הארכיטקטורה הנבחרת:**

בחרתי בארכיטקטורת שרת-לקוח עם שרת מקומי לכל מסעדה המשתמשת באפליקציה בכדי לצבור על מחירי ה-Hosting ולהתמודד עם היכולות הפיננסיות של בעלי המסעדות. בחרתי העיקר ב-#C לשרת בשל מהירות השפה ובחרתי ב-React Native כספריה על גבי TypeScript לשם גמישותה עם סוגי מכשירים טלפונים שונים ויכולותיה הרחבות לתכנת UI לאותם מכשירים

## **חלוקה לרכיבים ומודולים:**

* **מחלקת Socket Service:** מחלקה המספקת תרחישים לניהול העברת מידע בין משתמשים ב-Session כגון אם משתמש ישב על שולחן התרחיש יסמן את השולחן כ-"לו פנוי" אצל שאר המשתמשים. השירות הזה גם מספק פונקציונאליות למלצר לארח שולחן לבדוק דרישות משתמש וכו...
* **מחלקת Owner Controller:** מחלקה שמנהלת את פעולות בעל המסעדה אשר לא צריכות חיבור ל-Session כגון מחיקת ארוחות מהתפריט, הוספת ארוחות ופעולות CRUD נוספות על מלצרים ושולחנות
* **מחלקת Security Manager:** מחלקה עם אחריות על הנפקת TOKENS לכל משתמש בנוסף להצפנת נתונים רגישים לפני שמירתן במסד הנתונים כמו סיסמאות.

## **סביבת השרת:**

בחרתי בארכיטקטורת שרת-לקוח עם שרת מקומי לכל מסעדה המשתמשת באפליקציה בכדי לצבור על מחירי ה-Hosting ולהתמודד עם היכולות הפיננסיות של בעלי המסעדות.

## **ממשק המשתמש/לקוח – :GUI**

**עמוד בית (משתמש)**



**תמונה שמכילה טקסט, צילום מסך, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.Table taken** **Table not taken.**



**תפריט אוכל (משתמש)**

**A screenshot of a phone screen

AI-generated content may be incorrect.**

**עמוד בית (מנהל)**



**תפריט משתמש – הזמנות שרות מהיר קריאה למלצר**

**שליחת הודעה מהירה למלצר (צד לקוח)**



**בקשת סועד (עמוד מלצר)**  **ממשק מלצר – בקשה של סועד.**

בקשה מהסועד להוסיף להזמנה פריט מהתפריט.

## **ממשקים למערכות אחרות : API/**

הפרויקט משתמש ב-API של **RSA** לצורך תקשורת בין השרת ללקוח ולהבאת נתונים של ערכים מתוזמנים, וכן לצורכי אבטחה ובקרה על הגישה למידע רגיש.

## **שימוש בחבילות תוכנה:**

**React**: ספרייה ליצירת ממשק משתמש אינטראקטיבי עם רכיבים.

**Axios**:ספרייה לביצוע בקשות HTTP ולניהול התשובות מהשרת.

**Mongoose** : ספרייה ליצירת מודלים ולניהול הקשר עם מסדי נתונים ב-MongoDB ב-Node.js.

**Express.js** : פריימוורק ליצירת אפליקציות רשת ושרתים ב-Node.js.

**CORS** : מנגנון אבטחה לניהול בקשות ממקורות שונים ולמניעת בעיות גישה בין דומיינים שונים.

# שימוש במבני נתונים וארגון קבצים :

## **ERD**



# תרשימי מערכת מרכזיים:

## **Use Case**



## **Sequence Diagram:**

## **Data Flow**

#### **Context Diagram**



#### **DFD0**



#### **DFD1 for P1 – Find Table**



#### **DFD1 for P5 – Waiter**



#### **מילונים**

#### **מילון ישויות:-**

|  |  |  |
| --- | --- | --- |
| **קוד** | **שם** | **תיאור** |
| E1 | User | ישות המתארת משתמש שסועד במסעדה בעלת האפליקציה |
| E2 | Owner | ישות בעל מסעדה |

#### **מילון תהליכים:-**

|  |  |  |
| --- | --- | --- |
| **קוד** | **שם** | **תיאור** |
| P1 | Find Table | תהליך אשר מנהל הבאת השולחנים מה-Session storage או ממסדי הניתונים במקרה והסועד הוא הסועד הראשון שהתחבר |
| P 2,6 | Meal handler | תהליך המאפשר פעולות CRUD על ארוחות יחד עם אפשרות ללקוח להזמין ארוחות |
| P3 | Special Request | תהליך אשר נותן ללקוח לשלוח הודעה למלצר בדבר אשר הוא דורש כגון "חשבון" |
| P4 | Payment | תהליך שמחזיר תוצאת תשלום על ארוחה "שולם בהצלחה\סירוב" אם בעל הכרטיס שילם באשראי |
| P5 | Waiter Handler | פעולה המאפשרת לבעל מסעדה לבצע פעולות CRUD על מלצרים |
| P5.1 | Add Waiter | תהליך שמתאר פונקציות של בעל מסעדה |
| P5.2 | Remove Waiter | תהליך שמתאר פונקציות של מלצר |
| P5.3 | Get Waiters | תהליך שמתאר פונקציות של בעל מסעדה |
| P7 | Table handler | פעולות CRUD על שולחנות |

#### **מילון מאגרי מידע:-**

|  |  |  |
| --- | --- | --- |
| **קוד** | **שם** | **שדות** |
| D1 | Tables | Id tableNumber  Capacity  waiterId  userId  isOccupied  isWindowSide |
| D2 | Order | TanleNumber  Meals  Quantities  TotalPrice |
| D3 | Waiter | Id name email password date phone |

#### **מילון זרימות מידע:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **קוד** | **שם הזרימה** | **מקור** | **יעד** | **שדות זרימה** |
| F1 | העברה לעמוד בית | 1E | 1P | User\_Id |
| F2 | שליפת שולחנות | 1P | 1D | - |
| F3 | נתוני שולחנות | 1D | 1P | Table\_Id  tableNumber numOfSeats  isWindowSide  isOccupied |
| F4 | נתוני שולחנות | 1P | 1E | Table\_Id  tableNumber numOfSeats  isWindowSide  isOccupied |
| F5 | נתוני ארוחות | E1 | P2 | mealName quantity  tableNumber |
| F6 | נתוני משתמש | P2 | P2 | userId |
| F7 | נתוני ארוחות | P2 | 4P | userId |
| F8 | נתוני משתמש | P2 | E1 | mealName quantity  tableNumber |
| F9 | נתוני הודעת בקשה מיוחדת | 1E | P3 | Messege  sid |
| F10 | פרטי תשלום | 1E | 4P | userId  orderId |
| F11 | אחסון פרטי תשלום | 4P | D2 | userId  orderId |
| F12 | הודעת סיום תשלום | 4P | 1E | - |
| F13 | נתוני מלצר | E2 | P5 | Name  Email  Phone  password |
| F14 | סיום הוספה\מחיקה | P5 | E2 | userId  Name  Email  Phone  password |
| F15 | פרטי ארוחה | E2 | P6 | mealName  price |
| F16 | סיום הוספה\מחיקה | P6 | E2 | mealId  mealName  price |
| F17 | נתוני שולחן | 2E | P7 | Capacity  isWindowside |
| F18 | הודעת הוספה\מחיקה | P7 | 2E | tableNumber  capacity  isWindowside  isOccupied |
| F16.1 | חיבור לסשן מסעדה | E1 | P1 | sid |
| F17.1 | שליפת נתוני שולחנות | P1 | D1 | tableNumber  capacity  isWindowside  isOccupied |
| F18.1 | נתוני שולחנות | D1 | P1 | tableNumber isWindowSide isOccupied  capacity |
| F19 | נתוני שולחנות | P1 | E1 | tableNumber  capacity  isWindowside  isOccupied |
| F20 | נתוני מלצר | E2 | P5.1 | userId  Name  Email  Phone  password |
| F21 | הודעת סיום הוספה | P5.1 | E2 | userId  Name  Email  Phone  password |
| F22 | נתוני מלצר | E2 | P5.2 | userId |
| F23 | הודעת סיום מחיקה | P5.2 | E2 | - |
| F24 | בקשת כל פרטי מלצרים | E2 | P5.3 | TOKEN |
| F25 | נתוני מלצרים | P5.3 | E2 | userId  Name  Email  Phone  password |

# . תיאור המרכיב האלגוריתמי – חישובי



## 

## **איסוף מידע וניתוחים סטטיסטיים (אנליטיקות)**

|  |  |
| --- | --- |
| **מידע** | **מטרתו** |
| נתוני לקוח, מלצר ובעל כמו כתובת ד"אל וסיסמא | נתונים אלו חשובים לשם אימות משתמשיחם והחלפת מידע בעיקר ללקוח שמקבל קבלות במייל |
| נתוני SESSION | נתוני ה-SESSION הם מי שעוזר לנו לראות איזה שולחן הוא פנוי\ ריק הוא מי שנותן ללקוח לשבת על שולחן ולמלצר לשרת שולחן מסוים והוא גם חשוב לפעולות כמו מחיקת מלצר כי חיבור מלצר למערכת (מציאותו במשמרת) מונעת מן הבעל למחוק אותו |
| נתוני ארוחות | ארוחות הם המוצרים העיקריים אשר שירותינו מספק לשם נתינת היכולת למשתמש להזמין הארוחות חייבות להיות שמורות במסד נתונים |
| נתוני שולחנות | השולחנות במסעדה הם הדבר אשר נותן ללקוח את הגישה להזמין (כתוצאה של ישיבתו) ולמלצר את היכולת לשרת את אותו לקוח |

## 

# . תיאור/התייחסות לנושאי אבטחת מידע

הנתונים תחת סכנה בפרויקט זה בצורה עיקרית היו סיסמאות משתמשים לשם כך השתמשתי בספריית bcrypt לשם ביצוע Hashig כך שאוכל לאחסן את הסיסמה של המשתמש בצורה בלתי קריאה ב-MongoDB

שינית השתמשתי ב-JSON Web Token (JWT) על מנת להפריד בין משתמשים ברמות גישה שונות. למשל משתמש רגיל אסור לו לגשת לפונקציות של בעל אז היה ניתן לתת לכל משתמש Role שהשרת יעיין כשהוא יאמת Token מסוים

# משאבים הנדרשים לפרויקט:

## **מספר שעות המוקדש לפרויקט**

|  |  |  |
| --- | --- | --- |
| שעות | מטרה | מבצע/ים |
| 50 | חיפוש ו למידה על ספריות API ו תכניכות לתכנון הפרוייקט | תותרי ג'וליאן |
| 100 | כתיבת ו ארגון קוד לשרת C# | תותרי ג'וליאן |
| 50 | כתיבת קוד ל-client (react native) | תותרי ג'וליאן |
| 100 | בדיקה ו-debug לתכנה ו לשרת ו ל- CLIENT | תותרי ג'וליאן |

## **ציוד נדרש**

החומרה הנדרשת עבור הפרויקט :

* פלפון חכם עם מערכת הפעלה Android או IOS
* מעבד עובד ברמת intel core i3

## **תוכנות נדרשות**

Visual Studio Code

MongoDB

dotnet

Fork

Git/github

PostMan

## **ידע חדש שנדרש ללמוד לצורך ביצוע הפרויקט**

* MVC
* MongoDB
* Mailjet
* Sockets

## **ספרות ומקורות מידע**

* YouTube
* אתר Geeks4Geeks.com
* תיעוד Mailjet: https://dev.mailjet.com/content/guides

# תכנית עבודה ושלבים למימוש הפרויקט

|  |  |  |
| --- | --- | --- |
| 1. | קבלת החלטה על נושא הפרויקט והתנעה ראשונית | 12/2024 |
| 2. | כתיבת הצעה / ניתוח פרויקט וממשקים | 12/09/2025 |
| 3. | Code Review 1 | 12/09/2025 |
| 4. | ועדה פנימית - Code Review 2 | 15/10/2025 |
| 5. | מסירת ספרי פרויקט / המתנה לבחינה | 15/10/2025 |

# תכנון הבדיקות שיבוצעו

|  |  |
| --- | --- |
| תיאור הבדיקה | תגובה צפויה |
| לחיצה על כפתור שולחן | סימון שולחן כ-"לא פנוי" והעברה לדף תפריט |
| לחיצה על כפתור הוספה לארוחה בתפריט | הוספת הארוחה להזמנה |
| לחיצה על Icon בקשות למלצר | העברה לדף בקשות ממלצר |
| לחיצה על כפתור "Wait Table" בטרמינל מלצר | המלצר מסומן כמטפל בשולחן ושאר פוהקציות השולחן כגון "צפייה בהזמנה" יופיעו |
| לחיצה על כפתור "מחק מלצר" בטרמינל הבעל | הודעת הצלחת מחיקה מופיעה אם המלצר עובד אחרת הודעת כישלון |
| לחיצה על כפתור "הוספת ארוחה" ע"י בעל | הארוחה נכתבת אם היא לא נמצאת |
| כפתור "סמן הזמנה כמוכנה" נלחץ ע"י מלצר | תוצג הודעה מתאימה למלצר ותועבר התרעה ללקוח המזמין |
| לחיצה על כפתור "עזוב שולחן" ע"י משתמש | פונקציות השולחן כמו לחצן בקשות ממלצר נעלמים |
| לחיצה על כפתור "צפיה בהזמנה" ע"י מלצר | המלצר מועבר לדף ההזמנה ורואה את פרטיה |

|  |  |
| --- | --- |
|  |  |
|  |  |
| חתימת הסטודנט | חתימת המנחה האישי |

ג. **הערות ראש המגמה**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ד**. אישור ראש המגמה**

|  |  |  |
| --- | --- | --- |
| להב רון | תמונה שמכילה שרטוט, אומנות ילדים, ציור, כתב יד  תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי. |  |
|  |  |  |
| שם | חתימה | תאריך |

ה. **הערות הגורם המקצועי מטעם מה''ט**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ו**. אישור הגורם המקצועי מטעם מה''ט**

**שם:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **חתימה:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **תאריך:**\_\_\_\_\_\_\_\_\_\_\_\_

הצהרת סטודנט

שם הסטודנט: ג'וליאן תותרי ת.ז: 214659385

שם המכללה בה לומד הסטודנט: המכללה הטכנולוגית נוף הגליל.

אני הח"מ, מצהיר בזאת כי פרויקט הגמר וספר הפרויקט המצ"ב נעשו על ידי בלבד.

פרויקט הגמר נעשה על סמך הנושאים שלמדתי במכללה ובאופן עצמאי.

פרויקט הגמר וספר הפרויקט נעשו על בסיס הנחייתו של המנחה האישי.

מקורות המידע בהם השתמשתי לביצוע פרויקט הגמר מצוינים ברשימת המקורות המצוינים בספר הפרויקט .

אני מודע לאחריות שהנני מקבל על עצמי על ידי חתימתי על הצהרה זו שכל הנאמר בה אמת ורק אמת .

תמונה שמכילה טיפוגרפיה, גופן, קליגרפיה, כתב יד

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

חתימת הסטודנט: תאריך: 19/09/2025

אישור המנחה האישי

הריני מאשר שהפרויקט בוצע בהנחייתי, קראתי את ספר הפרויקט ומצאתי כי הוא מוכן לצורך הגשת הסטודנט להגנה על פרויקט גמר .

שם המנחה: רון להב חתימה:  תאריך:19/09/2025

**אישור ראש המגמה**

הריני מאשר שספר הפרויקט מוכן לצורך הגשת הסטודנט להגנה על פרויקט הגמר .

שם ראש המגמה: רון להב חתימה : A close up of a signature

AI-generated content may be incorrect. תאריך:19/09/2025

הבעת תודה

פרויקט זה היווה עבורי הזדמנות נדירה לבחון וליישם את הידע הנרחב והמגוון,

אותו רכשתי במהלך שנות לימודי במגמת הנדסת תוכנה במכללה הטכנולוגית נוף הגליל .

חשוב לי, להביע את תודתנו הרבה והערכתנו לכל אלו אשר תמכו , סייעו ועזרו לי לכל אורך הדרך.

ברצוני להודות :

למנחה ו ראש המגמה רון להב- על הנחיית הפרויקט, והעזרה המקצועית לאורך הדרך, תודה על הסבלנות והשקעת הזמן, תוך דגש על הקידום המקצועי שרכשתי דרכך.

**מ"ר שרארי פיראס-** על הדרכתו והובלת יעוץ לספריות אשר בהן אוכל להשתמש ב-React Native ועל הליווי שלי ושל שאר התלמדים בקורס ה-Full Stack.

**מ"ר שרארי איאס-** על העברתו ליעוץ כלפי בניית השרת ועל שיפור היכולת שלי ושל שאר התלמידים לפתור בעיות אלגוריתמיקה ותכנות בשעות התרגול אשר הוא העביר במכללה.

תקציר

מאז השעה בה סמארטפונים יצאו לעולם ראינו עלייה בשאיפת עסקים שונים להפוך את עצמם לעסקים דיגיטליים לו ברמה חלקית וזאת בגלל סיבה עיקרית אחת נוחות. נוחות מהירות וגישה קלה הם המוטו של המאה ה-21 בשל כך יצרתי את אפליקציית "Bite Byte" אפליקציית "Bite Byte" מובילה 3 שירותים בקופסה אחת והן כנ"ל

השירות הראשון הינו שירות ללקוחות מסעדה אשר דרכו ניתן לסמן את עצמך כיושב על שולחן, להזמין אוכל, ולתקשר עם המלצר שלך לגבי הצרכים שלך תוך כדי זמן ישיבתך במסעדה.

השירות השני הוא שירות טרמינל מלצר שנותן לעובדי המסעדה לצפות בדרישות הלקוחות כמו הזמנות והודעות מהירות שהסועד יכול לשלוח למלצר על מנת להביע את דרישותיו. דבר זה מהווה הקלה לעבודת המלצר.

השירות השלישי והאחרון הוא טרמינל בעל מסעדה. טרמינל בעל המסעדה נותן לו אוסף של פעולות כמו הוספת\הורדת עובדים, הוספת\הורדת ארוחות, והוספת\הורדת שולחנות

תוכן

[1. שם הפרויקט 3](#_Toc210150830)

[2. רקע 3](#_Toc210150831)

[**2.1.** **תיאור ורקע כללי** 3](#_Toc210150832)

[**2.2.** **מטרות המערכת** 3](#_Toc210150833)

[3. סקירת מצב קיים בשוק, אלו בעיות קימות 3](#_Toc210150834)

[4. מה הפרויקט אמור לחדש או לשפר 3](#_Toc210150835)

[5. דרישות מערכת ופונקציונאליות 4](#_Toc210150836)

[5.1. **דרישות מערכת** – סביבת הטמעה ושימוש. שרידות, ביצועים\ התמודדות עם עומסים. 4](#_Toc210150837)

[**5.2.** **דרישות פונקציונאליות – רשימת דרישות המשתמש מהמערכת, מהן הפעולות בהן נדרשת המערכת לתמוך.** 4](#_Toc210150838)

[6. בעיות צפויות במהלך הפיתוח ופתרונות: 9](#_Toc210150839)

[7. פתרון טכנולוגי נבחר: 9](#_Toc210150840)

[**7.1.** **טופולוגית הפתרון:** 9](#_Toc210150841)

[7.2. **טכנולוגיות בשימוש**: 9](#_Toc210150842)

[**7.3.** **שפות הפיתוח:** 10](#_Toc210150843)

[**7.4.** **תיאור הארכיטקטורה הנבחרת:** 10](#_Toc210150844)

[**7.5.** **חלוקה לרכיבים ומודולים:** 10](#_Toc210150845)

[**7.6.** **סביבת השרת:** 10](#_Toc210150846)

[**7.7.** **ממשק המשתמש/לקוח – :GUI** 11](#_Toc210150847)

[**7.8.** **ממשקים למערכות אחרות : API/** 15](#_Toc210150848)

[**7.9.** **שימוש בחבילות תוכנה:** 15](#_Toc210150849)

[8. שימוש במבני נתונים וארגון קבצים : 15](#_Toc210150850)

[**8.1.** **ERD** 15](#_Toc210150851)

[9. תרשימי מערכת מרכזיים: 16](#_Toc210150852)

[**9.1.** **Use Case** 16](#_Toc210150853)

[**9.2.** **Sequence Diagram:** 17](#_Toc210150854)

[**9.3.** **Data Flow** 18](#_Toc210150855)

[10. . תיאור המרכיב האלגוריתמי – חישובי 24](#_Toc210150856)

[**10.1.** **איסוף מידע וניתוחים סטטיסטיים (אנליטיקות)** 25](#_Toc210150857)

[11. . תיאור/התייחסות לנושאי אבטחת מידע 25](#_Toc210150858)

[12. משאבים הנדרשים לפרויקט: 26](#_Toc210150859)

[**12.1.** **מספר שעות המוקדש לפרויקט** 26](#_Toc210150860)

[**12.2.** **ציוד נדרש** 26](#_Toc210150861)

[**12.3.** **תוכנות נדרשות** 26](#_Toc210150862)

[**12.4.** **ידע חדש שנדרש ללמוד לצורך ביצוע הפרויקט** 26](#_Toc210150863)

[**12.5.** **ספרות ומקורות מידע** 26](#_Toc210150864)

[13. תכנית עבודה ושלבים למימוש הפרויקט 27](#_Toc210150865)

[14. תכנון הבדיקות שיבוצעו 27](#_Toc210150866)

[15. רשימת דפים 34](#_Toc210150867)

[16. מבוא 39](#_Toc210150868)

# רשימת דפים

1. דף כניסה
2. דף בחירת שולחן / דף שיחת מלצר
3. תצוגת תפריט
4. דף הודעות מהירות
5. דף תפריט
6. דף כניסה לצוות/בעלים
7. דף תפעול בעל מסעדה
8. דף הוסף מלצר
9. דף הסר מלצר
10. דף הוסף ארוחה
11. דף הסר ארוחה
12. דף הוסף הודעה מהירה
13. דף הסר הודעה מהירה
14. דף הוסף שולחן
15. דף הסר שולחן
16. דף הצג שולחן תפוס (מלצר)
17. דף הצג הזמנות אוכל (מלצר)
18. דף הצג הזמנות/הודעות מהירות
19. כניסה ראשונה

(1 דף כניסה 2) דף בחירת שולחן / דף שיחת מלצר



תמונה שמכילה טקסט, צילום מסך, תוכנה, סמל מחשב

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.3)תצוגת תפריט 4)דף הודעות מהירות 5) דף תפריט

תמונה שמכילה טקסט, צילום מסך, תוכנה, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.תמונה שמכילה טקסט, צילום מסך, תוכנה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

6) דף כניסה לצוות/בעלים 7) דף תפעול בעל מסעדה

תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

8) דף הוסף מלצר 9) דף הסר מלצר

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.תמונה שמכילה טקסט, צילום מסך, עיצוב

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

10) דף הוסף ארוחה 11) דף הסר ארוחה

תמונה שמכילה טקסט, צילום מסך, תוכנה, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.12) דף הוסף הודעה מהירה 13) דף הסר הודעה מהירה



14) דף הוסף שולחן 15) דף הסר שולחן

תמונה שמכילה טקסט, צילום מסך, גופן, תוכנה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

תמונה שמכילה טקסט, צילום מסך, עיצוב

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.16) דף הצג שולחן תפוס (מלצר) 17) דף הצג הזמנות אוכל (מלצר)

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

18) דף הצג הזמנות/הודעות מהירות (מלצר) 19)כניסה ראשונה

תמונה שמכילה טקסט, צילום מסך, תוכנה, עיצוב

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.תמונה שמכילה טקסט, צילום מסך, תוכנה, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

# מבוא

בעידן הדיגיטלי המודרני, מסעדות רבות מחפשות דרכים לייעל את עבודתן ולהעניק ללקוחות חווית שירות מהירה, נוחה ויעילה יותר. בעיה מרכזית הקיימת כיום ברוב המסעדות היא איטיות השירות וזמני המתנה ממושכים, הנובעים מתהליכי עבודה ידניים וחוסר ניצול של כלים טכנולוגיים מתקדמים.

פרויקט זה מציע מערכת **"תפריט חכם" (Smart Menu)**, אשר נועדה לנהל את הקשר בין הלקוח, המלצר והמטבח בצורה מקוונת ודיגיטלית. המערכת מציגה תפריט דיגיטלי ומאפשרת ללקוח לבצע פעולות מגוונות – החל מסימון ישיבה על שולחן, דרך הזמנת אוכל ושתייה, שליחת בקשות מהירות למלצר, ועד תשלום חשבון בכרטיס אשראי – והכול דרך האפליקציה בלבד, ללא צורך בהמתנה למלצר.

באמצעות המערכת מושגות מספר מטרות מרכזיות:

1. **קיצור זמני ההמתנה במסעדה** – הפחתת התלות במלצרים לביצוע פעולות בסיסיות והעברת השליטה לידי הלקוח.
2. **ייעול תהליכי העבודה** – הפיכת פעולות ידניות רבות לפעולות דיגיטליות, מה שמאפשר למלצרים להתמקד בשירות אישי ואיכותי יותר.
3. **שיפור חווית המשתמש** – יצירת ממשק נגיש, מהיר ואינטואיטיבי, שמבטיח ללקוח חווית שירות חדשנית.
4. **שילוב טכנולוגיות מתקדמות** – שימוש בכלים כגון Sockets לעדכון בזמן אמת, מנגנוני אבטחת מידע (כגון הצפנת סיסמאות ו־JWT), ובסיס נתונים גמיש לניהול מידע עסקי ולקוח.

ייחודיות המערכת היא בכך שהיא אינה מסתפקת בהצגת תפריט בלבד, אלא מעניקה פתרון כולל לניהול כלל הפעולות במסעדה – החל מבקשת שירות פשוטה ועד לניהול מלאי ותיאום בין הגורמים השונים. בכך היא מביאה בשורה חדשה לעולם המסעדנות, ומאפשרת לשלב בין חווית סועד מתקדמת לניהול מסעדה יעיל ורווחי יותר.

# קוד

* צד כניסה ללקוחות ולעובדים
* שפת: React Native, TypeScript
* לקוחות מזינים אימייל וסיסמה ונכנסים לפריסת הטאבים; צוות מזין אימייל וסיסמה ונכנס לטרמינל הצוות; למשתמש ללא חשבון קיימת אפשרות הרשמה.

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import React, { useState } from 'react';

import { StyleSheet, View } from 'react-native';

import { useNavigation } from '@react-navigation/native';

import Logo from "@/components/ui/Logo";

import CurvedButton from '@/components/ui/CurvedButton';

import axios from 'axios';

import ip from '@/Data/Addresses';

import AsyncStorage from '@react-native-async-storage/async-storage';

import ThemedInput from '@/components/ThemedInput';

import { NavigationProp } from '@/Routes/NavigationTypes';

/\*\*

 \* A login screen for customers and staff to log in to the system.

 \* Customers can log in with their email and password to access the tabs layout.

 \* Staff can log in with their email and password to access the staff terminal.

 \* If the user does not have an account, they can sign up.

 \*

 \* @remarks

 \* This component uses the `useNavigation` hook from `@react-navigation/native` to

 \* navigate to the tabs layout or the signup screen.

 \*

 \* @returns A JSX element representing the login screen.

 \*/

export default function Login() {

  const [email, setEmail] = useState<string>('');

  const [password, setPassword] = useState<string>('');

  const navigation = useNavigation<NavigationProp>();

  const styles = StyleSheet.create({

    headerImage: {

      color: '#808080',

      bottom: -90,

      left: -35,

      position: 'absolute',

    },

    container: {

      flex: 1,

      justifyContent: 'center',

      alignItems: 'center',

      padding: 20,

      gap: 20,

    },

    emailText: {

      fontSize: 16,

      fontWeight: 'bold',

      marginBottom: 10,

    },

    signupContainer:{

      flexDirection: 'row',

      justifyContent: 'center',

      alignItems: 'center',

      gap: 20,

    },

    staff:{

      flexDirection: 'row',

      justifyContent: 'space-around',

      alignItems: 'center',

      width: '90%',

      backgroundColor:"transparent"

    },

    text:{

      fontSize: 25,

      fontWeight: 'bold',

      height: 'auto',

      width: 'auto',

    },

  });

  /\*\*

   \* Handles the login process for customers and staff.

   \* Performs validation and sends a request to the server to

   \* authenticate the user.

   \* If the user is authenticated, sets the token and user data in

   \* AsyncStorage and navigates to the tabs layout.

   \* If the user is not authenticated, displays an error message.

   \*/

  const handleLogin = async() => {

    // Perform login validation (optional)

    if (email && password) {

      // Send login request to the server

      try{

        const res = await axios.post(`http://${ip.julian}:5256/api/user/login`,{

            email: email,

            password: password

        })

        if(res&& res.status===200){

            await AsyncStorage.setItem('token', res.headers['x-auth-token']);

            await AsyncStorage.setItem('user', JSON.stringify(res.data.user));

            const user =await AsyncStorage.getItem('user')

        }

        // Navigate to the tabs layout

        navigation.reset({

          index: 0,

          routes: [{ name: 'Tabs' }],

        });

      }

      catch(e:any){

        alert(e.message);

      }

    } else {

      alert('Please enter email and password!');

    }

  };

  return (

    <ThemedView style={styles.container}>

      <ThemedText style={styles.text}>Customer Login</ThemedText>

      <Logo />

      <ThemedInput

        type="email-address"

        value={email}

        placeholder="Email"

        action={(text)=>setEmail(text)}

        />

      <ThemedInput

        placeholder='Password'

        type="password"

        value={password}

        action={(text) => setPassword(text) }

      />

      <CurvedButton

        title="Login"

        action={handleLogin}

        style={{ backgroundColor: '#34baeb' }}

      />

      <View style={styles.signupContainer}>

        <ThemedText style={styles.emailText}>Don't have an account?</ThemedText>

        <CurvedButton

          title="Sign Up"

          action={() => navigation.navigate('Signup')}

          style={{backgroundColor:"rgb(134, 0, 175)"}}

          />

      </View>

      <View style={styles.staff}>

          <ThemedText style={styles.emailText}>Are you staff?</ThemedText>

          <CurvedButton

            title="Staff Terminal"

            action={()=>navigation.navigate('Terminals')}

            style={{backgroundColor:"rgb(0, 175, 122)"}}

            />

      </View>

    </ThemedView>

  );

}

* צד הרישום הראשון
* שפת: React Native, TypeScript

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import React, { useState } from 'react';

import { TextInput, StyleSheet, Button, TouchableOpacity,Platform  } from 'react-native';

import DateTimePicker from '@react-native-community/datetimepicker';

import { useTheme } from '@react-navigation/native';

import Logo from "@/components/ui/Logo";

import axios from 'axios';

import ip from '@/Data/Addresses'

import CurvedButton from '@/components/ui/CurvedButton';

import ThemedInput from '@/components/ThemedInput';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

/\*\*

 \* Signup component

 \*

 \* @return {JSX.Element} - Signup component

 \*/

export default function Signup() {

    const [name, setName] = useState('');

    const [email, setEmail] = useState('');

    const [password, setPassword] = useState('');

    const [dob, setDob] = useState(''); // Store as YYYY-MM-DD

    const [dobDate, setDobDate] = useState(new Date(2000, 0, 1)); // Default to Year 2000

    const [phone, setPhone] = useState('');

    const [showDatePicker, setShowDatePicker] = useState(false);

    const { colors, dark: isDark } = useTheme();

/\*\*

 \* Handles the change of the date selection from the date picker.

 \* Updates both the display date and the formatted date of birth.

 \*

 \* @param {any} event - The event triggered by the date picker.

 \* @param {Date} [selectedDate] - The date selected by the user.

 \*                                If a date is selected, the function updates the

 \*                                state with the formatted date in 'YYYY-MM-DD' format.

 \*/

    const handleDateChange = (event:any, selectedDate?:Date) => {

        if (selectedDate) {

            setDobDate(selectedDate);

            // ✅ Fix: Use toLocaleDateString() to prevent timezone issues

            const formattedDate = selectedDate.toLocaleDateString('en-CA'); // YYYY-MM-DD format

            setDob(formattedDate);

        }

        setShowDatePicker(false); // Hide picker after selecting

    };

    /\*\*

     \* Handles the sign up request by sending a post request to the backend

     \* with the user's details. If the request is successful, it shows an alert

     \* with the response message. If there is an error, it shows an alert with

     \* the error message.

     \*

     \* @function

     \* @async

     \* @returns {undefined}

     \*/

    const handleSignup = async() => {

        try{

            const res = await axios.post(`http://${ip.julian}:5256/api/user/signup`,{

                name:name,

                email: email,

                password: password,

                date: dob,

                phone: phone

            })

            if(res && res.status===200){

                ShowMessageOnPlat(res.data);

            }

        }

        catch(e:any){

            alert(e.message);

        }

    };

    const styles = StyleSheet.create({

        container: {

            flex: 1,

            justifyContent: 'center',

            alignItems: 'center',

            padding: 20,

            gap: 20,

          },

        header:{

            fontWeight: 'bold',

            fontSize: 25,

        },

        input: {

            height: 40,

            width: '95%',

            borderWidth: 0.5,

            borderRadius: 10,

            backgroundColor: 'transparent',

            paddingHorizontal: 10,

            marginBottom: 20,

            color: isDark ? "white" : "black",

            borderColor: isDark ? "white" : "black",

          },

        button: {

            marginTop: 20,

            backgroundColor: 'rgb(134, 0, 175)',

        },

        opacity:{

            width: '100%',

           justifyContent: 'center',

           alignItems: 'center',

        }

    });

    return (

        <ThemedView style={styles.container}>

            <Logo />

            <ThemedText style={styles.header}>Sign up</ThemedText>

            <ThemedInput

                type="text"

                 placeholder="Name"

                action={(text) => setName(text)}

                value={name}/>

            <ThemedInput

                placeholder="example@domain.com"

                type={"email-address"}

                action={(text) => setEmail(text)}

                value={email}

            />

            <ThemedInput

                placeholder="Password"

                type="password"

                action={(text) => setPassword(text)}

                value={password}

            />

               <TouchableOpacity style={styles.opacity} onPress={() => setShowDatePicker(true)} >

                <TextInput

                    style={styles.input}

                    placeholder="Date of Birth"

                    placeholderTextColor={isDark ? "white" : "black"}

                    value={dob}

                    editable={false} // Prevent manual typing

                />

            </TouchableOpacity>

            {/\* Show picker when `showDatePicker` is true \*/}

            {showDatePicker && (

                <DateTimePicker

                    value={dobDate} // Default date

                    mode="date"

                    display={"spinner"} // 🔥 Now "spinner" on Android

                    onChange={handleDateChange}

                    maximumDate={new Date()} // Prevent selecting future dates

                />

            )}

            <ThemedInput

              placeholder='Phone Number'

                type="phone-pad"

                action={(text) => setPhone(text)}

                value={phone}

            />

            <CurvedButton

                title="Signup"

                action={async() => await handleSignup()}

                style={ {backgroundColor: "rgb(134, 0, 175)"}}

            />

        </ThemedView>

    );

}



import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import React, { useState } from 'react';

import { TextInput, StyleSheet, Button, TouchableOpacity,Platform  } from 'react-native';

import DateTimePicker from '@react-native-community/datetimepicker';

import { useTheme } from '@react-navigation/native';

import Logo from "@/components/ui/Logo";

import axios from 'axios';

import ip from '@/Data/Addresses'

import CurvedButton from '@/components/ui/CurvedButton';

import ThemedInput from '@/components/ThemedInput';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

/\*\*

 \* Signup component

 \*

 \* @return {JSX.Element} - Signup component

 \*/

export default function Signup() {

    const [name, setName] = useState('');

    const [email, setEmail] = useState('');

    const [password, setPassword] = useState('');

    const [dob, setDob] = useState(''); // Store as YYYY-MM-DD

    const [dobDate, setDobDate] = useState(new Date(2000, 0, 1)); // Default to Year 2000

    const [phone, setPhone] = useState('');

    const [showDatePicker, setShowDatePicker] = useState(false);

    const { colors, dark: isDark } = useTheme();

/\*\*

 \* Handles the change of the date selection from the date picker.

 \* Updates both the display date and the formatted date of birth.

 \*

 \* @param {any} event - The event triggered by the date picker.

 \* @param {Date} [selectedDate] - The date selected by the user.

 \*                                If a date is selected, the function updates the

 \*                                state with the formatted date in 'YYYY-MM-DD' format.

 \*/

    const handleDateChange = (event:any, selectedDate?:Date) => {

        if (selectedDate) {

            setDobDate(selectedDate);

            // ✅ Fix: Use toLocaleDateString() to prevent timezone issues

            const formattedDate = selectedDate.toLocaleDateString('en-CA'); // YYYY-MM-DD format

            setDob(formattedDate);

        }

        setShowDatePicker(false); // Hide picker after selecting

    };

    /\*\*

     \* Handles the sign up request by sending a post request to the backend

     \* with the user's details. If the request is successful, it shows an alert

     \* with the response message. If there is an error, it shows an alert with

     \* the error message.

     \*

     \* @function

     \* @async

     \* @returns {undefined}

     \*/

    const handleSignup = async() => {

        try{

            const res = await axios.post(`http://${ip.julian}:5256/api/user/signup`,{

                name:name,

                email: email,

                password: password,

                date: dob,

                phone: phone

            })

            if(res && res.status===200){

                ShowMessageOnPlat(res.data);

            }

        }

        catch(e:any){

            alert(e.message);

        }

    };

    const styles = StyleSheet.create({

        container: {

            flex: 1,

            justifyContent: 'center',

            alignItems: 'center',

            padding: 20,

            gap: 20,

          },

        header:{

            fontWeight: 'bold',

            fontSize: 25,

        },

        input: {

            height: 40,

            width: '95%',

            borderWidth: 0.5,

            borderRadius: 10,

            backgroundColor: 'transparent',

            paddingHorizontal: 10,

            marginBottom: 20,

            color: isDark ? "white" : "black",

            borderColor: isDark ? "white" : "black",

          },

        button: {

            marginTop: 20,

            backgroundColor: 'rgb(134, 0, 175)',

        },

        opacity:{

            width: '100%',

           justifyContent: 'center',

           alignItems: 'center',

        }

    });

    return (

        <ThemedView style={styles.container}>

            <Logo />

            <ThemedText style={styles.header}>Sign up</ThemedText>

            <ThemedInput

                type="text"

                 placeholder="Name"

                action={(text) => setName(text)}

                value={name}/>

            <ThemedInput

                placeholder="example@domain.com"

                type={"email-address"}

                action={(text) => setEmail(text)}

                value={email}

            />

            <ThemedInput

                placeholder="Password"

                type="password"

                action={(text) => setPassword(text)}

                value={password}

            />

               <TouchableOpacity style={styles.opacity} onPress={() => setShowDatePicker(true)} >

                <TextInput

                    style={styles.input}

                    placeholder="Date of Birth"

                    placeholderTextColor={isDark ? "white" : "black"}

                    value={dob}

                    editable={false} // Prevent manual typing

                />

            </TouchableOpacity>

            {/\* Show picker when `showDatePicker` is true \*/}

            {showDatePicker && (

                <DateTimePicker

                    value={dobDate} // Default date

                    mode="date"

                    display={"spinner"} // 🔥 Now "spinner" on Android

                    onChange={handleDateChange}

                    maximumDate={new Date()} // Prevent selecting future dates

                />

            )}

            <ThemedInput

              placeholder='Phone Number'

                type="phone-pad"

                action={(text) => setPhone(text)}

                value={phone}

            />

            <CurvedButton

                title="Signup"

                action={async() => await handleSignup()}

                style={ {backgroundColor: "rgb(134, 0, 175)"}}

            />

        </ThemedView>

    );

}

* עמוד ניווט לטרמינלי עובדים\בעלים
* שפה: React Native + Typescript
* עמוד זה נותן את היכולת לעובד לנווט לעמודי הכניסה של הטרמינל שלו

import React from "react";

import { ThemedView } from "@/components/ThemedView";

import { ThemedText } from "@/components/ThemedText";

import { StyleSheet, Image, TouchableOpacity } from "react-native";

import { NavigationProp } from "@/Routes/NavigationTypes";

import { useNavigation } from "@react-navigation/native";

/\*\*

 \* This component is the main screen of the staff app. It displays a list of buttons

 \* for different types of staff, each of which navigates to a different login screen.

 \*/

function Main() {

    const navigation = useNavigation<NavigationProp>();

    return (

        <ThemedView style={styles.container}>

            <ThemedText style={styles.header}>Terminals</ThemedText>

            <TouchableOpacity onPress={() =>navigation.navigate("WaiterLogin")} style={styles.staffBox}>

                <Image source={require("@/assets/images/chef.png")} style={styles.image} />

                <ThemedText style={styles.text}>Chef/Waiter</ThemedText>

            </TouchableOpacity>

            <TouchableOpacity onPress={() => navigation.navigate("OwnerLogin")} style={styles.staffBox}>

                <Image style={styles.image} source={require("@/assets/images/owner.png")} />

                <ThemedText style={styles.text}>Owner</ThemedText>

            </TouchableOpacity>

        </ThemedView>

    );

}

const styles = StyleSheet.create({

    container:{

        flex:1,

        height:'auto',

        width:'auto',

        display:'flex',

        flexDirection:'column',

        alignItems:'center',

        justifyContent:'center',

        padding:10,

        paddingHorizontal:50,

        gap:20

    },

    header:{

        fontSize:24,

        fontWeight:'bold'

    },

    staffBox:{

        display:'flex',

        flexDirection:'column',

        alignItems:'center',

        justifyContent:'center',

        width:200,

        height:200,

        margin:10,

        borderColor:"grey",

        borderWidth:1,

        borderRadius:5,

        padding:10,

        gap:5,

    },

    image: {

        width: 100,

        height: 100,

        borderRadius: 10,

        resizeMode: 'cover',

      },

      text:{

        fontSize:15,

        fontWeight:'bold'

      },

});

export default Main;

* עמוד הוספת ארוחה
* שפה: React Native + Typescript
* עמוד שנותן לבעל את היכולת להוסיף ארוחה לתפריט הקובץ מכיל קוד לממשק משתמש בנוסף לפונקציה אשר מתקשרת עם נקודת הקצה של השרת אשר מטרתה היא לכתוב את נתוני הארוחה

import { useState } from "react";

import ip from "../../../Data/Addresses";

import { ThemedText } from "@/components/ThemedText";

import { ThemedView } from "@/components/ThemedView";

import ThemedInput from "@/components/ThemedInput";

import CurvedButton from "@/components/ui/CurvedButton";

import { StyleSheet} from "react-native";

import Logo from "@/components/ui/Logo";

import axios from "axios";

import AsyncStorage from "@react-native-async-storage/async-storage";

import DropDownPicker from "react-native-dropdown-picker";

import { useTheme } from "@react-navigation/native";

import ShowMessageOnPlat from "@/components/ui/ShowMessageOnPlat";

export const [items, setItems] = useState([

        { label: 'Desserts', value: 'Desserts' },

        { label: 'Main Dish', value: 'Main Dish' },

        { label: 'Side Dish', value: 'Side Dish' },

        { label: 'Beverages', value: 'Beverages' },

        // change the all the drinks to beverages

    ]);

/\*\*

 \* Component for adding a meal to the database and AsyncStorage.

 \* Provides a form with three fields: name, price, and category.

 \* The category is a dropdown menu with the options from the database.

 \* The user can add a meal by pressing the "Add meal" button.

 \* If the meal is added successfully, the user is alerted with a success message.

 \* If there is an error, the user is alerted with the error message.

 \*/

export default function AddMealForm() {

    const { colors } = useTheme();

    const [name, setName] = useState<string>('');

    const [price, setPrice] = useState<number>(0);

    const [open, setOpen] = useState(false);

    const [category, setCategory] = useState<string | null>(null);

    /\*\*

     \* Handles the addition of a meal to the database and AsyncStorage.

     \*

     \* Before making the API call, the function checks if the meal name and price are valid.

     \* If the meal name is empty or the price is 0, the user is alerted with an error message.

     \* If the meal is added successfully, the user is alerted with a success message.

     \* If there is an error, the user is alerted with the error message.

     \*/

    const handleAddMeal = async () => {

        if(name==='' || price===0){

            alert("Can't add a no named OR free meal");

            return;

        }

        if(price<0)

        {

            alert("Price can't be negative");

            return;

        }

        try{

            const token = await AsyncStorage.getItem('token');

            const res = await axios.post(`http://${ip.julian}:5256/api/owner/add/meal`,{

                mealName:name,

                price:price,

                category:category

            },

            {

                headers:{

                    'x-auth-token':token

                }

            })

            ShowMessageOnPlat(`${name} Added Successfully!\nPrice: ${price}`);

        }catch(e){

            ShowMessageOnPlat(`${e}`);

        }

    }

    return (

        <ThemedView style={styles.container}>

            <ThemedText style={styles.text}>Add a meal</ThemedText>

            <Logo/>

            <ThemedInput

                placeholder="Name"

                type="default"

                action={(text:string)=>setName(text)}

                value={name}

            />

            <ThemedInput

                placeholder="Price"

                type="phone-pad"

                action={(price:number)=>setPrice(price)}

                value={price==0? '' : price.toString()}

            />

            <ThemedText style={styles.text}>Category</ThemedText>

            <ThemedView>

            <DropDownPicker

                    open={open}

                    value={category}

                    items={items}

                    setOpen={setOpen}

                    setValue={setCategory}

                    setItems={setItems}

                    placeholder="Select a category"

                    style={[styles.picker, { backgroundColor: colors.card }]}  // Background color from theme

                    dropDownContainerStyle={[styles.dropdown, { backgroundColor: colors.card }]}  // List background color from theme

                    textStyle={{ color: colors.text }}  // Text color from theme

                    labelStyle={{ fontWeight: 'bold' }}  // Bolding the labels

                />

            </ThemedView>

            <CurvedButton

                title="Add meal"

                action={async()=>await handleAddMeal()}

                style={{backgroundColor:"rgb(72, 0, 255)"}}

            />

        </ThemedView>

    )

}

const styles = StyleSheet.create({

    container:{

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'center',

        justifyContent: 'center',

        paddingHorizontal: 10,

        paddingVertical: 35,

        gap: 20,

        height: '100%',

        width: '100%',

    },

    text:{

        fontSize: 25,

        fontWeight: 'bold',

        height: 'auto',

        width: 'auto',

    },

    picker: {

        borderColor: '#ccc',

        borderWidth: 1,

        borderRadius: 8,

        height: 50,

        width: '90%',

    },

    dropdown: {

        borderColor: '#ccc',

        borderWidth: 1,

        width: '90%',

    },

})

* עמוד הוספת הודעה מהירה למלצר
* שפה: React Native + Typescript
* עמוד הוספת הודעה מהירה אשר המשתמש יכול לשלוח למלצר. עמוד זה מנוהל ע"י הבעל

import { useState, useEffect, use } from "react";

import { ThemedView } from "@/components/ThemedView";

import { ThemedText } from "@/components/ThemedText";

import { StyleSheet} from "react-native";

import ThemedInput from "@/components/ThemedInput";

import CurvedButton from "@/components/ui/CurvedButton";

import axios from "axios";

import ip from "@/Data/Addresses";

import AsyncStorage from "@react-native-async-storage/async-storage";

import ShowMessageOnPlat from "@/components/ui/ShowMessageOnPlat";

import { QuickMessage } from "@/Types/QuickMessage";

import { ScrollView } from "react-native-gesture-handler";

import TableNeedMessage from "@/components/ui/TableNeedMessage";

/\*\*

 \* AddQuickMessage component allows the user to input a quick message and

 \* add it using a button. It utilizes ThemedView for layout, ThemedText for

 \* displaying the title, ThemedInput for capturing the message, and CurvedButton

 \* for the action to add the message.

 \*

 \* @component

 \* @returns A component consisting of a themed view, text, input field, and button.

 \*/

export default function AddQuickMessage() {

    const [message, setMessage] = useState<string>("")

    const [existingMessages , setExistingMessages] = useState<QuickMessage[]>([])

/\*\*

 \* Sends a POST request to add a quick message to the server.

 \*

 \* This function retrieves the authentication token from AsyncStorage and

 \* uses it to authenticate the request. The message to be added is sent

 \* in the request body. On success, it displays a toast with the server's

 \* response message. On failure, it displays an error message in a toast.

 \*/

    const handleAddMessage = async() => {

        try{

            const token = await AsyncStorage.getItem('token')

            const res = await axios.post(`http://${ip.julian}:5256/api/owner/add/message`,

                {

                    message: message

                },

                {

                    headers:

                    {

                    'x-auth-token':token

                    }

                }

            )

            ShowMessageOnPlat(res.data)

        }

        catch(e){

            ShowMessageOnPlat(`${e}`)

        }

    }

    const getExistingMessages = async() => {

        try{

            const token = await AsyncStorage.getItem('token')

            const res = await axios.get(`http://${ip.julian}:5256/api/owner/messages`, {

                headers: {

                    'x-auth-token': token

                }

            })

            if(res.status === 200 && res.data){

                setExistingMessages(res.data)

            }

        }

        catch(e){

            alert(`${e}`)

        }

    }

    useEffect(() => {

        getExistingMessages()

    }, [])

    return(

        <ThemedView style={styles.container}>

            <ThemedText style={styles.text}>Quick Message</ThemedText>

            <ThemedInput

                placeholder="Quick Message"

                type="text"

                value={message}

                action={(msg)=>setMessage(msg)}

            />

            <CurvedButton title="Add" action={() => handleAddMessage()} style={{backgroundColor:"#4800ff"}}/>

            <ThemedText style={{marginTop: 20, fontSize: 20,fontWeight: 'bold'}}>Existing Messages</ThemedText>

            <ScrollView contentContainerStyle={styles.messageContainer}>

               {existingMessages.map((msg, index) => (

                  <TableNeedMessage key={index} message={msg.message} handleClick={() => {alert(msg.message)}} />

               ))}

            </ScrollView>

        </ThemedView>

    )

}

const styles = StyleSheet.create({

    container: {

        flex: 1,

        alignItems: 'center',

        justifyContent: 'center',

    },

    text: {

        fontSize: 20,

        fontWeight: 'bold',

        marginBottom: 10,

        marginTop: "82%",

    },

    messageContainer: {

        justifyContent:"space-between",

        marginTop: 20,

        gap: "10%",

    }

});

* עמוד הוספת שולחנות
* שפה: React Native + Typescript
* עמוד זה נמצא לשם הוספת שולחנות למסעדה ע"י הבעל לשם נוחות ניהול מספור השולחנות יהיה אוטומטי בשרת

import {useState} from 'react';

import ip from '@/Data/Addresses';

import axios from 'axios';

import { ThemedView } from '@/components/ThemedView';

import { ThemedText } from '@/components/ThemedText';

import ThemedInput from '@/components/ThemedInput';

import CurvedButton from '@/components/ui/CurvedButton';

import { StyleSheet, ToastAndroid } from 'react-native';

import AsyncStorage from '@react-native-async-storage/async-storage';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

/\*\*

 \* A form for the owner to add a table to the database. The form

 \* includes two inputs: capacity and isWindowSide. The capacity is

 \* the number of customers that can sit at the table, and isWindowSide

 \* is a boolean that indicates whether the table is window side or not.

 \* The form also includes two buttons: "Add Table" and "Cancel". The

 \* "Add Table" button will add the table to the database and alert the

 \* user that the table was added successfully. The "Cancel" button will

 \* navigate the user back to the previous screen.

 \*/

export default function AddTableForm() {

    const [capacity,setCapacity] = useState<number>(0);

    const [isWindowSide,setIsWindowSide] = useState<boolean>(false);

/\*\*

 \* Submits the form data to add a new table to the restaurant's database.

 \* This function first checks for user authentication via token retrieval.

 \* If the user is authenticated, it sends a POST request with the table's

 \* capacity and window side preference. On successful addition, it alerts

 \* the user. If an error occurs, it alerts the user with the error message.

 \*/

    const handleSubmit=async()=>{

        const token = await AsyncStorage.getItem("token");

        if(!token){

            alert("User is not authenticated.");

            return;

        }

        try{

            const res = await axios.post(`http://${ip.julian}:5256/api/owner/add/table`,{

                capacity:capacity,

                isWindowSide:isWindowSide

            },

            {

                headers:{

                    "Content-Type":"application/json",

                    "X-Auth-Token":token

                }

            })

            if(res && res.status===200){

                ShowMessageOnPlat("Table added successfully!");

            }

        }catch(e){

            ShowMessageOnPlat(`${e}`);

        }

    }

    return(

        <ThemedView style={styles.container}>

            <ThemedText style={styles.text}>Capacity:</ThemedText>

            <ThemedInput placeholder="Capacity" action={(text)=>setCapacity(parseInt(text))} type="phone-pad" />

            <ThemedText style={styles.text}>Is the table Window Side:</ThemedText>

            <CurvedButton title={isWindowSide?"Yes":"No"}

            action={()=>setIsWindowSide(!isWindowSide)}

            style={{backgroundColor:"rgb(72, 0, 255)"}}

            />

            <CurvedButton title="Add Table"

            action={async()=>await handleSubmit()}

            style={{backgroundColor:"rgb(72, 0, 255)"}}

            />

        </ThemedView>

    )

}

const styles = StyleSheet.create({

    container:{

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'center',

        justifyContent: 'center',

        paddingHorizontal: 10,

        paddingVertical: 35,

        gap: 20,

        height: '100%',

        width: '100%',

    },

    text:{

        fontSize: 25,

        fontWeight: 'bold',

        height: 'auto',

        width: 'auto',

    },

})

* עמוד הסרת מלצרים
* שפה: React Native + Typescript
* עמוד זה נועד להסרת מלצרים ע"י בעל אם המלצר נמצא במשמרת אז לא ניתן להסיר אותו

import { useState, useEffect } from "react";

import { ThemedView } from "@/components/ThemedView";

import { ThemedText } from "@/components/ThemedText";

import { StyleSheet, TouchableOpacity, View, ScrollView, ToastAndroid } from "react-native";

import axios from "axios";

import ip from "@/Data/Addresses";

import AsyncStorage from "@react-native-async-storage/async-storage";

import { GestureHandlerRootView } from "react-native-gesture-handler";

import { FireWaiterParams } from "@/Types/FireWaiterParams";

import CurvedButton from "@/components/ui/CurvedButton";

import ShowMessageOnPlat from "@/components/ui/ShowMessageOnPlat";

/\*\*

 \* FireStaff Component

 \*

 \* This component displays a list of waiters, along with a button to remove each waiter.

 \* Upon removing a waiter, the component updates the list of waiters and re-fetches from the API.

 \*

 \* @returns JSX.Element

 \*/

export default function FireStaff() {

    const [waiters, setWaiters] = useState<FireWaiterParams[]>([]);

/\*\*

 \* Fetches the list of waiters from the server and updates the component's state.

 \*

 \* @async

 \* @returns {Promise<void>} A Promise that resolves when the waiters list is successfully

 \* fetched and updated in the state, or rejects with an error if the request fails.

 \*

 \* @remarks

 \* This function retrieves the authentication token from AsyncStorage and uses it

 \* to make an authorized GET request to the server. Upon successful response, it

 \* maps the waiters' data to ensure proper handling of various field names and updates

 \* the state with the list of waiters. If an error occurs, it is caught and handled.

 \*/

    const fetchWaiters = async () => {

        try {

            const token = await AsyncStorage.getItem('token');

            const res = await axios.get(`http://${ip.julian}:5256/api/owner/waiters`, {

                headers: {

                    "X-Auth-Token": token,

                },

            });

            if (res && res.status === 200) {

                // Ensure proper mapping (handling both `id` and `\_id`)

                setWaiters(res.data.map((waiter: any) => ({

                    id: waiter.id || waiter.\_id,

                    name: waiter.name || waiter.Name

                })));

            }

        } catch (error) {

        }

    };

    /\*\*

     \* Removes a waiter from the restaurant's database.

     \*

     \* @param id The ID of the waiter to be removed.

     \*

     \* @returns A Promise that resolves if the waiter was removed successfully, or rejects if there is an error.

     \*

     \* @remarks

     \* This function is called when a user clicks the "Remove" button on a waiter in the "Fire Staff" screen.

     \* It removes the waiter from the restaurant's database and updates the component's state immediately.

     \* It also re-fetches the list of waiters from the API to ensure consistency.

     \*/

    const RemoveWaiter = async (id: string) => {

        try {

            const token = await AsyncStorage.getItem('token');

            const res = await axios.delete(`http://${ip.julian}:5256/api/owner/delete/waiter?id=${id}`, {

                headers: {

                    "X-Auth-Token": token,

                },

            });

            if(res && res.status===400){

                ShowMessageOnPlat(`${res.data}`);

                return;

            }

            if (res && res.status === 200) {

                ToastAndroid.show("Removed Waiter Successfully", ToastAndroid.LONG);

                // ✅ Remove from state immediately

                setWaiters(prevWaiters => prevWaiters.filter(waiter => waiter.id !== id));

                // ✅ Refetch from API to ensure consistency

                fetchWaiters();

            }

        } catch (error) {

            console.error("Error removing waiter:", error);

            ShowMessageOnPlat("Error removing waiter.");

        }

    };

    // ✅ useEffect similar to fetchMeals()

    useEffect(() => {

        fetchWaiters();

    }, []); // ✅ Runs only on component mount

    return (

        <GestureHandlerRootView style={{ flex: 1 }}>

            <ThemedView style={styles.container}>

                <ScrollView contentContainerStyle={styles.scrollContainer}>

                    {waiters.length === 0 ? (

                        <ThemedText style={styles.noWaitersText}>No waiters found.</ThemedText>

                    ) : (

                        waiters.map((waiter) => (

                            <ThemedView key={waiter.id} style={styles.waiterBox}>

                                <ThemedText style={styles.waiterText}>{waiter.name}</ThemedText>

                                <CurvedButton

                                    style={styles.removeButton}

                                    action={() => RemoveWaiter(waiter.id)}

                                    title="Remove"

                                />

                            </ThemedView>

                        ))

                    )}

                </ScrollView>

            </ThemedView>

        </GestureHandlerRootView>

    );

}

const styles = StyleSheet.create({

    container: {

        flex: 1,

        alignItems: 'center',

        justifyContent: 'center',

        width: '100%',

    },

    scrollContainer: {

        flexGrow: 1,

        justifyContent: 'center',

        alignItems: 'center',

        paddingVertical: 20,

    },

    waiterBox: {

        width: '90%',

        padding: 15,

        borderRadius: 10,

        marginBottom: 10,

        flexDirection: 'row',

        alignItems: 'center',

        justifyContent: 'space-between',

        shadowColor: "#351fff",

        shadowOffset: { width: 5, height: 5 }, // Higher shadow

        shadowOpacity: 1, // Full opacity

        shadowRadius: 10, // Spread out more

        elevation: 20, // Higher elevation for Android

    },

    waiterText: {

        fontSize: 18,

        fontWeight: 'bold',

    },

    removeButton: {

        backgroundColor: '#ff4d4d',

        padding: 10,

        borderRadius: 5,

    },

    noWaitersText: {

        fontSize: 20,

        fontWeight: "bold",

        color: "gray",

    },

});

* עמוד בית לבעל
* שפה: React Native + Typescript
* עמוד הבית של הבעל הוא עמוד אשר משרת כמרכז ניווט לכל אחד מעמודי הניהול של הבעל כגון הוספת\הסרת עובד, הוספת\הסרת שולחן וכו...

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import { useState,useEffect } from 'react';

import { StyleSheet, TouchableOpacity, ScrollView, Image, ToastAndroid } from 'react-native';

import AsyncStorage from '@react-native-async-storage/async-storage';

import LogoutButton from '@/components/LogoutButton';

import axios from 'axios';

import ip from '@/Data/Addresses';

import \* as signalR from '@microsoft/signalr';

import { NavigationProp } from '@/Routes/NavigationTypes';

import { useNavigation } from '@react-navigation/native';

import { Connection } from '@/Data/Hub';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

type OwnerDTO={

    id : string,

    name: string,

    email: string,

    phone: string,

    password: string,

    restaurantNumber: string

}

/\*\*

 \* The owner screen, which displays the main menu of options for the owner

 \*

 \* This screen is only accessible to the owner of the restaurant

 \*

 \* It displays 4 main options: add worker, add table, add meals, and view tables

 \* It also displays 4 main options to remove: remove worker, remove table, remove meals, and view active workers

 \*

 \* The owner can navigate to the respective screens using the buttons

 \*

 \* The owner can also logout by using the logout button

 \*/

function Owner() {

    const [owner, setOwner] =useState<OwnerDTO>({id: "", name: "" , email: "", phone: "", password: "", restaurantNumber: "" });

    const [signalRConnection, setSignalRConnection] = useState<signalR.HubConnection | null>(null);

    const navigation = useNavigation<NavigationProp>();

    useEffect(() => {

        /\*\*

         \* Fetches the owner data from AsyncStorage and sets the state of the owner to that data

         \* If no data is found, the owner state is not changed

         \*/

        const fetchOwner = async () => {

            const ownerData = await AsyncStorage.getItem("owner");

            if (ownerData) {

                setOwner(JSON.parse(ownerData));

            }

        };

        /\*\*

         \* Fetches the list of meals from the server and updates the AsyncStorage

         \* If the response is successful (200), the meals are stored in AsyncStorage

         \* If there is an error, an alert is displayed to the user

         \*/

        const fetchMeals = async () => {

            const token = await AsyncStorage.getItem('token');

            try {

                const res = await axios.get(`http://${ip.julian}:5256/api/owner/meals`, {

                    headers: { 'x-auth-token': token }

                });

                if (res && res.status === 200) {

                    await AsyncStorage.setItem('meals', JSON.stringify(res.data));

                }

            } catch {

                alert("An error occurred or waiter already online");

            }

        };

        fetchOwner();

        fetchMeals();

    }, []); // Runs only once on mount

    useEffect(() => {

        if (!owner.id) return; // Prevent connecting with an empty owner ID

        /\*\*

         \* Establishes a connection to the SignalR hub with the specified owner ID and "owner" privilage level

         \*

         \* The connection is established using the HubConnectionBuilder

         \*

         \* The connection is then stored in the component's state

         \*

         \* An event listener is set for the "ConnectNotification" event. When this event is triggered, an alert is displayed to the user

         \* and the session ID is stored in AsyncStorage

         \*

         \* If an error occurs during connection, an error is logged to the console

         \*/

        const connect = async () => {

            const connection = await Connection.connectHub(owner.id, "owner");

            try {

                setSignalRConnection(connection);

                if(connection){

                    // detach any old listeners

                    connection.off("ConnectNotification");

                    await connection.on("ConnectNotification", async(sid: string,isOkay: boolean) => {

                        if(isOkay){

                            ShowMessageOnPlat('Session established');

                            await AsyncStorage.setItem('sid', sid);

                        }

                    })

                }

            } catch (error) {

                console.error('SignalR connection error:', error);

            }

        };

        connect();

    }, [owner.id]); // Runs when owner.id changes

    return (

        <ThemedView style={styles.view}>

        <ScrollView contentContainerStyle={styles.container}>

            <ThemedText style={styles.header}>Welcome {owner.name}</ThemedText>

            <LogoutButton action={async()=>await signalRConnection?.stop()} />

            <ThemedView style={styles.gridContainer}>

                {/\* Left side - Add buttons \*/}

                <ThemedView style={styles.leftColumn}>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("WaiterSignup")}>

                        <ThemedText style={styles.largeText}>+</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Add Worker</ThemedText>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("AddTableForm")}>

                        <ThemedText style={styles.largeText}>+</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Add Table</ThemedText>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("AddMealForm")}>

                        <ThemedText style={styles.largeText}>+</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Add Meals</ThemedText>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("AddQuickMessage")}>

                    <ThemedText style={styles.largeText}>+</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Add Quick Message</ThemedText>

                    </TouchableOpacity>

                </ThemedView>

                {/\* Right side - Remove buttons \*/}

                <ThemedView style={styles.rightColumn}>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("FireStaff")}>

                        <ThemedText style={styles.largeText}>-</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Remove Worker</ThemedText>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("RemoveTable")}>

                        <ThemedText style={styles.largeText}>-</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Remove Table</ThemedText>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("RemoveMeal")}>

                        <ThemedText style={styles.largeText}>-</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Remove Meals</ThemedText>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.functionBox} onPress={() => navigation.navigate("RemoveQuickMessage")}>

                        <ThemedText style={styles.largeText}>-</ThemedText>

                        <ThemedText style={styles.boldSmallText}>Remove Quick Messages</ThemedText>

                    </TouchableOpacity>

                </ThemedView>

            </ThemedView>

        </ScrollView>

    </ThemedView>

    );

}

const styles = StyleSheet.create({

    view: {

        height: '100%',

        width: '100%',

    },

    container: {

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'center',

        justifyContent: 'flex-start',

        paddingTop: 70,

        gap: 20,

        paddingBottom: 20,

    },

    header: {

        fontSize: 25,

        fontWeight: 'bold',

        height: 'auto',

        width: 'auto',

    },

    gridContainer: {

        display: 'flex',

        flexDirection: 'row',

        justifyContent: 'space-between',

        width: '90%', // Ensure grid fits within the screen

    },

    leftColumn: {

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'flex-start', // Align to the left

        gap: 10,

        width: '48%',

    },

    rightColumn: {

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'flex-end', // Align to the right

        gap: 10,

        width: '48%',

    },

    functionBox: {

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'center',

        justifyContent: 'center',

        width: '100%', // Takes the full width of the column

        aspectRatio: 1, // Keeps buttons square

        borderColor: "grey",

        borderWidth: 1,

        borderRadius: 5,

        padding: 10,

        gap: 10,

    },

    largeText: {

        fontSize: 60,

        fontWeight: 'bold',

        color: 'rgb(152, 76, 222)',

        flexShrink: 1,

        textAlign: 'center',

        paddingTop: 40,

    },

    boldSmallText: {

        fontSize: 15,

        fontWeight: 'bold',

        textAlign: 'center',

    }

});

export default Owner;

* עמוד כניסת הבעל
* שפה: React Native + Typescript
* עמוד הכניסה הראשי לטרמינל בעל מסעדה

import ThemedInput from '@/components/ThemedInput';

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import CurvedButton from '@/components/ui/CurvedButton';

import React,{ useState} from 'react';

import { StyleSheet } from 'react-native';

import Logo from '@/components/ui/Logo';

import axios from 'axios';

import ip from '@/Data/Addresses';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { NavigationProp } from '@/Routes/NavigationTypes';

import { useNavigation } from '@react-navigation/native';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

/\*\*

Component for the owner login screen

\*

@remarks

This component is navigated to when the user clicks on the "Owner Login" button on the login screen.

The component renders a logo, two input fields for the email and password, a login button, and a sign up button.

When the user clicks on the login button, the email and password are sent to the server to be validated.

If the validation is successful, the user is navigated to the Owner screen.

If the validation fails, the user is shown an alert with the error message.

When the user clicks on the sign up button, the user is navigated to the Owner Signup screen.

\*

@returns the Owner Login component

\*/

export default function OwnerLogin() {

const [email,setEmail] = useState<string>('');

const [password,setPassword] = useState<string>('');

const navigation = useNavigation<NavigationProp>();

/\*\*

Handles the login process for the owner.

Performs validation and sends a request to the server to

authenticate the user.

If the user is authenticated, sets the token and user data in

AsyncStorage and navigates to the Owner screen.

If the user is not authenticated, displays an error message.

\*/

const handleLogin = async() => {

try{

if(email==='' || password===''){

alert("Please fill all the fields");

return;

}

const res = await axios.post(`http://${ip.julian}:5256/api/owner`,{

email: email,

password: password

})

if(res && res.status===200){

await AsyncStorage.setItem('token', res.headers['x-auth-token']);

await AsyncStorage.setItem("owner",JSON.stringify(res.data.owner))

navigation.navigate("Owner");

}

}

catch(e){

alert(e);

}

}

return (

<ThemedView style={styles.container}>

<ThemedText style={styles.text}>Owner Login</ThemedText>

<Logo />

<ThemedInput

placeholder="Email"

type='email-address'

value={email}

action={(text) => setEmail(text)}

/>

<ThemedInput

placeholder="Password"

type='password'

value={password}

action={(text) => setPassword(text)}

/>

<CurvedButton

title="Login"

action={async() => await handleLogin()}

style={{backgroundColor:"rgb(153, 0, 255)"}}

/>

<CurvedButton title="Sign Up"

action={()=>navigation.navigate("OwnerSignup")}

style={{backgroundColor:"rgb(1, 119, 80)"}} />

</ThemedView>

);

}

const styles = StyleSheet.create({

container:{

display: 'flex',

flexDirection: 'column',

alignItems: 'center',

justifyContent: 'center',

padding: 20,

height: '100%',

width: '100%',

gap:25,

},

text:{

fontSize: 25,

fontWeight: 'bold',

height: 'auto',

width: 'auto',

},

});

* עמוד הרשמה כבעל
* שפה: React Native + Typescript
* עמוד זה הוא עמוד ההרשמה לבעל מסעדה

import ThemedInput from '@/components/ThemedInput';

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import CurvedButton from '@/components/ui/CurvedButton';

import { useState} from 'react';

import { StyleSheet, ToastAndroid } from 'react-native';

import axios from 'axios';

import ip from '@/Data/Addresses';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

/\*\*

The owner signup screen, which displays a form for the owner to sign up

\*

The form takes in the following fields: name, email, password, phone, and restaurant number

\*

When the form is submitted, the `handleSignup` function is called, which sends a POST request to the server with the data.

If the response is successful (200), the owner is signed up and a success alert is displayed

If there is an error, an alert is displayed to the user

\*

The user can navigate back to the login screen using the back button

\*/

export default function OwnerSignup() {

const [name,setName] = useState<string>('');

const [email,setEmail] = useState<string>('');

const [password,setPassword] = useState<string>('');

const [phone,setPhone] = useState<string>('');

const [restaurantNumber,setRestaurantNumber] = useState<string>('');

/\*\*

Handles the sign up request by sending a post request to the backend

with the user's details. If the request is successful, it shows an alert

with the response message. If there is an error, it shows an alert with

the error message.

\*

@function

@async

@returns {undefined}

\*/

const handleSignup = async() => {

try{

if(name==='' || email==='' || password==='' || phone==='' || restaurantNumber===''){

alert("Please fill all the fields");

return;

}

const res = await axios.post(`http://${ip.julian}:5256/api/owner/signup`,{

Name:name,

Email:email,

Password:password,

Phone: phone,

RestaurantNumber:restaurantNumber

})

if(res && res.status===200){

ShowMessageOnPlat("Signup Successful");

}

}

catch(e){

alert(e);

}

}

return (

<ThemedView style={styles.container}>

<ThemedText style={styles.text}>Owner Signup</ThemedText>

<ThemedInput type="text" placeholder="Name" action={(text) => setName(text)} value={name} />

<ThemedInput type="email-address" placeholder="Email" action={(text) => setEmail(text)} value={email} />

<ThemedInput type="password" placeholder="Password" action={(text) => setPassword(text)} value={password} />

<ThemedInput type="phone-pad" placeholder="Phone" action={(text) => setPhone(text)} value={phone} />

<ThemedInput type="phone-pad" placeholder="Restaurant Number" action={(text) => setRestaurantNumber(text)} value={restaurantNumber}/>

<CurvedButton title="Signup"

action={async() => await handleSignup()}

style={{backgroundColor:"rgb(72, 0, 255)"}} />

</ThemedView>

);

}

const styles = StyleSheet.create({

container:{

display: 'flex',

flexDirection: 'column',

alignItems: 'center',

justifyContent: 'center',

padding: 20,

height: '100%',

width: '100%',

gap:25,

},

text:{

fontSize: 25,

fontWeight: 'bold',

height: 'auto',

width: 'auto',

},

});

* עמוד הסרת ארוחה
* שפה: React Native + Typescript
* עמוד זה הוא עמוד אשר נועד להסרת ארוחות שנמצאות בתפריט המסעדה

import { useState, useEffect } from "react";

import { ThemedView } from "@/components/ThemedView";

import { ThemedText } from "@/components/ThemedText";

import { StyleSheet, ToastAndroid} from "react-native";

import axios from "axios";

import ip from "@/Data/Addresses";

import AsyncStorage from "@react-native-async-storage/async-storage";

import router from "expo-router";

import { ScrollView } from "react-native-gesture-handler";

import { GestureHandlerRootView } from "react-native-gesture-handler";

import { Meal } from "../../(Menu)/Menu";

import CurvedButton from "@/components/ui/CurvedButton";

import ShowMessageOnPlat from "@/components/ui/ShowMessageOnPlat";

/\*\*

Component for removing meals from the database and AsyncStorage.

Fetches meals from AsyncStorage when mounted and displays them in a list.

Each meal has a remove button that calls the RemoveMealFromDbAndStorage function.

If the meal is removed successfully, the UI is updated to reflect the change.

If there are no meals available, displays a "No meals available" message.

\*/

export default function RemoveMeal() {

const [meals, setMeals] = useState<Meal[]>([]);

/\*\*

Removes a meal from the database and AsyncStorage by its ID.

\*

This function performs the following steps:

Finds the meal by its ID from the current state.

Alerts if the meal is not found.

Sends a DELETE request to the server to remove the meal from the database.

Alerts the user if the meal is removed successfully.

Updates the local meals array and AsyncStorage to exclude the removed meal.

Updates the state to reflect the removal in the UI.

\*

@param {string} mealId - The ID of the meal to be removed.

\*/

const RemoveMealFromDbAndStorage = async (mealId: string) => {

try {

// ✅ Find the meal by ID before removing it

const mealToRemove = meals.find(meal => meal.mealId === mealId);

if (!mealToRemove) {

alert(`Meal with ID ${mealId} not found.`);

return;

}

const token = await AsyncStorage.getItem('token');

const res = await axios.delete(`http://${ip.julian}:5256/api/owner/delete/meal?mealId=${mealId}`,{

headers: {

"x-auth-token": token,

}});

if (res && res.status === 200) {

ShowMessageOnPlat(`${mealToRemove.mealName} removed successfully!\nPrice: ${mealToRemove.price}`);

}

// ✅ Remove meal from array

const updatedMeals = meals.filter(meal => meal.mealId !== mealId);

// ✅ Update AsyncStorage

await AsyncStorage.setItem("meals", JSON.stringify(updatedMeals));

// ✅ Update the state to reflect UI changes

setMeals(updatedMeals);

} catch (error) {

console.error("Error removing meal:", error);

}

};

useEffect(() => {

/\*\*

Fetches meals from AsyncStorage and updates the component state.

If meals are found, they are parsed and mapped to ensure correct type.

If meals are not found, the state remains unchanged.

If there is an error parsing meals, an error is logged to the console.

\*/

const fetchMeals = async () => {

try {

const m = await AsyncStorage.getItem("meals");

if (m) {

// Parse and map the data to ensure correct type

const parsedMeals: Meal[] = JSON.parse(m).map((meal: any) => ({

mealId: meal.mealId || "",

mealName: meal.mealName || "",

price: typeof meal.price === "number" ? meal.price : 0, // Ensure price is a number

category: meal.category || "",

}));

setMeals(parsedMeals);

}

} catch (error) {

console.error("Error parsing meals:", error);

}

};

fetchMeals();

}, []);

return (

<GestureHandlerRootView style={{ flex: 1 }}>

<ThemedView style={styles.container}>

<ScrollView contentContainerStyle={styles.scrollContainer}>

{meals.length > 0 ? (

meals.map((meal) => (

<ThemedView key={meal.mealId} style={styles.mealItem}>

<ThemedView>

<ThemedText style={styles.mealName}>{meal.mealName}</ThemedText>

<ThemedText style={styles.mealName}>{meal.category}</ThemedText>

</ThemedView>

<ThemedText style={styles.mealPrice}>${meal.price.toFixed(2)}</ThemedText>

<CurvedButton

style={styles.removeButton}

action={() => RemoveMealFromDbAndStorage(meal.mealId)}

title="Remove"

/>

</ThemedView>

))

) : (

<ThemedText style={styles.text}>No meals available</ThemedText>

)}

</ScrollView>

</ThemedView>

</GestureHandlerRootView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

alignItems: "center",

justifyContent: "center",

width: "100%",

paddingVertical: 25,

},

scrollContainer: {

flexGrow: 1, // Ensures ScrollView content expands properly

paddingVertical: 10,

},

mealItem: {

flexDirection: "row",

alignItems: "center",

justifyContent: "space-between",

paddingVertical: 12,

paddingHorizontal: 15,

borderBottomWidth: 1,  // ✅ Adds an HR-like separator

borderBottomColor: "#ccc", // ✅ Light gray line between items

width: "100%",

},

mealName: {

fontSize: 18,

fontWeight: "bold",

flex: 1, // ✅ Allows text to take available space

},

mealPrice: {

fontSize: 16,

marginRight: 10, // ✅ Provides spacing between price and button

},

removeButton: {

paddingHorizontal: 10,

paddingVertical: 5,

borderRadius: 5,

backgroundColor: "#e74c3c", // ✅ Matches previous remove button color

},

text: {

fontSize: 20,

fontWeight: "bold",

textAlign: "center",

marginTop: 50,

},

});

* עמוד הסרת הודעות מהירות
* שפה: React Native + Typescript
* עמוד זה הוא עמוד להסרת הודעות מהירות

import React, { useEffect, useState, useCallback } from 'react';

import {

StyleSheet,

Platform,

ToastAndroid,

ActivityIndicator,

Alert,

FlatList,

View,

Button,

} from 'react-native';

import AsyncStorage from '@react-native-async-storage/async-storage';

import axios from 'axios';

import { useNavigation } from '@react-navigation/native';

import { NavigationProp } from '@/Routes/NavigationTypes';

import { ThemedView } from '@/components/ThemedView';

import { ThemedText } from '@/components/ThemedText';

import { HapticTab } from '@/components/HapticTab';

import ip from '@/Data/Addresses';

import { QuickMessage } from '@/Types/QuickMessage';

const API\_BASE = `http://${ip.julian}:5256/api/owner`;

/\*\*

Component for removing quick messages.

\*

The component displays a list of quick messages. Users can select multiple

quick messages by pressing on them. The selected quick messages can be

removed by pressing the "Remove Selected" button.

\*/

function RemoveQuickMessage() {

const navigation = useNavigation<NavigationProp>();

const [quickMessages, setQuickMessages] = useState<QuickMessage[]>([]);

const [selectedIds, setSelectedIds] = useState<string[]>([]);

const [loading, setLoading] = useState<boolean>(true);

const [removing, setRemoving] = useState<boolean>(false);

/\*\*

Shows a toast message, or an alert box on iOS.

@param {string} msg The message to display.

\*/

const showToast = (msg: string) => {

if (Platform.OS === 'android') ToastAndroid.show(msg, ToastAndroid.SHORT);

else Alert.alert(msg);

};

/\*\*

Fetches the quick messages from the server.

\*/

const fetchQuickMessages = useCallback(async () => {

setLoading(true);

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.get<QuickMessage[]>(`${API\_BASE}/messages`, {

headers: { 'x-auth-token': token || '' },

});

setQuickMessages(response.data);

} catch (error) {

console.error('Fetch error:', error);

showToast('Failed to load quick messages.');

} finally {

setLoading(false);

}

}, []);

useEffect(() => {

fetchQuickMessages();

}, [fetchQuickMessages]);

/\*\*

Toggles the selection of a quick message.

@param {string} id The ID of the quick message to toggle.

\*/

const toggleSelection = useCallback((id: string) => {

setSelectedIds(prev =>

prev.includes(id) ? prev.filter(i => i !== id) : [...prev, id]

);

}, []);

/\*\*

Removes the selected quick messages from the server.

\*/

const handleRemove = useCallback(async () => {

if (!selectedIds.length) return;

setRemoving(true);

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.delete(`${API\_BASE}/delete/messages`, {

headers: { 'x-auth-token': token || '' },

data: { quickMessageIds: selectedIds },

});

if (response.status === 200) {

setQuickMessages(prev => prev.filter(msg => !selectedIds.includes(msg.id)));

setSelectedIds([]);

showToast('Selected messages removed.');

} else {

showToast('Error removing messages.');

}

} catch (error) {

console.error('Remove error:', error);

showToast('Failed to remove messages.');

} finally {

setRemoving(false);

}

}, [selectedIds]);

/\*\*

Renders a single item of the FlatList, representing a quick message.

The item is a HapticTab that can be pressed to toggle the quick message's

selection. The tab displays a checkbox and the text of the quick message.

If the quick message is selected, its checkbox is marked with a '☑' symbol.

Otherwise, it is marked with a '☐' symbol.

@param {object} props The props of the FlatList item, containing the quick message.

@returns {JSX.Element} A JSX element representing the FlatList item.

\*/

const renderItem = ({ item }: { item: QuickMessage }) => {

const isSelected = selectedIds.includes(item.id);

return (

<HapticTab

onPress={() => toggleSelection(item.id)}

style={[styles.item, isSelected && styles.selectedItem]}

>

<View style={styles.itemContent}>

<ThemedText style={[styles.checkbox, isSelected && styles.checkboxSelected]}>

{isSelected ? '☑' : '☐'}

</ThemedText>

<ThemedText style={styles.messageText}>{item.message}</ThemedText>

</View>

</HapticTab>

);

};

return (

<ThemedView style={styles.container}>

<ThemedText style={styles.title}>Remove Quick Messages</ThemedText>

{loading ? (

<ActivityIndicator size="large" style={styles.loader} />

) : quickMessages.length ? (

<FlatList

data={quickMessages}

keyExtractor={item => item.id}

renderItem={renderItem}

contentContainerStyle={styles.list}

/>

) : (

<ThemedText style={styles.emptyText}>No quick messages available.</ThemedText>

)}

<View style={styles.buttonContainer}>

<Button

title={removing ? 'Removing...' : 'Remove Selected'}

onPress={handleRemove}

disabled={!selectedIds.length || removing}

/>

</View>

</ThemedView>

);

}

export default RemoveQuickMessage;

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

justifyContent: 'flex-start',

},

title: {

fontSize: 22,

fontWeight: '700',

marginBottom: 12,

textAlign: 'center',

},

loader: {

marginTop: 20,

},

list: {

paddingBottom: 20,

},

item: {

padding: 12,

marginVertical: 6,

marginHorizontal: 4,

borderWidth: 1,

borderColor: '#ccc',

borderRadius: 8,

},

selectedItem: {

borderColor: '#EE0000',

},

itemContent: {

flexDirection: 'row',

alignItems: 'center',

},

checkbox: {

fontSize: 18,

marginRight: 12,

color: '#888',

},

checkboxSelected: {

color: '#e00',

},

messageText: {

fontSize: 16,

flex: 1,

},

emptyText: {

marginTop: 20,

fontSize: 16,

textAlign: 'center',

},

buttonContainer: {

marginTop: 'auto',

marginBottom: 16,

},

});

* עמוד הסרת שולחנות
* שפה: React Native + Typescript
* עמוד זה נועד למחיקת שולחנות ע"י הבעל

import { useState, useEffect } from "react";

import { ThemedView } from "@/components/ThemedView";

import { ThemedText } from "@/components/ThemedText";

import { StyleSheet } from "react-native";

import axios from "axios";

import ip from "@/Data/Addresses";

import AsyncStorage from "@react-native-async-storage/async-storage";

import { ScrollView } from "react-native-gesture-handler";

import { GestureHandlerRootView } from "react-native-gesture-handler";

import CurvedButton from "@/components/ui/CurvedButton";

export type Table = {

id: string;

capacity: number;

tableNumber: number;

isWindowSide: boolean;

isOccupied: boolean;

waiterId: string;

userId: string;

};

/\*\*

A page for the owner to remove tables from the database.

The page will fetch the current list of tables and display them

in a scrollable list, with each item containing the table number,

capacity, window side status, and occupied status. The owner can

delete a table by tapping the "Remove" button next to it.

The page will also display a message if no tables are available.

\*/

export default function RemoveTable() {

const [tables, setTables] = useState<Table[]>([]);

const [loading, setLoading] = useState<boolean>(true);

useEffect(() => {

/\*\*

Fetches the list of tables from the server and updates the AsyncStorage.

If the response is successful (200), the tables are stored in AsyncStorage

If there is an error, an alert is displayed to the user

Finally, the loading state is set to false

\*/

const fetchTables = async () => {

try {

const token = await AsyncStorage.getItem("token");

const res = await axios.get(`http://${ip.julian}:5256/api/owner/tables`, {

headers: { "x-auth-token": token },

});

if (res && res.status === 200) {

setTables(res.data);

await AsyncStorage.setItem("tables", JSON.stringify(res.data));

}

} catch (e) {

alert(e);

} finally {

setLoading(false);

}

};

fetchTables();

}, []);

/\*\*

Deletes a table from the database and updates the AsyncStorage.

@param number the table number to delete

@returns a promise that resolves when the deletion is complete

\*/

const handleDelete = async (number: number) => {

try {

// Retrieve stored tables

const storedTables = await AsyncStorage.getItem("tables");

let tablesArray = storedTables ? JSON.parse(storedTables) : [];

// Remove the table with the given number

const updatedTables = tablesArray.filter((table: { tableNumber: number }) => table.tableNumber !== number);

// Update AsyncStorage with the filtered list

await AsyncStorage.setItem("tables", JSON.stringify(updatedTables));

const token = await AsyncStorage.getItem("token");

const res = await axios.delete(

`http://${ip.julian}:5256/api/owner/delete/tables?number=${number}`,

{ headers: { "x-auth-token": token } }

);

if (res && res.status === 200) {

console.log("API Response:", res.data);

// Ensure tables exist in response before setting state

if (Array.isArray(res.data.tables)) {

setTables(res.data.tables);

await AsyncStorage.setItem("tables", JSON.stringify(res.data.tables));

} else {

setTables(updatedTables); // Fallback to filtered list if API response is missing tables

}

}

} catch (e) {

alert(e);

}

};

return (

<GestureHandlerRootView style={{ flex: 1 }}>

<ScrollView contentContainerStyle={styles.scrollContainer} showsVerticalScrollIndicator={false}>

<ThemedView style={styles.container}>

<ThemedText style={styles.text}>Remove Table</ThemedText>

{loading ? (

<ThemedText>Loading...</ThemedText>

) : Array.isArray(tables) && tables.length > 0 ? (

tables.map((table) => (

<ThemedView key={table.id} style={styles.tableItem}>

<ThemedText style={styles.tableText}>Table {table.tableNumber}</ThemedText>

<ThemedText style={styles.tableText}>Capacity: {table.capacity}</ThemedText>

<ThemedText style={styles.tableText}>

{table.isWindowSide ? "Window Side" : "Not Window Side"}

</ThemedText>

<ThemedText style={styles.tableText}>

{table.isOccupied ? "Occupied" : "Not Occupied"}

</ThemedText>

<CurvedButton

action={() => handleDelete(table.tableNumber)}

title={"Remove"}

style={{ backgroundColor: "red" }}

/>

</ThemedView>

))

) : (

<ThemedText>No tables available.</ThemedText>

)}

</ThemedView>

</ScrollView>

</GestureHandlerRootView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1, // Ensures full screen usage

alignItems: "center",

justifyContent: "center",

width: "100%",

},

scrollContainer: {

alignItems: "center",

width: "100%", // Ensures ScrollView is as wide as ThemedView

},

text: {

fontSize: 25,

fontWeight: "bold",

marginTop: 50,

},

tableItem: {

flexDirection: "column",

padding: 20,

marginVertical: 10,

borderRadius: 10,

borderColor: "#351fff",

borderWidth: 1,

width: "70%",

alignItems: "center",

shadowColor: "#351fff",

shadowOffset: { width: 5, height: 5 },

shadowOpacity: 1,

shadowRadius: 10,

elevation: 15, // Required for Android shadow

},

tableText: {

fontSize: 20,

textAlign: "center",

fontWeight: "bold",

padding: 10,

},

});

* עמוד הוספת מלצר
* שפה: React Native + Typescript
* עמוד זה הוא חלק מטרמינל הבעל והוא נועד להוספת מלצר

import ThemedInput from ‘@/components/ThemedInput’;

import { ThemedText } from ‘@/components/ThemedText’;

import { ThemedView } from ‘@/components/ThemedView’;

import CurvedButton from ‘@/components/ui/CurvedButton’;

import { useState} from ‘react’;

import { StyleSheet, ToastAndroid } from ‘react-native’;

import axios from ‘axios’;

import ip from “@/Data/Addresses”;

import AsyncStorage from ‘@react-native-async-storage/async-storage’;

import ShowMessageOnPlat from ‘@/components/ui/ShowMessageOnPlat’;

/\*\*

The waiter signup screen, which displays a form for the owner to sign up a waiter

\*

The form takes in the following fields: name, email, password, and phone

\*

When the form is submitted, the `handleSignup` function is called, which sends a POST request to the server with the data.

If the response is successful (200), the waiter is signed up and a success alert is displayed

If there is an error, an alert is displayed to the user

\*

The user can navigate back to the login screen using the back button

\*/

export default function WaiterSignup() {

const [email,setEmail] = useState<string>(‘’);

const [password,setPassword] = useState<string>(‘’);

const [name,setName] = useState<string>(‘’);

const [phone,setPhone] = useState<string>(‘’);

/\*\*

Handles the signup request for a waiter by sending a POST request to the server

with the waiter’s details. If the request is successful, it shows an alert

with the response message. If there is an error, it shows an alert with the

error message.

\*

@async

@returns {undefined}

\*/

const handleSignup = async() => {

try{

const token = await AsyncStorage.getItem(“token”);

const res = await axios.post(`http://${ip.julian}:5256/api/owner/add/waiter`,{

name:name,

email: email,

password: password,

phone: phone,

},

{

headers:{

“Content-Type”:”application/json”,

“X-Auth-Token”:token

}

})

if(res && res.status===200){

ShowMessageOnPlat(“Waiter added successfully!”);

}

}

catch€{

alert€;

}

}

return (

<ThemedView style={styles.container}>

<ThemedText style={styles.text}>Add Worker</ThemedText>

<ThemedInput type=”text” value={name} action={(text) => setName(text)} placeholder=”Name” />

<ThemedInput type=”email-address” value={email} action={(text) => setEmail(text)} placeholder=”Email” />

<ThemedInput type=”password” value={password} action={(text) => setPassword(text)} placeholder=”Password” />

<ThemedInput type=”phone-pad” value={phone} action={(text) => setPhone(text)} placeholder=”Phone” />

<CurvedButton title=”Add Worker” action={async()=>await handleSignup()} style={{backgroundColor:”#4800ff”}} />

</ThemedView>

);

}

const styles = StyleSheet.create({

container:{

display: ‘flex’,

flexDirection: ‘column’,

alignItems: ‘center’,

justifyContent: ‘center’,

padding: 20,

height: ‘100%’,

width: ‘100%’,

gap:25,

},

text:{

fontSize: 25,

fontWeight: ‘bold’,

height: ‘auto’,

width: ‘auto’,

},

});

* מסך צפייה בבקשה
* שפה: React Native + Typescript
* מסך זה נותן למלצר לצפות בפרטי בקשה לארוחות אשר סועד עשה

import { useState } from “react”;

import { StyleSheet, FlatList, View } from “react-native”;

import { ThemedView } from “@/components/ThemedView”;

import { ThemedText } from “@/components/ThemedText”;

import { useRoute, RouteProp } from “@react-navigation/native”;

import { RootStackParamList } from “@/Routes/NavigationTypes”;

import { Order } from “@/Types/Order”;

import { useTheme } from “@react-navigation/native”;

/\*\*

A screen that displays the details of an order.

\*

This screen is used to display the details of an order that was placed

by a user. It displays each meal in the order, along with the quantity

and price of each meal. The total price of the order is also displayed

at the bottom of the screen.

\*

@returns A JSX element representing the OrderPeak screen.

\*/

export default function OrderPeak() {

const route = useRoute<RouteProp<RootStackParamList, ‘OrderPeak’>>();

const params = route.params;

const order = params && ‘order’ in params ? params.order : null;

const [orderDetails, setOrderDetails] = useState<Order | null>(order);

const { colors, dark } = useTheme();

return (

<ThemedView style={styles.container}>

<FlatList

data={orderDetails?.orders || []}

keyExtractor={(\_, index) => index.toString()}

contentContainerStyle={styles.list}

renderItem={({ item }) => (

<ThemedView

style={[

styles.item,

{ backgroundColor: dark ? “#2c2c2e” : “#f0f0f0” },

]}

>

<ThemedText style={[styles.mealText, { color: dark ? “#fff” : “#000” }]}>

{item.meal.mealName} x {item.quantity}

</ThemedText>

<ThemedText style={[styles.categoryText, { color: dark ? “#ccc” : “#444” }]}>

{item.meal.category+”\t\t”+item.meal.price.toFixed(2)} ₪

</ThemedText>

</ThemedView>

)}

ListFooterComponent={

orderDetails ? (

<View style={styles.totalContainer}>

<ThemedText style={styles.totalText}>

Total: {orderDetails.total.toFixed(2)} ₪

</ThemedText>

</View>

) : null

}

/>

</ThemedView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 50,

},

list: {

paddingBottom: 40,

},

item: {

marginBottom: 12,

padding: 16,

borderRadius: 12,

backgroundColor: ‘#2c2c2e’,

},

mealText: {

fontSize: 18,

fontWeight: ‘600’,

color: ‘#fff’,

marginBottom: 4,

},

categoryText: {

fontSize: 14,

color: ‘#aaa’,

},

totalContainer: {

marginTop: 20,

padding: 16,

borderTopWidth: 1,

borderTopColor: ‘#444’,

},

totalText: {

fontSize: 20,

fontWeight: ‘bold’,

color: ‘#00cc99’,

textAlign: ‘center’,

},

});

* מסך צפייה בצרכים
* שפה: React Native + Typescript
* מסך זה נותן למלצר לצפות בצרכי סועד צרכי סועד הם הודעות מהירות אשר הסועד שלח

import { useState, useEffect } from 'react';

import { ThemedView } from "@/components/ThemedView";

import { ThemedText } from "@/components/ThemedText";

import { useRoute, RouteProp } from '@react-navigation/native';

import { RootStackParamList } from "@/Routes/NavigationTypes";

import { StyleSheet } from 'react-native';

import { Connection } from '@/Data/Hub';

import { useTheme } from '@react-navigation/native';

/\*\*

Component to display the peak needs for a given table.

\*

@param {number} tableNumber - The table number to display peak needs for.

\*

This component will fetch the list of peak needs for the given table and display them

in a scrollable list. If there are no peak needs for the table, it will display a message

indicating that the customer does not need anything.

\*/

function PeakNeeds() {

const route = useRoute<RouteProp<RootStackParamList, 'PeakNeeds'>>();

const { tableNumber } = route.params as { tableNumber: number };

const [needs, setNeeds] = useState<string[]>([]);

const connection = Connection.getHub();

const { colors, dark } = useTheme();

useEffect(() => {}, [needs]);

useEffect(() => {

/\*\*

Fetches the user needs messages for a specific table.

\*

This function invokes the "GetUserNeeds" method on the SignalR connection

using the specified table number. It also listens for messages from the

server via the "ReceiveMessagesToWaiter" event, updating the state with

the received messages. If an error occurs during the process, it logs

the error to the console.

\*/

const getNeeds = () => {

try {

connection?.invoke("GetUserNeeds", tableNumber);

connection?.on("ReceiveMessagesToWaiter", (message: string[]) => {

setNeeds(message);

});

} catch (error) {

console.error("Error fetching needs:", error);

}

};

getNeeds();

return () => {

connection?.off("ReceiveMessagesToWaiter");

};

}, []);

return (

<ThemedView style={styles.container}>

<ThemedText style={styles.heading}>

Peak Needs for table {tableNumber}

</ThemedText>

{needs.length === 0 ? (

<ThemedText style={styles.emptyText}>

The customer does not need anything

</ThemedText>

) : (

<ThemedView style={styles.messagesContainer}>

{needs.slice().reverse().map((msg, idx) => (

<ThemedView style={[styles.messageBox, { backgroundColor: dark ? "#0c0c0e" : "#f0f0f0" }]} key={idx}>

<ThemedText  style={styles.message}>

{msg}

</ThemedText>

</ThemedView>

))}

</ThemedView>

)}

</ThemedView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'flex-start',

alignItems: 'center',

paddingTop: 40,

paddingHorizontal: 20,

},

heading: {

fontSize: 25,

fontWeight: 'bold',

marginBottom: 20,

},

emptyText: {

fontSize: 18,

fontStyle: 'italic',

color: '#888',

},

message: {

fontSize: 21,

marginVertical: 5,

padding: 10,

borderRadius: 10,

width: '100%',

textAlign: 'center',

},

messageBox:{

display: "flex",

flexDirection: "row",

justifyContent: "space-between",

alignItems: "center",

marginBottom: 10,

borderRadius: 10,

padding: 10,

},

messagesContainer: {

width: '100%',

flexDirection: 'column',

alignItems: 'flex-start',

},

});

export default PeakNeeds;

* מסך בית למלצר
* שפה: React Native + Typescript
* מסך זה הוא עמוד הבית של המלצר המסך יראה למלצר את כלל שולחנות המסעדה

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import React, { useEffect, useRef, useState } from 'react';

import { StyleSheet, ToastAndroid, ScrollView } from 'react-native';

import LogoutButton from '@/components/LogoutButton';

import AsyncStorage from '@react-native-async-storage/async-storage';

import \* as signalR from '@microsoft/signalr';

import { NavigationProp } from '@/Routes/NavigationTypes';

import { useNavigation } from '@react-navigation/native';

import WaiterTableCard from '@/components/WaiterTableCard';

import { WaiterTableProps } from '@/Types/WaiterTableProps';

import { TableProps } from '@/components/TableCard';

import { Order } from '@/Types/Order';

import { Connection } from '@/Data/Hub';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

type Waiter = { id: string; name: string; email: string; phone: string };

export default function Waiter() {

const [waiter, setWaiter] = useState<Waiter | null>(null);

const [signalRConnection, setSignalRConnection] = useState<signalR.HubConnection | null>(null);

const [tables, setTables] = useState<WaiterTableProps[]>([]);

const [orders, setOrders] = useState<Order[]>([]);

const navigation = useNavigation<NavigationProp>();

const pendingPeakTableRef = useRef<number | null>(null);

useEffect(() => {

/\*\*

Fetches the current waiter from AsyncStorage and updates the component's state.

\*

@async

@returns {Promise<void>} A Promise that resolves when the waiter data is successfully

fetched and updated in the state, or rejects with an error if the request fails.

\*

@remarks

This function retrieves the authentication token from AsyncStorage and uses it

to make an authorized GET request to the server. Upon successful response, it

maps the waiter's data to ensure proper handling of various field names and updates

the state with the waiter. If an error occurs, it is caught and handled.

\*/

const fetchWaiter = async () => {

try {

const waiterData = await AsyncStorage.getItem('waiter');

if (waiterData) setWaiter(JSON.parse(waiterData));

} catch (error) { alert(error); }

};

fetchWaiter();

}, []);

useEffect(() => {

/\*\*

Initializes the SignalR connection for the current waiter, and sets up

all the event handlers for the various hub events.

\*

@async

@returns {Promise<void>} A Promise that resolves when the connection is successfully

established and all event handlers are set up.

\*

@remarks

If the waiter is not logged in, or if there is an error in establishing the connection,

the function does nothing.

\*/

const initSignalR = async () => {

if (!waiter?.id) return;

try {

const connection = await Connection.connectHub(waiter.id, 'waiter');

if (!connection) return;

connection.off('ConnectNotification');

connection.off('ReceiveOrders');

connection.off('ReceiveTableMessage');

connection.off('ReceiveWaiterAssignMessage');

connection.off('ReceiveTableLeaveMessage');

connection.off('SendOrder');

connection.off('ReceiveOrderReadyMessage');

/\*\*

Handles the connection notification event.

\*

@param {string} sid - The session ID assigned by the server.

@param {boolean} isOkay - Indicates if the connection was successful.

@param {TableProps[]} \_tables - The array of table properties received from the server.

\*

@remarks

If the connection is successful, stores the session ID in AsyncStorage,

updates the state with the tables information, and displays a success message.

If the connection is not successful, the function returns early without any actions.

\*/

const onConnectNotification = async (sid: string, isOkay: boolean, \_tables: TableProps[]) => {

if (!isOkay) return;

await AsyncStorage.setItem('sid', sid);

setTables(\_tables.map((t) => ({

tableNumber: t.tableNumber,

waiterid: t.waiterId,

isOccupied: t.isOccupied,

userName: t.userName,

})));

ShowMessageOnPlat('Connected to the server');

};

/\*\*

Handles the table message event.

\*

@param {string} message - The message sent by the server.

@param {boolean} isOkay - Indicates if the operation was successful.

@param {string} userId - The MongoDB user ID for the user who occupied the table.

@param {number} tableNumber - The table number that was occupied.

@param {TableProps[]} \_tables - The array of table properties received from the server.

\*

@remarks

If the operation is not successful, displays the error message.

If the operation is successful, updates the state with the tables information and displays a success message.

\*/

const onReceiveTableMessage = (

message: string, isOkay: boolean, userId: string, tableNumber: number, \_tables: TableProps[]

) => {

if (!isOkay) { ShowMessageOnPlat(message); return; }

setTables(\_tables.map((t) => ({

tableNumber: t.tableNumber,

waiterid: t.waiterId,

isOccupied: t.isOccupied,

userName: t.userName,

})));

ShowMessageOnPlat(`Table ${tableNumber} is now occupied by ${userId}`);

};

/\*\*

Handles the waiter assignment event.

\*

@param {string} message - The message sent by the server.

@param {TableProps[]} \_tables - The array of table properties received from the server.

\*

@remarks

If the message indicates that the table is already occupied by another waiter, displays the error message.

If the message is successful, updates the state with the tables information.

\*/

const onReceiveWaiterAssignMessage = (message: string, \_tables: TableProps[]) => {

if (message.includes('Table is already occupied by waiter')) {

ShowMessageOnPlat(message);

}

setTables(\_tables.map((t) => ({

tableNumber: t.tableNumber,

waiterid: t.waiterId,

isOccupied: t.isOccupied,

userName: t.userName,

})));

};

/\*\*

Handles the waiter leave event.

\*

@param {TableProps[]} \_tables - The array of table properties received from the server.

\*

@remarks

Updates the state with the tables information.

\*/

const onReceiveWaiterLeaveMessage = (\_tables : TableProps[]) => {

setTables(\_tables.map((t) => ({

tableNumber: t.tableNumber,

waiterid: t.waiterId,

isOccupied: t.isOccupied,

userName: t.userName,

})));

}

/\*\*

Handles the updating of the state when a user leaves a table.

\*

@param {TableProps[]} \_tables - The array of table properties received from the server.

\*

@remarks

Updates the state with the tables information.

\*/

const onReceiveTableLeaveMessage = (\_tables: TableProps[]) => {

setTables(\_tables.map((t) => ({

tableNumber: t.tableNumber,

waiterid: t.waiterId,

isOccupied: t.isOccupied,

userName: t.userName,

})));

};

/\*\*

Handles the reception of incoming orders.

\*

@param {Order[]} incoming - An array of incoming order objects.

\*

@remarks

If the incoming orders array is not empty, updates the state with the new orders

and stores them in AsyncStorage. If an error occurs during storage, it is caught and ignored.

\*/

const onReceiveOrders = (incoming: Order[]) => {

if (!incoming?.length) return;

setOrders(incoming);

AsyncStorage.setItem('orders', JSON.stringify(incoming)).catch(() => {});

};

/\*\*

Handles the sending of an order to the server.

\*

@param {Order} order - The order object to be sent.

\*

@remarks

If the order is null or undefined, does nothing.

Otherwise, updates the state with the new order and stores it in AsyncStorage.

If an error occurs during storage, it is caught and ignored.

If the order is for the table that was previously requested, navigates to the OrderPeak screen

and resets the pending table number to null.

\*/

const onSendOrder = (order: Order) => {

if (!order) return;

setOrders((prev) => {

const updated = [...prev];

updated[order.tableNumber - 1] = order;

AsyncStorage.setItem('orders', JSON.stringify(updated)).catch(() => {});

return updated;

});

if (pendingPeakTableRef.current === order.tableNumber) {

navigation.navigate('OrderPeak', { order });

pendingPeakTableRef.current = null;

}

};

/\*\*

Handles the reception of the order ready message from the server.

\*

@param {Order} order - The order object that has been marked as ready.

@param {number} tblNum - The table number associated with the order.

\*

@remarks

If the order is null or undefined, does nothing.

Otherwise, updates the state with the new order and stores it in AsyncStorage.

If an error occurs during storage, it is caught and ignored.

Shows a toast message indicating that the order for the table is ready.

Customer devices handle nuking their own AsyncStorage["order"] when they receive this.

\*/

const onReceiveOrderReadyMessage = (order: Order, tblNum: number) => {

if (!order) return;

setOrders((prev) => {

const updated = [...prev];

updated[order.tableNumber - 1] = order;

AsyncStorage.setItem('orders', JSON.stringify(updated)).catch(() => {});

return updated;

});

ToastAndroid.show(`Order for table ${tblNum} is ready`, ToastAndroid.SHORT);

// Customer devices handle nuking their own AsyncStorage["order"] when they receive this.

};

connection.on('ConnectNotification', onConnectNotification);

connection.on('ReceiveOrders', onReceiveOrders);

connection.on('ReceiveTableMessage', onReceiveTableMessage);

connection.on('ReceiveWaiterAssignMessage', onReceiveWaiterAssignMessage);

connection.on('ReceiveTableLeaveMessage', onReceiveTableLeaveMessage);

connection.on('SendOrder', onSendOrder);

connection.on('ReceiveOrderReadyMessage', onReceiveOrderReadyMessage);

connection.on('ReceiveWaiterLeaveMessage', onReceiveWaiterLeaveMessage);

setSignalRConnection(connection);

} catch (error) {

console.error('SignalR connection error:', error);

}

};

initSignalR();

}, [waiter?.id, navigation]);

/\*\*

Handles the event when the waiter wants to view an order from a table.

\*

@param {number} tableNumber - The table number associated with the order.

\*

@remarks

If no SignalR connection is established, does nothing.

Otherwise, sets the current pending table number to the table number and

invokes the 'PeakOrder' method on the SignalR connection with the table number

as the argument. If an error occurs during invocation, it is caught and ignored.

\*/

const handlePeakOrder = (tableNumber: number) => {

if (!signalRConnection) return;

pendingPeakTableRef.current = tableNumber;

signalRConnection.invoke('PeakOrder', tableNumber).catch(() => {});

};

/\*\*

Handles the event when the waiter wants to wait a table.

\*

@param {number} tableNumber - The table number associated with the order.

\*

@remarks

If no SignalR connection is established, does nothing.

Otherwise, invokes the 'AssignWaiterToTable' method on the SignalR connection with

the waiter's ID and the table number as arguments. If an error occurs during invocation,

it is caught and ignored.

\*/

const handleWaitTable = (tableNumber: number) => {

if (!signalRConnection) return;

signalRConnection.invoke('AssignWaiterToTable', waiter?.id, tableNumber).catch(() => {});

};

/\*\*

Handles the event when the waiter wants to leave a table.

\*

@param {number} tableNumber - The table number associated with the order.

\*

@remarks

If no SignalR connection is established, does nothing.

Otherwise, invokes the 'StopWaitingTable' method on the SignalR connection with

the table number as an argument. If an error occurs during invocation, it is caught

and ignored.

\*/

const handleLeaveTable = (tableNumber: number) => {

if (!signalRConnection) return;

signalRConnection.invoke('StopWaitingTable', tableNumber).catch(() => {});

};

/\*\*

Handles the event when the waiter wants to view the peak needs for a table.

\*

@param {number} tableNumber - The table number associated with the order.

\*

@remarks

Navigates to the 'PeakNeeds' screen with the table number as a parameter.

\*/

const handlePeakNeeds = (tableNumber: number) => {

navigation.navigate('PeakNeeds', { tableNumber });

};

/\*\*

Handles the event when the waiter wants to mark an order as ready.

\*

@param {number} tableNumber - The table number associated with the order.

\*

@remarks

If no SignalR connection is established, does nothing.

Otherwise, invokes the 'MarkOrderAsReady' method on the SignalR connection with

the table number as an argument. If an error occurs during invocation, it is caught

and ignored.

\*/

const handleMarkOrderReady = (tableNumber: number) => {

if (!signalRConnection) return;

signalRConnection.invoke('MarkOrderAsReady', tableNumber).catch(() => {});

};

return (

<ThemedView style={styles.safeArea}>

<ThemedView>

<ScrollView contentContainerStyle={styles.scrollContainer}>

<LogoutButton action={async () => await signalRConnection?.stop()} />

<ThemedText style={styles.text}>Hello {waiter?.name}</ThemedText>

{signalRConnection && tables.length > 0 ? (

<>

{tables.map((table) => (

<WaiterTableCard

key={table.tableNumber}

tableNumber={table.tableNumber}

occupyAction={() => handleWaitTable(table.tableNumber)}

leaveAction={() => handleLeaveTable(table.tableNumber)}

peakOrderAction={() => handlePeakOrder(table.tableNumber)}

markOrderReadyAction={() => handleMarkOrderReady(table.tableNumber)}

peakNeedAction={() => handlePeakNeeds(table.tableNumber)}

isOccupied={table.isOccupied}

waiterid={table.waiterid}

setter={setTables}

userName={table.userName}

/>

))}

</>

) : (

<ThemedText>An error occurred or there are no tables.</ThemedText>

)}

</ScrollView>

</ThemedView>

</ThemedView>

);

}

const styles = StyleSheet.create({

safeArea: { flex: 1 },

scrollContainer: {

paddingHorizontal: 20,

paddingTop: 40,

paddingBottom: 20,

alignItems: 'center',

},

text: { fontSize: 25, fontWeight: 'bold', marginBottom: 10, marginTop: 20 },

});

* מסך כניסה למלצר
* שפה: React Native + Typescript
* מסך הנועד לכניסת המלצר

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import ThemedInput from '@/components/ThemedInput';

import { useState} from 'react';

import { StyleSheet } from 'react-native';

import CurvedButton from '@/components/ui/CurvedButton';

import axios from 'axios';

import ip from '@/Data/Addresses';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { useNavigation } from '@react-navigation/native';

import { NavigationProp } from '@/Routes/NavigationTypes';

import Logo from '@/components/ui/Logo';

import ShowMessageOnPlat from '@/components/ui/ShowMessageOnPlat';

/\*\*

 \* Component for the waiter login screen

 \*

 \* @remarks

 \* This component is navigated to when the user clicks on the "Waiter Login" button on the login screen.

 \* The component renders a logo, two input fields for the email and password, a login button, and a sign up button.

 \* When the user clicks on the login button, the email and password are sent to the server to be validated.

 \* If the validation is successful, the user is navigated to the Waiter screen.

 \* If the validation fails, the user is shown an alert with the error message.

 \* When the user clicks on the sign up button, the user is navigated to the Waiter Signup screen.

 \*

 \* @returns the Waiter Login component

 \*/

export default  function WaiterLogin() {

    const [email,setEmail] = useState<string>('');

    const [password,setPassword] = useState<string>('');

    const navigation = useNavigation<NavigationProp>();

    /\*\*

     \* Handles the login process for the waiter.

     \* Performs validation and sends a request to the server to

     \* authenticate the user.

     \* If the user is authenticated, sets the token and user data in

     \* AsyncStorage and navigates to the Waiter screen.

     \* If the user is not authenticated, displays an error message.

     \*/

    const handleLogin = async() => {

        try{

            if(email==='' || password===''){

                ShowMessageOnPlat("Please fill all the fields");

                return;

            }

            const res = await axios.post(`http://${ip.julian}:5256/api/waiter`,{

                email: email,

                password: password

            })

            if(res && res.status===200){

                await AsyncStorage.setItem('token', res.headers['x-auth-token']);

                await AsyncStorage.setItem("waiter",JSON.stringify(res.data.waiter))

                navigation.navigate("Waiter")

            }

        }

        catch(e){

            alert(e);

        }

    }

    return (

        <ThemedView style={styles.container}>

            <ThemedText style={styles.text}>Waiter Login</ThemedText>

            <Logo />

            <ThemedInput

            type="email-address"

            placeholder="Email"

            action={(text)=>setEmail(text)}

            value={email}/>

            <ThemedInput

            type="password"

            placeholder="Password"

            action={(text)=>setPassword(text)}

            value={password} />

            <CurvedButton

            title="Login"

            action={async()=>await handleLogin()}

            style={{backgroundColor:"rgb(72, 0, 255)"}} />

        </ThemedView>

    );

}

const styles = StyleSheet.create({

    container:{

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'center',

        justifyContent: 'center',

        padding: 20,

        height: '100%',

        width: '100%',

        gap: 20,

    },

    text:{

        fontSize: 25,

        fontWeight: 'bold',

        height: 'auto',

        width: 'auto',

    },

})

* עמוד ניווט ראשי לאפליקציה
* שפה: React Native + Typescript
* מסך זה הוא המסך האחראי על הניווט בין כלל עמודי האפליקציה ניווט בין עמודי האפליקציה נבנה כמבני נתונים של מחסנית לשם נתינת היכולת למשתמש לחזור אחורה לעמוד אשר בו הוא היה

import { DarkTheme, DefaultTheme, NavigationContainer, ThemeProvider } from '@react-navigation/native';

import { useFonts } from 'expo-font';

import { Stack } from 'expo-router';

import \* as SplashScreen from 'expo-splash-screen';

//import { StatusBar } from 'expo-status-bar';

import { useEffect } from 'react';

import 'react-native-reanimated';

import { createStackNavigator } from '@react-navigation/stack';

import { NavigationProp } from "@/Routes/NavigationTypes";

import { useColorScheme } from '@/hooks/useColorScheme';

import Login from './(login)';

import Main from './(staff)/Main';

import Waiter from './(staff)/(waiter)/Waiter';

import Owner from './(staff)/(owner)/Owner';

import WaiterLogin from './(staff)/(waiter)/WaiterLogin';

import OwnerLogin from './(staff)/(owner)/OwnerLogin';

import WaiterSignup from './(staff)/(owner)/WaiterSignup';

import OwnerSignup from './(staff)/(owner)/OwnerSignup';

import AddMealForm from './(staff)/(owner)/AddMealForm';

import AddTableForm from './(staff)/(owner)/AddTableForm';

import NotFoundScreen from './+not-found';

import TabLayout from './(tabs)/\_layout';

import Signup from './(signup)/signup';

import RemoveTable from './(staff)/(owner)/RemoveTable';

import RemoveMeal from './(staff)/(owner)/RemoveMeal';

import FireStaff from './(staff)/(owner)/FireStaff';

import Menu from '@/app/(Menu)/Menu';

import UserNeeds from './(user)/UserNeeds';

import OrderPeak from './(staff)/(waiter)/OrderPeak';

import AddQuickMessage from './(staff)/(owner)/AddQuickMessage';

import Toast from 'react-native-toast-message';

import RemoveQuickMessage from './(staff)/(owner)/RemoveQuickMessage';

import PeakNeeds from './(staff)/(waiter)/PeakNeeds';

import {StatusBar,SafeAreaView, View } from 'react-native';

import { useSafeAreaInsets } from 'react-native-safe-area-context'

import AsyncStorage from '@react-native-async-storage/async-storage';

SplashScreen.preventAutoHideAsync();

/\*\*

 \* RootLayout sets up the main application layout and navigation structure.

 \* It initializes the font loading, theme setting based on the color scheme,

 \* and configures the stack navigator with various screens for the application.

 \* The layout includes a status bar and a toast notification component.

 \*

 \* Screens are grouped into sections for login, staff operations, signup,

 \* and tab navigation, each with header visibility options.

 \* The component also handles splash screen visibility based on font loading status.

 \*/

export default function RootLayout() {

  const colorScheme = useColorScheme();

  const [loaded] = useFonts({

    SpaceMono: require('../assets/fonts/SpaceMono-Regular.ttf'),

  });

  const Stack = createStackNavigator();

  const insets = useSafeAreaInsets();

  const STATUS\_BAR\_BG= colorScheme === 'dark' ? '#121212' : '#eee';

  useEffect(() => {

    if (loaded) {

      SplashScreen.hideAsync();

    }

  /\*\*

   \* Clean AsyncStorage by clearing all stored data. This is mainly used

   \* for debugging purposes when we want to start fresh.

   \*/

    const clean= async()=>{

      await AsyncStorage.clear();

      console.log("AsyncStorage cleared");

    }

    clean();

  }, [loaded]);

  if (!loaded) {

    return null;

  }

  return (

    <>

     <StatusBar

        translucent

        barStyle={colorScheme === 'dark' ? 'light-content' : 'dark-content'}

      />

      {/\* this view sits under the status bar \*/}

      <View style={{

        height: insets.top,

        backgroundColor: STATUS\_BAR\_BG

      }}/>

    <ThemeProvider value={colorScheme === 'dark' ? DarkTheme : DefaultTheme}>

            <Toast />

      <Stack.Navigator>

        {/\* Login is the initial screen \*/}

        <Stack.Screen

        name="Login"

        component={Login}

        options={{

          headerShown: false

        }}

      />

      {/\*Staff Screens \*/}

      <Stack.Screen name="Terminals" component={Main} options={{ headerShown: false }} />

      <Stack.Screen name="Waiter" component={Waiter} options={{ headerShown: false }} />

      <Stack.Screen name="Owner" component={Owner} options={{ headerShown: false }} />

      <Stack.Screen name="WaiterLogin" component={WaiterLogin} options={{ headerShown: false }} />

      <Stack.Screen name="OwnerLogin" component={OwnerLogin} options={{ headerShown: false }} />

      <Stack.Screen name="WaiterSignup" component={WaiterSignup} options={{ headerShown: false }} />

      <Stack.Screen name="OwnerSignup" component={OwnerSignup} options={{ headerShown: false }} />

      <Stack.Screen name="AddMealForm" component={AddMealForm} options={{ headerShown: false }} />

      <Stack.Screen name="AddTableForm" component={AddTableForm} options={{ headerShown: false }} />

      <Stack.Screen name="RemoveTable" component={RemoveTable} options={{ headerShown: false }} />

      <Stack.Screen name="RemoveMeal" component={RemoveMeal} options={{ headerShown: false }} />

      <Stack.Screen name="FireStaff" component={FireStaff} options={{ headerShown: false }} />

      <Stack.Screen name="Menu" component={Menu}  options={{ headerShown: false }} />

      <Stack.Screen name = "UserNeeds" component={UserNeeds} options={{ headerShown: false }} />

      <Stack.Screen name='OrderPeak' component={OrderPeak} options={{ headerShown: false }}/>

      <Stack.Screen name='AddQuickMessage' component={AddQuickMessage} options={{ headerShown: false }}/>

      <Stack.Screen name='RemoveQuickMessage' component={RemoveQuickMessage} options={{ headerShown: false }}/>

      <Stack.Screen name='PeakNeeds' component={PeakNeeds} options={{ headerShown: false }}/>

        {/\* Signup screen \*/}

        <Stack.Screen name="Signup" component={Signup} options={

          { headerShown: false  }}/>

        {/\* Tabs layout \*/}

        <Stack.Screen name="Tabs" component={TabLayout} options={{ headerShown: false }} />

        {/\* 404 or fallback \*/}

         <Stack.Screen name="+not-found" component={NotFoundScreen} />

      </Stack.Navigator>

    </ThemeProvider>

    <View style={{ height: insets.bottom, backgroundColor: STATUS\_BAR\_BG }} />

    </>

  );

}

* מסך עמוד לא נמצא
* שפה: React Native + Typescript
* עמוד זה יופיע רק אם המשתמש הלך או ניסה ללכת לעמוד אשר לא נמצא

import { Link, Stack } from 'expo-router';

import { StyleSheet } from 'react-native';

import { ThemedText } from '@/components/ThemedText';

import { ThemedView } from '@/components/ThemedView';

import React from 'react';

/\*\*

 \* NotFoundScreen is a React component that displays a message indicating

 \* that the requested screen does not exist. It provides a link for users

 \* to navigate back to the home screen. The component utilizes ThemedText

 \* and ThemedView for consistent styling across the application.

 \*/

export default function NotFoundScreen() {

  return (

    <>

      <Stack.Screen options={{ title: 'Oops!' }} />

      <ThemedView style={styles.container}>

        <ThemedText type="title">This screen doesn't exist.</ThemedText>

        <Link href="/" style={styles.link}>

          <ThemedText type="link">Go to home screen!</ThemedText>

        </Link>

      </ThemedView>

    </>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    alignItems: 'center',

    justifyContent: 'center',

    padding: 20,

  },

  link: {

    marginTop: 15,

    paddingVertical: 15,

  },

});

* קובץ כתובות IP
* שפה: React Native + Typescript
* קובץ זה נועד לשמור על כתובות ה-IP הנוכחיות של משתמש באפליקציה וזאת בגלל שלא אכתוב את כתובת ה-IP כל פעם שאני רוצה לפנות לשרת שלי דרך הטלפון

const ip={

    julian:"10.0.0.24".trim(), //Continue last digit here

    nitc:"192.168.122.19" //Continue last digit here too

}

export default ip

* קובץ חיבור ל-SignalR
* שפה: React Native + Typescript
* קובץ זה מנהל את היכולת לצור אובייקט החיבור עם השקעים (Sockets) בשרת ניהלנו את החיבור בנפרד בגלל שאם שמרנו את החיבור בקובץ של עמוד מסוים יקרא Garbage Collection על האובייקט דבר שיוביל לאיבוד החיבור

// src/Data/Hub.ts (or wherever your connectHub lives)

import \* as signalR from "@microsoft/signalr";

import ip from "@/Data/Addresses";

export class Connection {

  private static hub: signalR.HubConnection | null = null;

  /\*\*

   \* Sets the active hub connection.

   \* @param conn The active hub connection.

   \*/

  public static setHub(conn: signalR.HubConnection) {

    this.hub = conn;

  }

  /\*\*

   \* Gets the active hub connection.

   \* @returns The active hub connection.

   \*/

  public static getHub() {

    return this.hub;

  }

  public static connectHub = async (

    id: string,

    privilagelevel: "user" | "waiter" | "owner",

    userName?: string

  ): Promise<signalR.HubConnection | null> => {

    // build the correct query string

    let qs: string;

    if (privilagelevel === "user") {

      qs = `userid=${id}&privilagelevel=${privilagelevel}&name=${encodeURIComponent(

        userName ?? ""

      )}`;

    } else if (privilagelevel === "waiter") {

      qs = `waiterid=${id}&privilagelevel=${privilagelevel}`;

    } else {

      qs = `ownerid=${id}&privilagelevel=${privilagelevel}`;

    }

    const hubUrl = `http://${ip.julian}:5256/hub?${qs}`;

    const connection = new signalR.HubConnectionBuilder()

      .withUrl(hubUrl, {

        // force raw WebSockets (skip negotiate) and override Origin

        transport: signalR.HttpTransportType.WebSockets,

        skipNegotiation: true,

        headers: {

          // this must be a valid HTTP(S) origin, not a ws:// URL

          Origin: `http://${ip.julian}:19006`,

        },

      })

      .withAutomaticReconnect()

      .build();

    try {

      await connection.start();

      this.setHub(connection);

      return connection;

    } catch (err) {

      console.error("SignalR connection failed:", err);

      return null;

    }

  };

}

* רכיב UI למסך מלצרים – WaiterTableCard (כרטיס שולחן
* : React Native + TypeScript קומפוננטה פונקציונלית עם Hooks ו-AsyncStorage)
* רכיב שמציג מספר שולחן וסטטוס תפוסה, מזהה אם המלצר הנוכחי ממתין לשולחן, ומספק כפתור ראשי ל"המתן/עזוב" וכן פעולות נוספות כמו הצצה לצרכי הלקוח/הזמנה וסימון הזמנה כמוכנה.

import { useEffect, useState } from 'react';

import { StyleSheet } from 'react-native';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { ThemedView } from './ThemedView';

import { ThemedText } from './ThemedText';

import CurvedButton from '@/components/ui/CurvedButton';

import { WaiterTableProps } from '@/Types/WaiterTableProps';

/\*\*

 \* A table card for waiters.

 \*

 \* This component displays a table number, occupation status, and a main button that

 \* allows a waiter to wait a table if it is free or leave a table if they are

 \* currently waiting it. If the waiter is waiting the table, they will also see

 \* additional actions to peak at the customer's needs or order, or to mark the order

 \* as ready.

 \*

 \* @param {WaiterTableProps} props - The properties for this component:

 \*   - `tableNumber`: The number of the table.

 \*   - `waiterid`: The id of the waiter currently waiting the table, if any.

 \*   - `isOccupied`: Whether the table is occupied by a customer.

 \*   - `userName`: The username of the waiter currently waiting the table.

 \*   - `occupyAction`: An action to wait the table.

 \*   - `leaveAction`: An action to leave the table.

 \*   - `peakOrderAction`: An action to peak at the customer's order.

 \*   - `peakNeedAction`: An action to peak at the customer's needs.

 \*   - `markOrderReadyAction`: An action to mark the order as ready.

 \*/

export default function WaiterTableCard(props: WaiterTableProps) {

  const { tableNumber, waiterid, isOccupied, userName } = props;

  // Only thing we need locally: \*current waiter id\* for comparison

  const [currWaiterId, setCurrWaiterId] = useState<string | null>(null);

  useEffect(() => {

/\*\*

 \* Retrieves the current waiter's ID from AsyncStorage and updates the component's state.

 \*

 \* @async

 \* @returns {Promise<void>} A promise that resolves when the waiter ID is successfully fetched and updated in the state.

 \*

 \* @remarks

 \* This function accesses AsyncStorage to obtain the stored waiter object, parses it to extract the waiter's ID, and sets the state with the retrieved ID. If no waiter object is found, the state is updated with null.

 \*/

    const getId = async () => {

      const stored = await AsyncStorage.getItem('waiter');

      const id = stored ? JSON.parse(stored).id : null;

      setCurrWaiterId(id);

    };

    getId();

  }, []);

  // Derive everything from props + currWaiterId (no extra local mirrors)

  const isFree = !waiterid;                         // no one waiting this table

  const isSelf = !!currWaiterId && waiterid === currWaiterId;

  let mainButtonText = 'Table being waited';

  if (isFree) mainButtonText = 'Wait Table';

  else if (isSelf) mainButtonText = 'Leave Table';

  /\*\*

   \* Handles the main button action for the WaiterTableCard component.

   \*

   \* If the table is free, waits the table by calling the occupyAction

   \* prop. If the table is being waited by the current waiter, leaves the

   \* table by calling the leaveAction prop. If someone else is waiting, does

   \* nothing.

   \*/

  const mainButtonAction = () => {

    if (isFree) {

      // Wait table

      props.occupyAction?.();

    } else if (isSelf) {

      // Leave table

      props.leaveAction?.();

    } else {

      // Someone else is waiting

      // no-op or toast – keep as is to match your UX

    }

  };

  return (

    <ThemedView style={styles.container}>

      <ThemedText style={styles.waiterCardText}>Table {tableNumber}</ThemedText>

      <ThemedText style={styles.waiterCardText}>

        {isOccupied ? `Occupied by ${userName ?? 'guest'}` : 'Free'}

      </ThemedText>

      <CurvedButton

        title={mainButtonText}

        action={mainButtonAction}

        style={isFree ? { backgroundColor: '#4800ff' } : isSelf ? { backgroundColor: '#ff0a00' } : undefined}

      />

      {/\* Only show the extra actions if \*you\* are the waiter for this table \*/}

      {isSelf ? (

        <>

          <CurvedButton

            title="Peak Needs"

            action={() => props.peakNeedAction?.()}

            style={{ backgroundColor: '#fc9b1c' }}

          />

          <CurvedButton

            title="Peak Order"

            action={() => props.peakOrderAction?.()}

            style={{ backgroundColor: '#fc9b1c' }}

          />

          <CurvedButton

            title="Mark order as ready"

            action={() => props.markOrderReadyAction?.()}

            style={{ backgroundColor: '#4800ff' }}

          />

        </>

      ) : null}

    </ThemedView>

  );

}

const styles = StyleSheet.create({

  container: {

    flexDirection: 'column',

    justifyContent: 'center',

    alignItems: 'center',

    gap: 10,

    padding: 20,

    borderColor: 'gray',

    borderWidth: 1,

    borderRadius: 10,

    margin: 10,

    minHeight: 240,

    minWidth: '50%',

  },

  waiterCardText: {

    fontSize: 20,

    fontWeight: 'bold',

  },

});

* רכיב UI לכרטיס שולחן ללקוחות TableCard.
* : React Native + TypeScript קומפוננטה פונקציונלית עם Hooks, AsyncStorage ו־SignalR כ־prop לחיבור בזמן אמת
* כרטיס שמציג פרטי שולחן (מספר, חלון, קיבולת, סטטוס תפוסה), מזהה אם המשתמש הנוכחי משויך, מאפשר כניסה לתפריט או שיבוץ לשולחן, וכן עזיבת שולחן כשהמשתמש הוא התופס.

import React, { useEffect, useState } from "react";

import { StyleSheet, Image, TouchableOpacity} from "react-native";

import { ThemedText } from "./ThemedText";

import { ThemedView } from "./ThemedView";

import { useNavigation } from "@react-navigation/native";

import { NavigationProp } from "@/Routes/NavigationTypes";

import AsyncStorage from "@react-native-async-storage/async-storage";

import \* as signalR from '@microsoft/signalr';

import CurvedButton from "./ui/CurvedButton";

 export type TableProps = {

  tableNumber: number;

  isWindowSide: boolean;

  isOccupied: boolean;

  waiterId: string;

  userId: string;

  capacity : number;

  width: number |null | undefined;

  hub: signalR.HubConnection |null

  onAssignUserToTable: ()=>Promise<void>

  onLeaveTable: (tableNumber:number)=>Promise<void>

  userName?: string;

}

/\*\*

 \* A component that represents a table card with information and actions for a specific table.

 \*

 \* This component displays details about a table, such as its number, occupancy status,

 \* window side status, and capacity. It allows users to interact with the table card

 \* through touch events.

 \*

 \* - If the table is occupied, it checks if the current user is assigned to it. If not,

 \*   an alert is displayed; if yes, the user is navigated to the menu.

 \* - If the table is not occupied, the current user can be assigned to it using the

 \*   provided callback function.

 \* - Users can also leave the table through the "Leave" button, triggering an appropriate

 \*   callback.

 \*

 \* Props:

 \* - `tableNumber` (number): The number of the table.

 \* - `isWindowSide` (boolean): Indicates if the table is by the window.

 \* - `isOccupied` (boolean): Indicates if the table is currently occupied.

 \* - `waiterId` (string): The ID of the waiter assigned to the table.

 \* - `userId` (string): The ID of the user assigned to the table.

 \* - `capacity` (number): The seating capacity of the table.

 \* - `width` (number | null | undefined): The width of the table card.

 \* - `hub` (signalR.HubConnection | null): SignalR hub connection for real-time updates.

 \* - `onAssignUserToTable` (function): Callback to assign the current user to the table.

 \* - `onLeaveTable` (function): Callback for the user to leave the table.

 \*/

export default function TableCard(props: TableProps) {

  const {

    isOccupied,

    isWindowSide,

    userId,

    waiterId,

    capacity,

    tableNumber,

    width,

    hub,

    onAssignUserToTable,

    onLeaveTable

  } = props;

  const navigation = useNavigation<NavigationProp>();

  const [currentUserId, setCurrentUserId] = useState<string | null>(null);

  const [connection, setConnection] = useState<signalR.HubConnection | null>(null);

  const [loading, setLoading] = useState(true);

  useEffect(() => {

    /\*\*

     \* Initializes the component by fetching the user data from AsyncStorage

     \* and setting the userId state. Also sets the SignalR hub connection.

     \* If either fails, logs the error to the console.

     \* Finally, sets loading to false.

     \*/

    const init = async () => {

      try {

        const stringfiedUser = await AsyncStorage.getItem('user');

        const u = JSON.parse(stringfiedUser!);

        setCurrentUserId(u.id);

        setConnection(hub);

      } catch (e) {

        console.error("Failed to initialize user or hub:", e);

      } finally {

        setLoading(false);

      }

    };

    init();

  }, [hub]);

  /\*\*

   \* Handles the press event on the table card.

   \*

   \* This function checks if the current user is assigned to the table or not.

   \* - If the table is occupied by another user, an alert is shown indicating that

   \*   the table is already occupied.

   \* - If the table is occupied by the current user, an alert is displayed to notify

   \*   that the user is already assigned to the table, and then navigates to the menu.

   \* - If the table is not occupied, it calls the callback function to assign the

   \*   current user to the table.

   \*/

  const handlePress = async () => {

    if (!currentUserId) return;

    const stringfiedUser = await AsyncStorage.getItem('user');

    const u = JSON.parse(stringfiedUser!);

    if (isOccupied && userId !== u.id) {

      alert("Table is already occupied");

      return;

    }

    if (isOccupied && userId === u.id) {

      alert("You are already assigned to this table");

      navigation.navigate('Menu', { tableNumber});

      return;

    }

    if (!isOccupied && onAssignUserToTable) {

      onAssignUserToTable();

    }

  };

/\*\*

 \* Handles the event when the current user wants to leave the table.

 \*

 \* @async

 \* @remarks

 \* This function first checks if there is a current user ID. If not, it returns early.

 \* It retrieves the user data from AsyncStorage to verify the user's identity.

 \* If the current user is the one assigned to the table and the `onLeaveTable` callback is provided,

 \* it invokes the callback to handle the leave action.

 \*/

  const handleLeave = async () => {

    if (!currentUserId) return;

    const stringfiedUser = await AsyncStorage.getItem('user');

    const u = JSON.parse(stringfiedUser!);

    if (userId === u.id && onLeaveTable) {

      await onLeaveTable(tableNumber);

    }

  };

  if (loading) return null; // Or show a skeleton, shimmer, or placeholder

  return (

    <TouchableOpacity onPress={async ()=> await handlePress()}>

      <ThemedView style={[styles.container, { width }]}>

        <ThemedView style={styles.imageContainer}>

          <Image source={require("@/assets/images/table.png")} style={styles.image} />

        </ThemedView>

        <ThemedText style={styles.text}>{"Table:" + tableNumber}</ThemedText>

        <ThemedView style={styles.bottomInfoContainer}>

          <ThemedText style={styles.bottomInfoText}>

            {isOccupied ? "Occupied" : "Not Occupied"}

          </ThemedText>

          <ThemedText style={styles.bottomInfoText}>{isWindowSide ? "Window Side" : "Not Window Side"}</ThemedText>

          <ThemedText style={styles.bottomInfoText}>{"Capacity: " + capacity}</ThemedText>

        </ThemedView>

        {isOccupied && userId === currentUserId && (

          <>

            <TouchableOpacity

              onPress={() =>

                navigation.navigate("UserNeeds", { tableNumber })

              }

              style={styles.waiterImage}

            >

              <Image style={styles.image} source={require("@/assets/images/waiter.png")} />

            </TouchableOpacity>

            <CurvedButton

              title="Leave"

              action={handleLeave}

              style={{ backgroundColor: "red" }}

            />

          </>

        )}

      </ThemedView>

    </TouchableOpacity>

  );

}

const styles = StyleSheet.create({

  container: {

    width: 230,

    padding: 10,

    borderRadius: 10,

    borderColor:"grey",

    borderWidth: 1,

    marginVertical: 10,

    backgroundColor: 'transparent',

    flexDirection: 'column',

    alignItems: 'center',

    justifyContent: 'flex-start',

  },

  message:{

    position: 'absolute',

    top: -20,

    right: -16.5,

    width:'28%',

    height: '18%',

    backgroundColor: 'rgb(0, 177, 0)',

    borderRadius: 50,

    alignItems: 'center',

    justifyContent: 'center',

  },

  imageContainer: {

    borderRadius: 10,

    backgroundColor: 'transparent',

    marginBottom: 10,

  },

  waiterImage:{

    marginVertical: 5,

  },

  image: {

    width: 50,

    height: 50,

    borderRadius: 10,

    resizeMode: 'cover',

  },

  text: {

    fontSize: 20,

    fontWeight: 'bold',

    marginBottom: 10,

  },

  bottomInfoContainer: {

    flexDirection: 'column',

    alignItems: 'flex-start',

    width: '100%',

    backgroundColor: 'transparent',

    gap: 10,

  },

  bottomInfoText: {

    fontSize: 16,

    fontWeight: 'bold',

    backgroundColor: 'transparent',

  },

  bottomTableFunctions:{

    flexDirection: 'row',

    alignItems: 'center',

    justifyContent: 'space-around',

    width: 'auto',

    backgroundColor: 'transparent',

    gap: 40,

  },

});

* רכיב UI לכפתור מותאם אישית CurvedButton.
* React Native + TypeScript
* כפתור מעוגל עם צל והדגשה שמציג טקסט ומבצע פעולה בעת לחיצה, עם אפשרות להתאים את צבע הרקע והצל שלו.

import React from 'react'

import {

  StyleSheet,

  Text,

  TouchableOpacity,

  StyleProp,

  ViewStyle,

} from 'react-native'

type CurvedButtonPropTypes = {

  title: string

  action: () => void

  style?: StyleProp<ViewStyle>

}

/\*\*

 \* A button with a rounded rectangle shape and a drop shadow.

 \* @param {{title: string, action: () => void, style?: StyleProp<ViewStyle>}} props

 \* @prop {string} title - The text to display on the button.

 \* @prop {() => void} action - The function to call when the button is pressed.

 \* @prop {StyleProp<ViewStyle>} style - An optional style object to customize the button's appearance.

 \* The button will use the backgroundColor from this object as the color of its drop shadow.

 \* @returns {JSX.Element} A TouchableOpacity component with the button's style and title.

 \*/

export default function CurvedButton(props: CurvedButtonPropTypes) {

  // flatten into a real object so we can read backgroundColor

  const flat: ViewStyle = StyleSheet.flatten(props.style) || {}

  const shadowClr = flat.backgroundColor ?? 'black'

  const styles = StyleSheet.create({

    button: {

      width: 'auto',

      borderRadius: 7,

      padding: 5,

      alignItems: 'center',

      fontWeight: 'bold',

      paddingHorizontal: 15,

      shadowColor: shadowClr,

      shadowOffset: { width: 5, height: 5 },

      shadowOpacity: 1,

      shadowRadius: 10,

      elevation: 15, // Required for Android shadow

      zIndex: 10,

    },

    buttonText: {

      color: 'white',

      fontWeight: 'bold',

      fontSize: 17,

    },

  })

  return (

    <TouchableOpacity onPress={props.action} style={[props.style, styles.button]}>

      <Text style={styles.buttonText}>{props.title}</Text>

    </TouchableOpacity>

  )

}



using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MongoDB.Driver;

using Server.DB;

using Server.Models;

using Server.Services;

using Server.Security;

namespace Server.Controllers;

[ApiController]

[Route("api/user")] // Base route for all actions in this controller

/// <summary>

/// The user controller class is responsible for handling user-related actions such as login, sign-up, and account termination.

/// </summary>

public class UserController : ControllerBase

{

    IMongoCollection<User> \_users;

    IMongoCollection<Meal> \_meals;

    private readonly EmailService \_emailService;

    private readonly SecurityManager \_securityManager;

    /\*

     <summary>

     This is the user controller constructor to perform user actions

     using services such as an email service ETC

     </summary>

    \*/

    public UserController(MongoDBWrapper dBWrapper,

     EmailService emailService,

    SecurityManager securityManager

    )

    {

        \_users = dBWrapper.Users;

        \_meals = dBWrapper.Meals;

        \_emailService = emailService;

        \_securityManager = securityManager;

    }

    /// <summary>

    /// Authenticates a user and provides access to the system. This action is accessible to anonymous users.

    /// </summary>

    /// <returns>A successful status code (200) if the login request was successful.</returns>

    [AllowAnonymous]

    [HttpPost("login")] // Route: api/user/login

    public IActionResult Login([FromBody] Services.LoginRequest request)

    {

        try

        {

            if (string.IsNullOrEmpty(request.Email) || string.IsNullOrEmpty(request.Password))

            {

                return BadRequest("Invalid login credentials.");

            }

            var user = \_users.Find(user => user.Email == request.Email).FirstOrDefault();

            if (user == null)

            {

                return BadRequest("User not found.");

            }

            if (user != null && \_securityManager.Validate(request.Password, user.Password))

            {

                var token = \_securityManager.GenerateJwtToken(user.Id ?? new Guid().ToString(), request.Email);

                var resbod = new User

                { Id = user.Id, Name = user.Name, Email = user.Email, Password = user.Password, date = user.date, phone = user.phone };

                Response.Headers["X-Auth-Token"] = token;

                return Ok(new { User = resbod });

            }

            else

            {

                return BadRequest("Invalid login credentials or user not found." + $"{user?.Email ?? "No email or user found"}\nplain pass encrypted:{\_securityManager.Encrypt(request.Password)}\nalready encrypted:{user?.Password ?? "No password found"}");

            }

        }

        catch (Exception ex)

        {

            return StatusCode(500, $"An error occurred: {ex.Message}");

        }

    }

    /// <summary>

    /// Sign-up a new user using the provided user credentials. This action

    /// is accessible to anonymous users.

    /// </summary>

    /// <returns>A successful status code (200) if the sign-up request was

    /// successful.</returns>

    [AllowAnonymous]

    [HttpPost("signup")] // Route: api/user/signup

    public async Task<IActionResult> SignUp([FromBody] SignupRequest request)

    {

        try

        {

            if (string.IsNullOrEmpty(request.Email) || string.IsNullOrEmpty(request.Name) || string.IsNullOrEmpty(request.Password) || string.IsNullOrEmpty(request.date.ToString()))

            {

                return BadRequest("All fields are required.");

            }

            if (\_users.FindSync(user => user.Email == request.Email).Any())

            {

                return BadRequest("User already exists.");

            }

            var user = new User

            {

                Name = request.Name,

                Email = request.Email,

                Password = \_securityManager.Encrypt(request.Password),

                date = request.date,

                phone = request.phone

            };

            \_users.InsertOne(user);

            await \_emailService.SendEmailAsync(user.Email, "Welcome to The Service"

            ,

            $@"<html><body>Hello {user.Name}, <p>Welcome to Bite Byte.

                We're glad you're here. We're here to make it easier for you to dine out.</p>

                <br />

                <p>We hope you have a nice day!</p>

                The ReCoursia Team</body></html>");

            return Ok("Sign-up successful.");

        }

        catch (Exception ex)

        {

            System.Console.WriteLine(ex.ToString());

            return StatusCode(500, $"An error occurred: {ex.Message} at signup");

        }

    }

    /// <summary>

    /// Terminates the user account. This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) if the account termination was successful.</returns>

    [Authorize]

    [HttpDelete("terminate/account/{email}")] // Route: api/user/terminate/account

    public IActionResult TerminateAccount(string email)

    {

        var user = \_users.FindSync(user => user.Email == email).FirstOrDefault();

        if (user == null || (user is not null && string.IsNullOrEmpty(user.Email)))

        {

            return BadRequest("User not found.");

        }

        \_users.DeleteOne(user => user.Email == email);

        return Ok("Account terminated.");

    }

    /// <summary>

    /// A test endpoint to verify that the API is properly deployed.

    /// </summary>

    /// <returns>A successful status code (200) if the test request was successful.</returns>

    [HttpGet("test")]

    [AllowAnonymous]

    public IActionResult Test() => Ok("Test Successful");

    /// <summary>

    /// Retrieves the list of meals in the database. This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) if the request was successful, along with the list of meals as a JSON payload.</returns>

    [Authorize]

    [HttpGet("meals")]

    public async Task<IActionResult> GetMeals()

    {

        var dbfetch = await \_meals.Find(\_ => true).ToListAsync();

        var meals = dbfetch.ToArray();

        return Ok(new { meals });

    }

    /// <summary>

    /// Retrieves the list of tables currently available in the restaurant system.

    /// This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) along with the list of tables as a JSON payload.</returns>

    [Authorize]

    [HttpGet("tables")]

    public IActionResult GetTables()

    {

        var tables = SocketService.\_tables.ToArray();

        return Ok(new { tables });

    }

}

[ApiController]

[Route("api/waiter")]

/// <summary>

/// A controller for managing waiters and waiter actions in the system.

/// </summary>

public class WaiterController : ControllerBase

{

    IMongoCollection<Waiter> \_waiters;

    private readonly SecurityManager \_securityManager;

    /// <summary>

    /// A controller for managing waiters and waiter actions in the system.

    /// </summary>

    /// <param name="dBWrapper">The MongoDB wrapper for the database.</param>

    /// <param name="securityManager">The security manager for encrypting and decrypting passwords.</param>

    /// <remarks>

    /// The controller contains the following actions:

    /// <list type="bullet">

    /// <item><description>Authenticate a waiter and provide access to the system.</description></item>

    /// </list>

    /// </remarks>

    public WaiterController(MongoDBWrapper dBWrapper, SecurityManager securityManager)

    {

        \_waiters = dBWrapper.Waiters;

        \_securityManager = securityManager;

    }

    /// <summary>

    /// Authenticates a waiter and provides access to the system. This action is accessible to anonymous users.

    /// </summary>

    /// <param name="request">The waiter's login credentials.</param>

    /// <returns>A successful status code (200) if the login request was successful.</returns>

    /// <remarks>

    /// The request body should contain a JSON object with the following structure:

    /// <code>

    /// {

    ///     "Email": string,

    ///     "Password": string

    /// }

    /// </code>

    /// </remarks>

    [HttpPost]

    public async Task<IActionResult> Login([FromBody] WaiterLogin request)

    {

        if (string.IsNullOrEmpty(request.Email) && string.IsNullOrEmpty(request.Password))

        {

            return BadRequest("Credentials are missing.");

        }

        var cursor = await \_waiters.FindAsync<Waiter>(waiter => waiter.Email == request.Email);

        var waiter = cursor.FirstOrDefault();

        if (waiter is null)

        {

            return BadRequest("Waiter not found.");

        }

        if (string.IsNullOrEmpty(request.Email) || string.IsNullOrEmpty(request.Password))

        {

            return BadRequest("Invalid login credentials.");

        }

        Response.Headers["X-Auth-Token"] = \_securityManager.GenerateJwtToken(waiter.Id ?? new Guid().ToString(), request.Email);

        return Ok(new { Waiter = waiter });

    }

    [HttpGet("test")]

    public IActionResult Test() => Ok("Test Successful");

}

[ApiController]

[Route("api/owner")]

/// <summary>

/// A controller for managing restaurant owners and owner actions in the system.

/// </summary>

public class OwnerController : ControllerBase

{

    IMongoCollection<Owner> \_owners;

    IMongoCollection<Waiter> \_waiters;

    IMongoCollection<Meal> \_meals;

    IMongoCollection<Table> \_tables;

    IMongoCollection<QuickMessage> \_messages;

    private readonly SocketService \_socketService;

    private readonly SecurityManager \_securityManager;

    /// <summary>

    /// A controller for managing restaurant owners and owner actions in the system.

    /// The controller contains the following actions:

    /// <list type="bullet">

    /// <item><description>Authenticate an owner and provide access to the system.</description></item>

    /// </list>

    /// </summary>

    /// <param name="dBWrapper">An instance of the MongoDB wrapper class.</param>

    /// <param name="securityManager">An instance of the security manager class.</param>

    /// <param name="socketService">An instance of the socket service class.</param>

    public OwnerController(MongoDBWrapper dBWrapper, SecurityManager securityManager,

    SocketService socketService)

    {

        \_owners = dBWrapper.Owners;

        \_waiters = dBWrapper.Waiters;

        \_meals = dBWrapper.Meals;

        \_tables = dBWrapper.Tables;

        \_messages = dBWrapper.QuickMessages;

        \_securityManager = securityManager;

        \_socketService = socketService;

    }

    /// <summary>

    /// Authenticates an owner and provides access to the system. This action is accessible to anonymous users.

    /// </summary>

    /// <param name="request">The owner's login credentials.</param>

    /// <returns>A successful status code (200) if the login request was successful.</returns>

    /// <remarks>

    /// The request body should contain a JSON object with the following structure:

    /// <code>

    /// {

    ///     "Email": string,

    ///     "Password": string

    /// }

    /// </code>

    /// </remarks>

    [AllowAnonymous]

    [HttpPost]

    public async Task<IActionResult> Login([FromBody] Server.Services.LoginRequest request)

    {

        if (string.IsNullOrEmpty(request.Email) && string.IsNullOrEmpty(request.Password))

        {

            return BadRequest("Credentials are missing.");

        }

        var cursor = await \_owners.FindAsync<Owner>(owner => owner.Email == request.Email);

        var owner = cursor.FirstOrDefault();

        if (owner is null)

        {

            return BadRequest("Owner not found.");

        }

        if (!\_securityManager.Validate(request.Password, owner.Password))

        {

            return BadRequest("Invalid login credentials.");

        }

        var token = \_securityManager.GenerateJwtToken(owner.Id ?? new Guid().ToString(), request.Email);

        Response.Headers["X-Auth-Token"] = token; // ✅ Correctly set in response headers

        return Ok(new { Owner = owner });

    }

    /// <summary>

    /// Registers a new owner in the system.

    /// </summary>

    /// <param name="request">The owner's signup details including name, email, password, phone, and restaurant number.</param>

    /// <returns>A successful status code (200) if the signup is successful, otherwise a bad request error if required fields are missing or the owner already exists.</returns>

    /// <remarks>

    /// This endpoint is accessible to anonymous users.

    /// </remarks>

    [AllowAnonymous]

    [HttpPost("signup")]

    public async Task<IActionResult> SignUp([FromBody] OwnerSignupRequest request)

    {

        if (string.IsNullOrEmpty(request.Name) ||

            string.IsNullOrEmpty(request.Email) ||

            string.IsNullOrEmpty(request.Password) ||

            string.IsNullOrEmpty(request.Phone) ||

            string.IsNullOrEmpty(request.RestaurantNumber))

        {

            return BadRequest("All fields are required.");

        }

        var cursor = await \_owners.FindAsync<Owner>(owner => owner.Email == request.Email);

        var owner = cursor.FirstOrDefault();

        if (owner is not null)

        {

            return BadRequest("Owner already exists.");

        }

        var newOwner = new Owner

        {

            Name = request.Name,

            Email = request.Email,

            Password = \_securityManager.Encrypt(request.Password),

            Phone = request.Phone,

            RestaurantNumber = request.RestaurantNumber

        };

        await \_owners.InsertOneAsync(newOwner);

        return Ok("Sign-up successful.");

    }

    /// <summary>

    /// Adds a waiter to the restaurant staff.

    /// </summary>

    /// <param name="request">The waiter's details.</param>

    /// <returns>A successful status code (200) if the waiter was added successfully.</returns>

    /// <remarks>

    /// This action is restricted to authorized users (i.e. the restaurant owners).

    /// </remarks>

    [Authorize]

    [HttpPost("add/waiter")]

    public async Task<IActionResult> AddWaiter([FromBody] WaiterSignupRequest request)

    {

        if (string.IsNullOrEmpty(request.Name) ||

        string.IsNullOrEmpty(request.Email) ||

        string.IsNullOrEmpty(request.Password)

        || string.IsNullOrEmpty(request.Phone))

        {

            return BadRequest("All fields are required.");

        }

        var cursor = await \_waiters.FindAsync<Waiter>(waiter => waiter.Email == request.Email);

        var waiter = cursor.FirstOrDefault();

        if (waiter is not null)

        {

            return BadRequest("Waiter already exists.");

        }

        var newStaff = new Waiter

        {

            Name = request.Name,

            Email = request.Email,

            Password = \_securityManager.Encrypt(request.Password),

            Phone = request.Phone,

        };

        \_waiters.InsertOne(newStaff);

        return Ok("Waiter added successfully.");

    }

    /// <summary>

    /// Adds a new meal to the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <param name="meal">The meal object containing the name and price of the meal to be added.</param>

    /// <returns>A successful status code (200) if the meal was added successfully, or a bad request status code (400) if the meal already exists or required fields are missing.</returns>

    [Authorize]

    [HttpPost("add/meal")]

    public async Task<IActionResult> AddMeal([FromBody] Meal meal)

    {

        System.Console.WriteLine(meal.Category ?? "category is null");

        if (string.IsNullOrEmpty(meal.MealName) || meal.Price <= 0 || string.IsNullOrEmpty(meal.Category))

        {

            System.Console.WriteLine("Meal name and price are required.");

            return BadRequest("All fields are required.");

        }

        var cursor = await \_meals.FindAsync(m => m.MealName == meal.MealName);

        var existingMeal = cursor.FirstOrDefault();

        if (existingMeal is not null)

        {

            System.Console.WriteLine("Meal already exists.");

            return BadRequest("Meal already exists.");

        }

        await \_meals.InsertOneAsync(meal);

        return Ok("Meal added successfully.");

    }

    /// <summary>

    /// Adds a new table to the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <param name="request">The table data including capacity and window side preference.</param>

    /// <returns>A successful status code (200) if the table was added successfully, or a bad request status code (400) if the table capacity is invalid.</returns>

    [Authorize]

    [HttpPost("add/table")]

    public async Task<IActionResult> AddTable([FromBody] Table request)

    {

        if (request.Capacity <= 0)

        {

            return BadRequest("All fields are required.");

        }

        long tableCount = await \_tables.CountDocumentsAsync(FilterDefinition<Table>.Empty);

        request.TableNumber = (int)tableCount + 1;

        await \_tables.InsertOneAsync(request);

        return Ok("Table added successfully.");

    }

    /// <summary>

    /// Deletes a meal from the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <param name="mealId">The ID of the meal to be deleted.</param>

    /// <returns>A successful status code (200) if the meal was deleted successfully, or a bad request status code (400) if the meal was not found.</returns>

    [Authorize]

    [HttpDelete("delete/meal")]

    public async Task<IActionResult> DeleteMeal([FromQuery] string mealId)

    {

        var meal = await \_meals.FindOneAndDeleteAsync(m => m.MealId == mealId);

        if (meal is null)

        {

            return BadRequest("Meal not found.");

        }

        return Ok($"{meal.MealName} deleted successfully.");

    }

    /// <summary>

    /// Retrieves all meals from the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) along with the list of meals as a JSON payload.</returns>

    [Authorize]

    [HttpGet("meals")]

    public async Task<IActionResult> GetMeals()

    {

        var dbfetch = await \_meals.Find(\_ => true).ToListAsync();

        var meals = dbfetch.ToArray();

        return Ok(meals);

    }

    /// <summary>

    /// Retrieves the list of waiters from the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) along with the list of waiters as a JSON payload.</returns>

    [Authorize]

    [HttpGet("waiters")]

    public async Task<IActionResult> GetWaiters()

    {

        var dbfetch = await \_waiters.Find(\_ => true).ToListAsync();

        var waiters = dbfetch.ToArray();

        return Ok(waiters);

    }

    /// <summary>

    /// Deletes a waiter from the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <param name="id">The ID of the waiter to be deleted.</param>

    /// <returns>A successful status code (200) if the waiter was deleted successfully, or a bad request status code (400) if the waiter is online or was not found.</returns>

    /// <remarks>

    /// This action is restricted to authorized users (i.e. the restaurant owners).

    /// </remarks>

    [Authorize]

    [HttpDelete("delete/waiter")]

    public IActionResult RemoveWaiter([FromQuery] string id)

    {

        // Check if the waiter ID exists in the values of \_waiterConnections dictionary

        if (SocketService.\_waiterConnections.Values.Contains(id))

        {

            return BadRequest("Waiter is currently online and cannot be removed.");

        }

        // Proceed with removing the waiter from your database

        var result = \_waiters.DeleteOne(w => w.Id == id);

        if (result.DeletedCount == 0)

        {

            return BadRequest("Waiter not found.");

        }

        return Ok("Waiter removed successfully.");

    }

    /// <summary>

    /// Deletes a table from the restaurant's database and updates the table numbers accordingly.

    /// This action is restricted to authorized users.

    /// </summary>

    /// <param name="number">The table number to be deleted.</param>

    /// <returns>A successful status code (200) with the updated list of tables if the deletion is successful,

    /// or a bad request status code (400) if the table is occupied, other tables are occupied making shifting impossible,

    /// or if the table was not found.</returns>

    /// <remarks>

    /// The function checks if the table is occupied or if it exists. It prevents deletion if the table is occupied

    /// or if shifting other table numbers is not possible due to occupied tables.

    /// The table is removed from the database and from the SocketService list if all conditions are met.

    /// </remarks>

    [Authorize]

    [HttpDelete("delete/tables")]

    public async Task<IActionResult> DeleteTable([FromQuery] int number)

    {

        System.Console.WriteLine("Attempting to delete table number: " + number);

        // Find the table in the SocketService list

        var table = SocketService.\_tables.FirstOrDefault(t => t.TableNumber == number);

        // If the table is not found in the list, assume it’s not occupied and delete it from MongoDB

        if (table == null)

        {

            var result = await \_tables.DeleteOneAsync(t => t.TableNumber == number);

            if (result.DeletedCount == 0)

            {

                return BadRequest("Table not found.");

            }

            await ShiftTableNumbers(number);

            return Ok("Table deleted successfully.");

        }

        // If the table exists but is occupied, prevent deletion

        if (!string.IsNullOrEmpty(table.UserId))

        {

            return BadRequest("Table is currently in use and cannot be removed.");

        }

        // If other tables are occupied and shifting is not possible, prevent deletion

        if (SocketService.\_tables.Any(t => t.TableNumber > number && t.isOccupied))

        {

            return BadRequest("Tables are occupied and their indexes cannot be shifted down.");

        }

        // Delete the table from MongoDB

        var deleteResult = await \_tables.DeleteOneAsync(t => t.TableNumber == number);

        if (deleteResult.DeletedCount == 0)

        {

            return BadRequest("Table not found.");

        }

        // Remove the table from the SocketService list

        SocketService.\_tables.RemoveAll(t => t.TableNumber == number);

        // Shift table numbers in MongoDB and SocketService.\_tables

        await ShiftTableNumbers(number);

        var tables = await \_tables.Find(\_ => true).ToListAsync();

        return Ok(tables);

    }

    /// <summary>

    /// Shifts the table numbers down by one for all tables with numbers greater than the deleted table number.

    /// </summary>

    /// <param name="deletedTableNumber">The table number that was deleted.</param>

    /// <remarks>

    /// Shifting the table numbers is done in two steps: first, all tables in the database are updated in a bulk operation,

    /// and second, the in-memory list of tables is updated.

    /// </remarks>

    private async Task ShiftTableNumbers(int deletedTableNumber)

    {

        var tablesToShift = await \_tables

            .Find(t => t.TableNumber > deletedTableNumber)

            .SortBy(t => t.TableNumber)

            .ToListAsync();

        if (tablesToShift.Any())

        {

            // Bulk update MongoDB tables

            var bulkOps = new List<WriteModel<Table>>();

            foreach (var table in tablesToShift)

            {

                var filter = Builders<Table>.Filter.Eq(x => x.TableNumber, table.TableNumber);

                var update = Builders<Table>.Update.Set(x => x.TableNumber, table.TableNumber - 1);

                bulkOps.Add(new UpdateOneModel<Table>(filter, update));

            }

            await \_tables.BulkWriteAsync(bulkOps);

            // Update the in-memory list

            foreach (var table in tablesToShift)

            {

                table.TableNumber--;

            }

        }

    }

    /// <summary>

    /// Retrieves the list of tables from the restaurant's database. This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) along with the list of tables as a JSON payload.</returns>

    [Authorize]

    [HttpGet("tables")]

    public async Task<IActionResult> GetTables()

    {

        if (SocketService.\_tables.Count == 0)

        {

            var dbfetch = await \_tables.Find(\_ => true).ToListAsync();

            var tables = dbfetch.ToArray();

            return Ok(tables);

        }

        else

        {

            var servicefetch = SocketService.\_tables;

            var tables = servicefetch.ToArray();

            return Ok(tables);

        }

    }

    /// <summary>

    /// Adds a new quick message to the database. This action is restricted to authorized users.

    /// </summary>

    /// <param name="request">The quick message to be added.</param>

    /// <returns>A successful status code (200) if the message was added successfully, or a bad request status code (400) if the message is empty or already exists.</returns>

    [Authorize]

    [HttpPost("add/message")]

    public async Task<IActionResult> AddMessage([FromBody] QuickMessage request)

    {

        if (string.IsNullOrEmpty(request.Message))

        {

            return BadRequest("Message cannot be empty.");

        }

        var cursor = await \_messages.FindAsync<QuickMessage>(message => message.Message.Trim() == request.Message.Trim());

        if (cursor.Any())

        {

            return BadRequest("Message already exists.");

        }

        await \_messages.InsertOneAsync(request);

        return Ok($"Message \"{request.Message}\" added successfully.");

    }

    // DELETE api/owner/delete/message

    [HttpDelete("delete/message")]

    public async Task<IActionResult> DeleteMessage([FromBody] SingleDeleteRequest request)

    {

        if (!EnsureRestaurantEmpty())

        {

            return BadRequest(new { error = "Restaurant must be empty before deleting messages." });

        }

        var filter = Builders<QuickMessage>.Filter.Eq(m => m.Id, request.QuickMessageId);

        var deleted = await \_messages.FindOneAndDeleteAsync(filter);

        if (deleted == null)

        {

            return NotFound(new { error = "Message not found." });

        }

        return Ok(new { success = true, message = deleted.Message });

    }

    /// <summary>

    /// Deletes multiple quick messages from the database. This action is restricted to authorized users.

    /// </summary>

    /// <param name="request">Contains the list of quick message IDs to be deleted.</param>

    /// <returns>A successful status code (200) with the count of deleted messages, or a bad request status code (400) if the restaurant is not empty or no IDs are provided, or a not found status code (404) if no messages are found for the given IDs.</returns>

    [HttpDelete("delete/messages")]

    public async Task<IActionResult> DeleteMessages([FromBody] BulkDeleteRequest request)

    {

        if (!EnsureRestaurantEmpty())

        {

            return BadRequest(new { error = "Restaurant must be empty before deleting messages." });

        }

        if (request.QuickMessageIds == null || !request.QuickMessageIds.Any())

        {

            return BadRequest(new { error = "No message IDs provided." });

        }

        var filter = Builders<QuickMessage>.Filter.In(m => m.Id, request.QuickMessageIds);

        var result = await \_messages.DeleteManyAsync(filter);

        if (result.DeletedCount == 0)

        {

            return NotFound(new { error = "No messages found for given IDs." });

        }

        return Ok(new { success = true, deletedCount = result.DeletedCount });

    }

    /// <summary>

    /// Checks whether the restaurant is empty by ensuring that all tables are not occupied and have no assigned user IDs.

    /// </summary>

    /// <returns>true if the restaurant is empty, false otherwise.</returns>

    private bool EnsureRestaurantEmpty()

    {

        return SocketService.\_tables.All(t => string.IsNullOrEmpty(t.UserId) && t.isOccupied == false);

    }

    /// <summary>

    /// Retrieves the list of quick messages from the database. This action is restricted to authorized users.

    /// </summary>

    /// <returns>A successful status code (200) along with the list of quick messages as a JSON payload.</returns>

    [Authorize]

    [HttpGet("messages")]

    public async Task<IActionResult> GetMessages()

    {

        var dbfetch = await \_messages.Find(\_ => true).ToListAsync();

        QuickMessage[] messages = dbfetch.ToArray();

        return Ok(messages);

    }

}



using MongoDB.Driver;

using Server.Models;

namespace Server.DB;

/// <summary>

/// Wrapper class for MongoDB client and database.

/// This class is responsible for initializing the MongoDB client and database, and providing access to collections.

/// </summary>

public class MongoDBWrapper

{

    private readonly IMongoClient \_client;

    private readonly IMongoDatabase \_database;

    public IMongoCollection<User> Users { get; init; }

    public IMongoCollection<Waiter> Waiters { get; init; }

    public IMongoCollection<Owner> Owners { get; init; }

    public IMongoCollection<Meal> Meals { get; init; }

    public IMongoCollection<Table> Tables { get; init; }

    public IMongoCollection<QuickMessage> QuickMessages { get; init; }

    /// <summary>

    /// Initializes the MongoDB client and database, and sets up collections.

    /// </summary>

    /// <param name="configuration">The configuration to use for setting up the MongoDB client and database.</param>

    /// <exception cref="ArgumentException">Thrown if the MongoDB configuration is missing or invalid.</exception>

    public MongoDBWrapper(IConfiguration configuration)

    {

        // Fetch values directly using GetValue<T>

        var connectionString = configuration.GetValue<string>("DB:ConnectionString");

        var databaseName = configuration.GetValue<string>("DB:DatabaseName");

        if (string.IsNullOrWhiteSpace(connectionString) || string.IsNullOrWhiteSpace(databaseName))

        {

            throw new ArgumentException("MongoDB configuration is missing or invalid.");

        }

        // Initialize MongoDB client and database

        \_client = new MongoClient(connectionString);

        \_database = \_client.GetDatabase(databaseName);

        // Initialize collections

        Users = \_database.GetCollection<User>("Users");

        Waiters = \_database.GetCollection<Waiter>("Waiters");

        Owners = \_database.GetCollection<Owner>("Owners");

        Meals = \_database.GetCollection<Meal>("Meals");

        Tables = \_database.GetCollection<Table>("Tables");

        QuickMessages = \_database.GetCollection<QuickMessage>("QuickMessages");

    }

    public IMongoDatabase GetDatabase() => \_database;

}

// Example POCO classes for MongoDB documents



/// <summary>

///  A request object for deleting a single quick message by its ID.

///  This class is used to encapsulate the ID of the quick message to be deleted

/// </summary>

public class SingleDeleteRequest

{

    public string? QuickMessageId { get; set; }

}

/// <summary>

/// A request object for bulk deleting quick messages by their IDs.

/// This class is used to encapsulate an array of IDs for the quick messages to be deleted

/// </summary>

public class BulkDeleteRequest

{

    public string[]? QuickMessageIds { get; set; }

}



using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace Server.Models;

/// <summary>

/// Represents a meal in the restaurant

/// </summary>

public class Meal

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? MealId { get; set; }

    [BsonElement("name")]

    public string MealName { get; set; } = "";

    [BsonElement("price")]

    public double Price { get; set; } = 0;

    [BsonElement("category")]

    public string Category { get; set; } = "";

    public Meal() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="Meal"/> class.

    /// </summary>

    /// <param name="mealName">Name of the meal.</param>

    /// <param name="price">The price.</param>

    public Meal(string mealName, double price, string category)

    {

        MealName = mealName;

        Price = price;

        Category = category;

    }

}

/// <summary>

/// Represents an order in the restaurant

/// </summary>

public class ProtoOrder

{

    public Meal? Meal { get; set; }

    public int Quantity { get; set; }

    public ProtoOrder() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="Order"/> class.

    /// </summary>

    /// <param name="meal">The meal.</param>

    /// <param name="quantity">The quantity.</param>

    public ProtoOrder(Meal? meal, int quantity)

    {

        Meal = meal;

        Quantity = quantity;

    }

}

/// <summary>

/// Represents an order in the restaurant.

/// </summary>

public class Order

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? Id { get; set; }

    [BsonElement("meals")]

    public ProtoOrder[]? Orders { get; set; }

    [BsonElement("tableNumber")]

    public int TableNumber { get; set; } = 0;

    [BsonElement("total")]

    public double Total { get; set; } = 0;

    public bool IsReady { get; set; } = false;

    public Order() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="Order"/> class.

    /// </summary>

    /// <param name="orders">The orders.</param>

    /// <remarks>

    /// This constructor is used to create a new order

    /// directly from an array of ProtoOrders.

    /// </remarks>

    public Order(ProtoOrder[] orders)

    {

        Orders = orders;

    }

}



using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace Server.Models;

/// <summary>

/// Represents a meal in the restaurant

/// </summary>

public class Meal

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? MealId { get; set; }

    [BsonElement("name")]

    public string MealName { get; set; } = "";

    [BsonElement("price")]

    public double Price { get; set; } = 0;

    [BsonElement("category")]

    public string Category { get; set; } = "";

    public Meal() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="Meal"/> class.

    /// </summary>

    /// <param name="mealName">Name of the meal.</param>

    /// <param name="price">The price.</param>

    public Meal(string mealName, double price, string category)

    {

        MealName = mealName;

        Price = price;

        Category = category;

    }

}

/// <summary>

/// Represents an order in the restaurant

/// </summary>

public class ProtoOrder

{

    public Meal? Meal { get; set; }

    public int Quantity { get; set; }

    public ProtoOrder() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="Order"/> class.

    /// </summary>

    /// <param name="meal">The meal.</param>

    /// <param name="quantity">The quantity.</param>

    public ProtoOrder(Meal? meal, int quantity)

    {

        Meal = meal;

        Quantity = quantity;

    }

}

/// <summary>

/// Represents an order in the restaurant.

/// </summary>

public class Order

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? Id { get; set; }

    [BsonElement("meals")]

    public ProtoOrder[]? Orders { get; set; }

    [BsonElement("tableNumber")]

    public int TableNumber { get; set; } = 0;

    [BsonElement("total")]

    public double Total { get; set; } = 0;

    public bool IsReady { get; set; } = false;

    public Order() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="Order"/> class.

    /// </summary>

    /// <param name="orders">The orders.</param>

    /// <remarks>

    /// This constructor is used to create a new order

    /// directly from an array of ProtoOrders.

    /// </remarks>

    public Order(ProtoOrder[] orders)

    {

        Orders = orders;

    }

}



namespace Server.Models;

/// <summary>

/// Represents a message in the system.

/// This class is used to encapsulate the sender, recipient, and content of a message.

/// </summary>

public class Message

{

    public required string SenderId { get; set; }

    public required string RecipientId { get; set; }

    public required string message { get; set; }

}



using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace Server.Models;

/// <summary>

/// Represents a quick message in the system.

/// </summary>

public class QuickMessage

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? Id { get; set; }

    [BsonRequired]

    [BsonElement("message")]

    public string Message { get; set; } = "";

    public QuickMessage() { }

    /// <summary>

    /// Initializes a new instance of the <see cref="QuickMessage"/> class with a specified message.

    /// </summary>

    /// <param name="message">The message content of the quick message.</param>

    public QuickMessage(string message)

    {

        Message = message;

    }

}

/// <summary>

/// Represents a request to select quick messages for a specific table.

/// </summary>

public class SelectedNeedMessages

{

    public int TableNumber { get; set; } = 0;

    public string[]? messages { get; set; } = null;

}



using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace Server.Models;

/// <summary>

/// Represents a table in the restaurant.

/// This class is used to encapsulate the properties of a table, including its number, associated waiter, user, and occupancy status.

/// It also includes properties to indicate whether the table is window-side and its capacity.

/// </summary>

public class Table

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? Id { get; set; }

    [BsonElement("tableNumber")]

    public int TableNumber { get; set; } = 0;

    [BsonElement("waiterId")]

    public string? WaiterId { get; set; } = string.Empty;

    [BsonElement("userId")]

    public string? UserId { get; set; } = string.Empty;

    [BsonIgnore]

    [BsonElement("userName")]

    public string? UserName { get; set; } = string.Empty;

    [BsonElement("isWindowSide")]

    public bool isWindowSide { get; set; } = false;

    [BsonElement("isOccupied")]

    public bool isOccupied { get; set; } = false;

    [BsonElement("capacity")]

    public int Capacity { get; set; } = 2;

    public Table()

    {

        isOccupied = CheckOccupation();

    }

    public Table(bool isWindowSide) => this.isWindowSide = isWindowSide;

    public Table(int tableNumber, string? waiterId, string? userId, bool isWindowSide, bool isOccupied)

    {

        TableNumber = tableNumber;

        WaiterId = waiterId;

        UserId = userId;

        this.isOccupied = !string.IsNullOrEmpty(userId) || isOccupied;

        this.isOccupied = CheckOccupation();

    }

    public bool CheckOccupation() => isOccupied = !string.IsNullOrEmpty(UserId);

}



using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace Server.Models

{

    /// <summary>

    /// Represents a user in the system.

    /// </summary>

    public class User

    {

        [BsonId]

        [BsonRepresentation(BsonType.ObjectId)] // Ensures correct ObjectId serialization

        public string? Id { get; set; }

        [BsonElement("name")]

        public required string Name { get; set; } = "";

        [BsonElement("email")]

        public required string Email { get; set; } = "";

        [BsonElement("password")]

        public required string Password { get; set; } = "";

        [BsonElement("date")]

        public required DateOnly date { get; set; } = DateOnly.FromDateTime(DateTime.Now);

        [BsonElement("phone")]

        public required string phone { get; set; } = "";

        public User()

        { }

        public User(string name, string email, string password, DateOnly date)

        {

            Name = name;

            Email = email;

            Password = password;

        }

    }

}



using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace Server.Models;

/// <summary>

/// Represents a waiter in the restaurant.

/// </summary>

public class Waiter

{

    [BsonId]

    [BsonRepresentation(BsonType.ObjectId)]

    public string? Id { get; set; }

    [BsonElement("name")]

    public required string Name { get; set; }

    [BsonElement("email")]

    public required string Email { get; set; }

    [BsonElement("password")]

    public required string Password { get; set; }

    [BsonElement("phone")]

    public required string Phone { get; set; }

    // Parameterless constructor

    public Waiter() { }

    public Waiter(string name, string email, string password, string phone)

    {

        Name = name;

        Email = email;

        Password = password;

        Phone = phone;

    }

}

/// <summary>

/// Represents an owner in the restaurant.

/// </summary>

public class Owner : Waiter

{

    [BsonElement("restaurantNumber")]

    public required string RestaurantNumber { get; set; }

    public Owner() { }

    public Owner(string name, string email, string password, string phone, string restaurantNumber) :

     base(name, email, password, phone)

    {

        RestaurantNumber = restaurantNumber;

    }

}



using Server.Services;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using System.Security.Cryptography;

namespace Server.Security;

/// <summary>

/// Manages the security of the server's users.W

/// </summary>

public class SecurityManager

{

    private IConfiguration \_conf;

    private EmailService \_emailService;

    /// <summary>

    /// Initializes a new instance of the <see cref="SecurityManager"/> class.

    /// </summary>

    /// <param name="conf">The configuration settings.</param>

    /// <param name="emailService">The email service used for sending emails.</param>

    /// <remarks>

    /// This constructor sets up the security manager with the necessary configuration and email service dependencies required for handling security operations such as JWT token generation and email-based two-factor authentication.

    /// </remarks>

    public SecurityManager(IConfiguration conf, EmailService emailService)

    {

        \_conf = conf;

        \_emailService = emailService;

    }

    /// <summary>

    /// Generates a JWT token for a user that is valid for a set period of time.

    /// </summary>

    /// <param name="Id">The user's unique identifier.</param>

    /// <param name="email">The user's email address.</param>

    /// <returns>The JWT token.</returns>

    /// <remarks>

    /// The token is valid for 30 days from the time of generation.

    /// </remarks>

    public string GenerateJwtToken(string Id, string email)

    {

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_conf["Jwt:Key"] ?? string.Empty));

        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var claims = new[]

        {

                new Claim(ClaimTypes.Email, email),

                new Claim(JwtRegisteredClaimNames.Jti,Id )

            };

        var token = new JwtSecurityToken(

            issuer: \_conf["Jwt:Issuer"],

            audience: \_conf["Jwt:Audience"],

            claims: claims,

            expires: DateTime.UtcNow.AddDays(30),

            signingCredentials: creds

        );

        return new JwtSecurityTokenHandler().WriteToken(token);

    }

    /// <summary>

    /// Encrypts the given password with the BCrypt algorithm.

    /// </summary>

    /// <param name="password">The password to encrypt.</param>

    /// <returns>The encrypted password.</returns>

    /// <remarks>

    /// The strength of the encryption is determined by the <c>BCrypt.Net.BCrypt.MinimumHashingIterations</c> value.

    /// </remarks>

    public string Encrypt(string password)

    {

        return BCrypt.Net.BCrypt.HashPassword(password);

    }

    /// <summary>

    /// Validates the password by comparing it with the hashed password stored in the database.

    /// </summary>

    /// <param name="plainText">The plaintext password</param>

    /// <param name="password">The hashed password</param>

    /// <returns>True if the password is valid, false otherwise</returns>

    public bool Validate(string plainText, string password) => BCrypt.Net.BCrypt.Verify(plainText, password);

    public async Task<string> Send2FAEmailAsync(string to)

    {

        string number = GenerateSecureCode(6);

        await \_emailService.SendEmailAsync(to, "2FA Code",

        @$"<html><body>

        Your 2FA code is <strong>{number}</strong>

        </body></html>");

        return number;

    }

    /// <summary>

    /// Generates a secure code of the given length.

    /// </summary>

    /// <param name="length">The length of the code to generate.</param>

    /// <returns>The generated code.</returns>

    /// <remarks>

    /// The code is generated using the <see cref="RandomNumberGenerator"/> class, which

    /// provides a cryptographically secure random number generator.

    /// Each byte is converted to a digit (0-9) by taking the modulo 10 of the byte value.

    /// </remarks>

    string GenerateSecureCode(int length)

    {

        using var rng = RandomNumberGenerator.Create();

        var bytes = new byte[length];

        rng.GetBytes(bytes);

        var code = string.Empty;

        foreach (var b in bytes)

        {

            // Convert each byte to a digit (0-9)

            code += (b % 10).ToString();

        }

        return code;

    }

}



namespace Server.Services;

/// <summary>

/// Represents a request for user login.

/// </summary>

public class LoginRequest

{

    public required string Email { get; set; }

    public required string Password { get; set; }

}



using Mailjet.Client;

using Mailjet.Client.Resources;

using Newtonsoft.Json.Linq;

namespace Server.Services

{

    /// <summary>

    /// Represents a service for sending emails using the Mailjet API.

    /// This service is initialized with the Mailjet API key and secret from the configuration.

    /// </summary>

    public class EmailService

    {

        public string? \_apiKey = "";

        public string? \_apiSecret = "";

        /// <summary>

        /// Initializes a new instance of the <see cref="EmailService"/> class.

        /// </summary>

        /// <param name="configuration">The configuration.</param>

        /// <remarks>

        /// Reads the Mailjet API key and secret from the configuration and sets

        /// the instance fields.

        /// </remarks>

        public EmailService(IConfiguration configuration)

        {

            \_apiKey = configuration["Mailjet:ApiKey"];

            \_apiSecret = configuration["Mailjet:ApiSecret"];

        }

        /// <summary>

        /// Sends an email asynchronously using the Mailjet client.

        /// </summary>

        /// <param name="to">The recipient's email address.</param>

        /// <param name="subject">The subject of the email.</param>

        /// <param name="body">The HTML content of the email body.</param>

        /// <exception cref="Exception">Thrown when the email fails to send.</exception>

        /// <remarks>

        /// This method constructs a MailjetRequest with the provided email details

        /// and sends it using the MailjetClient. If the response indicates failure,

        /// an exception is thrown with the error message from Mailjet.

        /// </remarks>

        public async Task SendEmailAsync(string to, string subject, string body)

        {

            MailjetClient client = new MailjetClient(\_apiKey, \_apiSecret);

            MailjetRequest request = new MailjetRequest

            {

                Resource = Send.Resource

            }

            .Property(Send.FromEmail, "noreply.recoursia@gmail.com")

            .Property(Send.FromName, "ReCoursia Team")

            .Property(Send.Subject, subject)

            .Property(Send.HtmlPart, body)

            .Property(Send.Recipients, new JArray

                {

                new JObject

                {

                    {"Email", to}

                }

                });

            MailjetResponse response = await client.PostAsync(request);

            if (!response.IsSuccessStatusCode)

            {

                throw new Exception($"Failed to send email: {response.GetErrorMessage()}");

            }

        }

    }

}



namespace Server.Services

{

    /// <summary>

    /// Request object for waiter signup.

    /// </summary>

    public class WaiterSignupRequest

    {

        public string Name { get; set; } = string.Empty;

        public string Email { get; set; } = string.Empty;

        public string Password { get; set; } = string.Empty;

        public string Phone { get; set; } = string.Empty;

        /// <summary>

        /// Initializes a new instance of the <see cref="WaiterSignupRequest"/> class with specified details.

        /// </summary>

        /// <param name="name">The name of the waiter.</param>

        /// <param name="email">The email address of the waiter.</param>

        /// <param name="password">The password for the waiter's account.</param>

        /// <param name="phone">The phone number of the waiter.</param>

        public WaiterSignupRequest(string name, string email, string password, string phone)

        {

            Name = name;

            Email = email;

            Password = password;

            Phone = phone;

        }

    }

}



namespace Server.Services;

/// <summary>

/// Request object for owner signup.

/// </summary>

public class OwnerSignupRequest

{

    public string Name { get; set; }

    public string Email { get; set; }

    public string Password { get; set; }

    public string Phone { get; set; }

    public string RestaurantNumber { get; set; }

    public OwnerSignupRequest(string name, string email, string password, string phone, string restaurantNumber)

    {

        Name = name;

        Email = email;

        Password = password;

        Phone = phone;

        RestaurantNumber = restaurantNumber;

    }

}



namespace Server.Services;

/// <summary>

/// Request object for user signup.

/// </summary>

public class SignupRequest

{

    public required string Name { get; set; }

    public required string Email { get; set; }

    public required string Password { get; set; }

    public required DateOnly date { get; set; }

    public required string phone { get; set; }

}



namespace Server.Services;

using System.Collections.Concurrent;

using System.Threading.Tasks;

using Microsoft.AspNetCore.SignalR;

using MongoDB.Driver;

using Server.DB;

using Server.Models;

using System.Linq;

using System;

using System.Collections.Generic;

using System.Text;

using System.Net;

using System.Globalization;

/// <summary>

/// Interface for the SignalR hub service.

/// </summary>

public interface IHubService

{

    Task ConnectNotification(string sid, bool isOkay, List<Table> tables);

    Task ReceiveMessage(Message message);

    Task ReceiveTableMessage(string message, bool isOkay = false, string userId = "", int tableNum = 0, List<Table>? tables = null);

    Task ReceiveWaiterAssignMessage(string message, List<Table> tables);

    Task ReceiveTableLeaveMessage(List<Table> tables);

    Task ReceiveWaiterLeaveMessage(List<Table> tables);

    Task ReceiveOrderSuccessMessage(bool isOkay = false, Order? order = null);

    Task ReceiveOrders(List<Order?> orders);

    Task SendOrder(Order order);

    Task ReceiveQuickMessageList(List<QuickMessage> messages);

    Task ReceiveOrderReadyMessage(Order order, int tableNumber);

    Task ReceiveMessagesToWaiter(string[] msgs);

    Task ReceiveSuccessOrFail(string message);

}

/// <summary>

/// SignalR service for handling real-time communication between clients.

/// </summary>

public class SocketService : Hub<IHubService>

{

    public SocketService(MongoDBWrapper dBWrapper, IConfiguration configuration)

    {

        \_tableCollection = dBWrapper.Tables;

        \_messageCollection = dBWrapper.QuickMessages;

        \_userCollection = dBWrapper.Users;

        \_emailService = new EmailService(configuration);

    }

    IMongoCollection<Table> \_tableCollection;

    IMongoCollection<QuickMessage> \_messageCollection;

    IMongoCollection<User> \_userCollection;

    EmailService \_emailService;

    public static readonly ConcurrentDictionary<string, string> \_userConnections = new();   // sid => MongoDB User ID

    public static readonly ConcurrentDictionary<string, string> \_waiterConnections = new(); // sid => MongoDB Waiter ID

    public static readonly ConcurrentDictionary<string, string> \_ownerConnections = new();  // sid => MongoDB Owner ID

    public static readonly ConcurrentDictionary<string, HashSet<string>> \_userids2sid = new();   // UserID => set of sids

    public static readonly ConcurrentDictionary<string, HashSet<string>> \_waiterids2sid = new(); // WaiterID => set of sids

    public static readonly ConcurrentDictionary<int, string> \_tableToUser = new();   // TableNumber => UserID

    public static readonly ConcurrentDictionary<int, string> \_tableToWaiter = new(); // TableNumber => WaiterID

    public static readonly ConcurrentDictionary<string, SelectedNeedMessages> tableToNeeds = new(); // TableNumber(string) => needs

    public static List<Table> \_tables = new();   // In-memory tables snapshot

    public static List<Order?> \_orders = new();  // In-memory orders per table index (tableNumber - 1)

    #region Group Helpers (use connectionIds, not userId/waiterId)

    private Task AddUserToTableGroup(string userId, int tableNumber)

    {

        if (\_userids2sid.TryGetValue(userId, out var sids) && sids.Count > 0)

        {

            return Task.WhenAll(sids.Select(sid => Groups.AddToGroupAsync(sid, tableNumber.ToString())));

        }

        return Task.CompletedTask;

    }

    private Task RemoveUserFromTableGroup(string userId, int tableNumber)

    {

        if (\_userids2sid.TryGetValue(userId, out var sids) && sids.Count > 0)

        {

            return Task.WhenAll(sids.Select(sid => Groups.RemoveFromGroupAsync(sid, tableNumber.ToString())));

        }

        return Task.CompletedTask;

    }

    private Task AddWaiterToTableGroup(string waiterId, int tableNumber)

    {

        if (\_waiterids2sid.TryGetValue(waiterId, out var sids) && sids.Count > 0)

        {

            return Task.WhenAll(sids.Select(sid => Groups.AddToGroupAsync(sid, tableNumber.ToString())));

        }

        return Task.CompletedTask;

    }

    private Task RemoveWaiterFromTableGroup(string waiterId, int tableNumber)

    {

        if (\_waiterids2sid.TryGetValue(waiterId, out var sids) && sids.Count > 0)

        {

            return Task.WhenAll(sids.Select(sid => Groups.RemoveFromGroupAsync(sid, tableNumber.ToString())));

        }

        return Task.CompletedTask;

    }

    #endregion

    /// <summary>

    /// Called when a new connection is established.

    /// </summary>

    /// <remarks>

    /// This method is called when a new connection is established.

    /// It stores the connection id (sid) and the corresponding MongoDB user id in the dictionary.

    /// If the user is a waiter, it also stores the sid in a HashSet for the waiter's id.

    /// If the user is an owner, it stores the sid in a dictionary for the owner's id.

    /// If the user is a regular user, it stores the sid in a HashSet for the user's id.

    /// </remarks>

    public override async Task OnConnectedAsync()

    {

        if (\_tables.Count == 0 || \_tableCollection.CountDocuments(FilterDefinition<Table>.Empty) == 0)

        {

            Console.WriteLine("Tables are null or empty.");

            \_tables.Clear();

            \_tables.AddRange(await \_tableCollection.Find(\_ => true).ToListAsync());

            \_orders = Enumerable.Repeat<Order?>(null, \_tables.Count).ToList();

        }

        var sid = Context.ConnectionId;

        try

        {

            var httpContext = Context.GetHttpContext();

            if (httpContext?.Request.Query.ContainsKey("privilagelevel") == true)

            {

                var privilageLevel = httpContext.Request.Query["privilagelevel"].ToString();

                if (privilageLevel == "waiter")

                {

                    var waiterid = httpContext.Request.Query["waiterid"].ToString() ?? string.Empty;

                    \_waiterConnections[sid] = waiterid;

                    \_waiterids2sid.AddOrUpdate(waiterid,

                        new HashSet<string> { sid },

                        (key, existingSids) =>

                        {

                            existingSids.Add(sid);

                            return existingSids;

                        });

                    Console.WriteLine($"Waiter connected sid: {sid}\n waiterid: {waiterid}");

                    await Clients.Caller.ConnectNotification(sid, true, \_tables);

                    await Clients.Caller.ReceiveOrders(\_orders); // Send orders to the waiter

                }

                else if (privilageLevel == "owner")

                {

                    var ownerId = httpContext.Request.Query["ownerid"].ToString() ?? string.Empty;

                    \_ownerConnections[sid] = ownerId;

                    Console.WriteLine($"Owner connected sid: {sid}\n ownerid: {ownerId}");

                    await Clients.Caller.ConnectNotification(sid, true, \_tables);

                }

                else if (privilageLevel == "user" && httpContext.Request.Query.ContainsKey("userid"))

                {

                    var userid = httpContext.Request.Query["userid"].ToString() ?? string.Empty;

                    \_userConnections[sid] = userid;

                    \_userids2sid.AddOrUpdate(userid,

                        new HashSet<string> { sid },

                        (key, existingSids) =>

                        {

                            existingSids.Add(sid);

                            return existingSids;

                        });

                    Console.WriteLine($"User connected sid: {sid}\n userid: {userid}");

                    await Clients.Caller.ConnectNotification(sid, true, \_tables);

                }

            }

        }

        catch (Exception ex)

        {

            await Clients.Caller.ConnectNotification(sid, false, \_tables);

            Console.WriteLine($"Connection Error: {ex.Message}");

        }

        finally

        {

            await base.OnConnectedAsync();

        }

    }

    /// <summary>

    /// Assigns a user to a table and adds them to the SignalR group.

    /// </summary>

    public async Task AssignUserToTable(string userId, int tableNumber)

    {

        Console.WriteLine($"Assigning user {userId} to table {tableNumber}");

        var sid = Context.ConnectionId;

        // If user already seated, notify caller

        if (\_tables.Count > 0 && \_tables.Any(t => t.UserId == userId) && \_tableToUser.Any(kv => kv.Value == userId))

        {

            await Clients.Caller.ReceiveTableMessage("User Already in a table.", false, userId, 0, \_tables);

            return;

        }

        // Assign user

        \_tableToUser[tableNumber] = userId;

        var idx = tableNumber - 1;

        \_tables[idx].UserId = userId;

        \_tables[idx].CheckOccupation();

        \_tables[idx].UserName = Context.GetHttpContext()?.Request.Query["name"].ToString() ?? string.Empty;

        // Add ALL current connections for this user to the table group

        await AddUserToTableGroup(userId, tableNumber);

        await Clients.All.ReceiveTableMessage($"User {userId} joined Table {tableNumber}", true, userId, tableNumber, \_tables);

        Console.WriteLine($"User {userId} joined Table {tableNumber}");

        // Track this connection

        \_userConnections[sid] = userId;

        \_userids2sid.AddOrUpdate(userId,

            new HashSet<string> { sid },

            (key, existingSids) =>

            {

                existingSids.Add(sid);

                return existingSids;

            });

    }

    /// <summary>

    /// Sends the given quick messages to the waiter of the given table number.

    /// </summary>

    /// <param name="tableNumber">The table number to send the messages to.</param>

    /// <param name="quickMessages">The list of quick messages to be sent.</param>

    /// <exception cref="Exception">An exception is thrown if there is an error sending the messages to the waiter.</exception>

    public async Task SendMessagesToWaiter(SelectedNeedMessages needs)

    {

        try

        {

            if (string.IsNullOrEmpty(\_tables[needs.TableNumber - 1].WaiterId))

            {

                await Clients.Caller.ReceiveSuccessOrFail("No waiter assigned to this table.");

                return;

            }

            Console.WriteLine($"Sending messages to waiter for Table {needs.TableNumber}");

            if (needs.messages == null || needs.messages.Length == 0)

            {

                await Clients.Caller.ReceiveSuccessOrFail("No messages to send.");

                return;

            }

            // Store messages

            tableToNeeds.AddOrUpdate(needs.TableNumber.ToString(), needs, (key, oldValue) => needs);

            // Send to table group (will reach waiter if assigned to group)

            await Clients.Group(needs.TableNumber.ToString()).ReceiveMessagesToWaiter(needs.messages);

            await Clients.Caller.ReceiveSuccessOrFail("Messages sent to waiter successfully.");

        }

        catch (Exception ex)

        {

            Console.WriteLine($"Error sending messages to waiter: {ex.Message}");

            await Clients.Caller.ReceiveSuccessOrFail($"Failed to send messages: {ex.Message}.");

        }

        finally

        {

            Console.WriteLine($"Finished sending messages to waiter for Table {needs.TableNumber}");

        }

    }

    /// <summary>

    /// Sends a meal order to the server.

    /// </summary>

    /// <param name="order">The order to be sent.</param>

    /// <remarks>

    /// This method is called when a user sends a meal order from the menu.

    /// It stores the order in the in-memory list of orders and sends a message to the waiter and all users at the table

    /// with the order details.

    /// </remarks>

    public async Task OrderMeal(Order order)

    {

        if (order != null && order.Orders != null)

        {

            int tableNumber = order.TableNumber;

            Console.WriteLine($"Order for Table {tableNumber} received: {order.Orders.Length} items");

            \_orders[tableNumber - 1] = order;

            await Clients.Caller.ReceiveOrderSuccessMessage(true, order);

            var context = Context.GetHttpContext();

            if (context == null) return;

            string id = context.Request.Query["userid"].ToString() ?? string.Empty;

            Console.WriteLine($"Order sent in Table {tableNumber} from {id}");

        }

        else

        {

            Console.WriteLine("Order is null.");

            await Clients.Caller.ReceiveOrderSuccessMessage(false, null);

        }

    }

    /// <summary>

    /// Removes the user from the table and sends a message to all users about the table's updated status.

    /// </summary>

    /// <param name="tableNumber">The table number the user is leaving.</param>

    /// <remarks>

    /// This method is called when a user wants to leave a table.

    /// It sets the user ID of the table to an empty string and sends a message to all users about the table's updated status.

    /// </remarks>

    public async Task LeaveTable(int tableNumber)

    {

        var userId = Context.GetHttpContext()?.Request.Query["userid"].ToString() ?? string.Empty;

        \_tableToUser[tableNumber] = string.Empty;

        var idx = tableNumber - 1;

        \_tables[idx].UserId = string.Empty;

        \_tables[idx].isOccupied = false;

        \_tables[idx].UserName = string.Empty;

        // Remove ALL user connections from the table group

        await RemoveUserFromTableGroup(userId, tableNumber);

        // Clear order for the table (once)

        \_orders[idx] = null;

        await Clients.All.ReceiveOrders(\_orders);

        // Clear needs

        tableToNeeds.TryRemove(tableNumber.ToString(), out \_);

        await Clients.All.ReceiveTableLeaveMessage(\_tables);

        Console.WriteLine($"User {userId} left Table {tableNumber}");

    }

    /// <summary>

    /// Assigns a waiter to a table and adds them to the SignalR group.

    /// </summary>

    public async Task AssignWaiterToTable(string waiterId, int tableNumber)

    {

        var httpWaiterId = Context.GetHttpContext()?.Request.Query["waiterid"].ToString() ?? string.Empty;

        var connectionId = Context.ConnectionId;

        var idx = tableNumber - 1;

        if (idx < 0 || idx >= \_tables.Count)

        {

            await Clients.Caller.ReceiveWaiterAssignMessage($"Invalid table number: {tableNumber}", \_tables);

            return;

        }

        var table = \_tables[idx];

        // only assign if there is no waiter yet

        if (!string.IsNullOrEmpty(table.WaiterId))

        {

            await Clients.Caller.ReceiveWaiterAssignMessage($"Table {tableNumber} is already occupied by waiter {table.WaiterId}", \_tables);

            return;

        }

        // assign the waiter

        table.WaiterId = waiterId;

        \_tableToWaiter[tableNumber] = waiterId;

        // Add ALL waiter connections to the group

        await AddWaiterToTableGroup(waiterId, tableNumber);

        // broadcast

        await Clients.All.ReceiveWaiterAssignMessage($"Waiter {waiterId} joined Table {tableNumber}", \_tables);

        Console.WriteLine($"Waiter {waiterId} joined Table {tableNumber}");

        // track this connection

        \_waiterConnections[connectionId] = waiterId;

        \_waiterids2sid.AddOrUpdate(waiterId, new HashSet<string> { connectionId }, (key, set) => { set.Add(connectionId); return set; });

    }

    /// <summary>

    /// Sends the order for a table to the waiter who requested it.

    /// </summary>

    public async Task PeakOrder(int tableNumber)

    {

        var order = \_orders[tableNumber - 1];

        if (order == null)

        {

            Console.WriteLine($"No order found for Table {tableNumber}");

            return;

        }

        // Only send to the caller (the waiter who asked)

        await Clients.Caller.SendOrder(order);

        Console.WriteLine($"Order for Table {tableNumber} sent to waiter.");

    }

    /// <summary>

    /// Retrieves the user needs messages for a table.

    /// </summary>

    public async Task GetUserNeeds(int tableNumber)

    {

        if (tableToNeeds.TryGetValue(tableNumber.ToString(), out var needs))

        {

            await Clients.Caller.ReceiveMessagesToWaiter(needs.messages ?? Array.Empty<string>());

        }

        else

        {

            await Clients.Caller.ReceiveSuccessOrFail("No messages found for this table.");

        }

    }

    /// <summary>

    /// Retrieves a list of quick messages from the message collection in MongoDB and sends them to the caller.

    /// </summary>

    public async Task GetQuickMsgs()

    {

        var msgs = await FetchMessages();

        await Clients.Caller.ReceiveQuickMessageList(msgs);

    }

    /// <summary>

    /// Removes a waiter from a table and updates all clients about the table's updated status.

    /// </summary>

    public async Task StopWaitingTable(int tableNumber)

    {

        var waiterId = Context.GetHttpContext()?.Request.Query["waiterid"].ToString() ?? string.Empty;

        \_tables[tableNumber - 1].WaiterId = string.Empty;

        // Remove ALL waiter connections from the table group

        await RemoveWaiterFromTableGroup(waiterId, tableNumber);

        await Clients.All.ReceiveWaiterLeaveMessage(\_tables);

        Console.WriteLine($"Waiter {waiterId} left Table {tableNumber}");

    }

    /// <summary>

    /// Marks a meal order as ready and sends a message to all users at the table and the waiter with the order details.

    /// </summary>

    public async Task MarkOrderAsReady(int tableNumber)

    {

        Console.WriteLine($"Marking order for Table {tableNumber} as ready");

        var order = \_orders[tableNumber - 1];

        if (order != null && order.Orders != null)

        {

            Console.WriteLine($"Order for Table {tableNumber} marked as ready: {order.Orders.Length} items");

            order.IsReady = true;

            // Notify everyone in the table group (users + waiter connections added earlier)

            await Clients.Group(tableNumber.ToString()).ReceiveOrderReadyMessage(order, tableNumber);

            Console.WriteLine($"Order for Table {tableNumber} ready event broadcast to group.");

            var userOnTable = \_tables[tableNumber - 1].UserId;

            string userEmail = \_userCollection.Find(x => x.Id == userOnTable).FirstOrDefault()?.Email ?? string.Empty;

            string emailBody = GenerateEmailRecieptString(order);

            await \_emailService.SendEmailAsync(userEmail, $"Order for Table {tableNumber} at {DateTime.Now}", emailBody);

        }

        else

        {

            Console.WriteLine($"No order found for Table {tableNumber}");

        }

    }

    /// <summary>

    /// Handles the disconnection of a client from the server.

    /// </summary>

    public override async Task OnDisconnectedAsync(Exception? exception)

    {

        var sid = Context.ConnectionId;

        var httpContext = Context.GetHttpContext();

        if (\_userConnections.TryGetValue(sid, out string? userMongoId))

        {

            \_userConnections.TryRemove(sid, out \_);

            if (\_userids2sid.TryGetValue(userMongoId, out var sids))

            {

                sids.Remove(sid);

                if (sids.Count == 0) \_userids2sid.TryRemove(userMongoId, out \_);

            }

        }

        else if (\_waiterConnections.TryGetValue(sid, out string? waiterMongoId))

        {

            \_waiterConnections.TryRemove(sid, out \_);

            if (\_waiterids2sid.TryGetValue(waiterMongoId, out var sids))

            {

                sids.Remove(sid);

                if (sids.Count == 0) \_waiterids2sid.TryRemove(waiterMongoId, out \_);

            }

            // Clear waiter assignment on tables that reference this waiter

            \_tables.ForEach(t =>

            {

                if (t.WaiterId == waiterMongoId)

                {

                    t.WaiterId = string.Empty;

                }

            });

            Console.WriteLine("Waiter left all tables");

        }

        else if (\_ownerConnections.ContainsKey(sid))

        {

            \_ownerConnections.TryRemove(sid, out \_);

        }

        else

        {

            Console.WriteLine("Connection not found");

            return;

        }

        Console.WriteLine($"Disconnected: {sid}");

        await Clients.All.ReceiveTableLeaveMessage(\_tables);

        await base.OnDisconnectedAsync(exception);

    }

    /// <summary>

    /// Asynchronously fetches a list of quick messages from the message collection in the database.

    /// </summary>

    /// <returns>A task that represents the asynchronous operation, containing a list of QuickMessage objects.</returns>

    private async Task<List<QuickMessage>> FetchMessages()

    {

        var msgs = await \_messageCollection.Find(\_ => true).ToListAsync();

        return msgs;

    }

    /// <summary>

    /// Generates an HTML email receipt for a given order.

    /// </summary>

    /// <param name="order">The order containing details of the items purchased.</param>

    /// <returns>A string representing the HTML structure of the email receipt.</returns>

    /// <remarks>

    /// If the order is null or contains no items, a message indicating no orders are found is returned.

    /// The HTML includes a styled table listing each meal item with its quantity, price, and line total,

    /// along with the overall total price of the order. The currency is formatted in shekels (₪).

    /// </remarks>

    private string GenerateEmailRecieptString(Order order)

    {

        if (order == null || order.Orders == null || order.Orders.Length == 0)

        {

            return "No orders found.";

        }

        // Currency helper (₪). Switch if you need locale-based formatting.

        static string Money(decimal val) => $"{val:0.00} ₪";

        var sbRows = new StringBuilder();

        foreach (var line in order.Orders)

        {

            var name = WebUtility.HtmlEncode(line?.Meal?.MealName ?? "Item");

            var qty = line?.Quantity ?? 0;

            decimal price = 0m;

            try

            {

                if (line?.Meal != null)

                    price = Convert.ToDecimal(line.Meal.Price, CultureInfo.InvariantCulture);

            }

            catch { /\* fallback stays 0m \*/ }

            var lineTotal = price \* qty;

            sbRows.Append($@"

          <tr>

            <td class=""item"">{name}</td>

            <td class=""qty"">x{qty}</td>

            <td class=""price"">{Money(price)}</td>

            <td class=""line"">{Money(lineTotal)}</td>

          </tr>");

        }

        // Compute total if the Order.Total isn't reliable

        decimal computedTotal = 0m;

        try

        {

            computedTotal = order.Orders.Sum(p =>

            {

                decimal pprice = 0m;

                try

                {

                    if (p?.Meal != null)

                        pprice = Convert.ToDecimal(p.Meal.Price, CultureInfo.InvariantCulture);

                }

                catch { /\* 0m \*/ }

                return pprice \* (p?.Quantity ?? 0);

            });

        }

        catch { /\* leave as 0m \*/ }

        decimal total;

        try

        {

            total = order.Total > 0 ? Convert.ToDecimal(order.Total, CultureInfo.InvariantCulture) : computedTotal;

        }

        catch

        {

            total = computedTotal;

        }

        var safeTable = WebUtility.HtmlEncode(order.TableNumber.ToString(CultureInfo.InvariantCulture));

        var html = $@"

<!DOCTYPE html>

<html lang=""en"">

<head>

  <meta charset=""utf-8"" />

  <meta name=""viewport"" content=""width=device-width, initial-scale=1.0"" />

  <title>Your order is ready</title>

  <style>

    body {{

      margin: 0;

      padding: 0;

      background: #f6f6f6;

      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Helvetica, Arial, sans-serif;

      color: #222;

    }}

    .wrap {{

      max-width: 560px;

      margin: 24px auto;

      background: #ffffff;

      border-radius: 12px;

      overflow: hidden;

      border: 1px solid #eaeaea;

    }}

    .header {{

      background: #111827;

      color: #fff;

      padding: 16px 20px;

    }}

    .header h1 {{

      margin: 0;

      font-size: 18px;

      font-weight: 700;

      letter-spacing: 0.3px;

    }}

    .meta {{

      padding: 12px 20px 0 20px;

      font-size: 13px;

      color: #6b7280;

    }}

    .content {{

      padding: 12px 20px 20px 20px;

    }}

    .table {{

      width: 100%;

      border-collapse: collapse;

      margin-top: 10px;

      border: 1px solid #e5e7eb;

      border-radius: 8px;

      overflow: hidden;

    }}

    .table th {{

      background: #f3f4f6;

      text-align: left;

      padding: 10px;

      font-size: 13px;

      border-bottom: 1px solid #e5e7eb;

    }}

    .table td {{

      padding: 10px;

      font-size: 14px;

      border-bottom: 1px solid #f3f4f6;

      vertical-align: top;

    }}

    .item {{ width: 55%; }}

    .qty  {{ width: 10%; white-space: nowrap; }}

    .price{{ width: 15%; white-space: nowrap; text-align: right; }}

    .line {{ width: 20%; white-space: nowrap; text-align: right; }}

    .total-row {{

      margin-top: 16px;

      font-size: 16px;

      font-weight: 700;

      text-align: left;

    }}

    .footer {{

      padding: 14px 20px 18px 20px;

      font-size: 12px;

      color: #6b7280;

      border-top: 1px solid #f3f4f6;

    }}

    .badge {{

      display: inline-block;

      padding: 3px 8px;

      font-size: 12px;

      border-radius: 9999px;

      background: #16a34a;

      color: #fff;

      font-weight: 600;

      vertical-align: middle;

    }}

  </style>

</head>

<body>

  <div class=""wrap"">

    <div class=""header"">

      <h1>Your order is ready ✅</h1>

    </div>

    <div class=""meta"">

      <div>Table: <strong>#{safeTable}</strong></div>

      <div>Status: <span class=""badge"">READY</span></div>

    </div>

    <div class=""content"">

      <table class=""table"" role=""presentation"" cellspacing=""0"" cellpadding=""0"">

        <thead>

          <tr>

            <th>Item</th>

            <th>Qty</th>

            <th style=""text-align:right"">Price</th>

            <th style=""text-align:right"">Total For Meal</th>

          </tr>

        </thead>

        <tbody>

          {sbRows}

        </tbody>

      </table>

      <div class=""total-row"">Total: {Money(total)}</div>

    </div>

    <div class=""footer"">

      Thanks for dining with us! If you have any questions, reply to this email.

    </div>

  </div>

</body>

</html>";

        return html;

    }

}



namespace Server.Services;

/// <summary>

/// Request object for waiter login.

/// </summary>

public class WaiterLogin

{

    public string? Email { get; set; }

    public string? Password { get; set; }

}