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>

    );

}



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;



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%',

    },

})



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%",

    }

});



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',

    },

})



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",

    },

});



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;



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',

},

});

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',

},

});

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,

},

});

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,

},

});

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,

},

});

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(e){

alert(e);

}

}

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',

},

});

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',

},

});

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;

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 },

});



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',

    },

})



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 }} />

    </>

  );

}



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,

  },

});



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



// 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;

    }

  };

}