تقرير مشروع نظام إدارة المستشفيات (مدمج بهياكل البيانات والخوارزميات)

التاريخ: 22 يناير 2026

1. مقدمة عن المشروع

يمثل هذا التقرير توثيقاً شاملاً لمشروع نظام إدارة المستشفيات المطور بلغة جافا، بعد إدخال تعديلات جوهرية لدمج مجموعة متقدمة من هياكل البيانات والخوارزميات. كان الهدف من هذا التعديل هو تحويل المشروع من مجرد تطبيق لقاعدة بيانات إلى مشروع تعليمي متكامل يوضح كيفية تطبيق المفاهيم النظرية لهياكل البيانات والخوارزميات في سياق نظام عملي حقيقي، مع التركيز على الكفاءة والأداء.

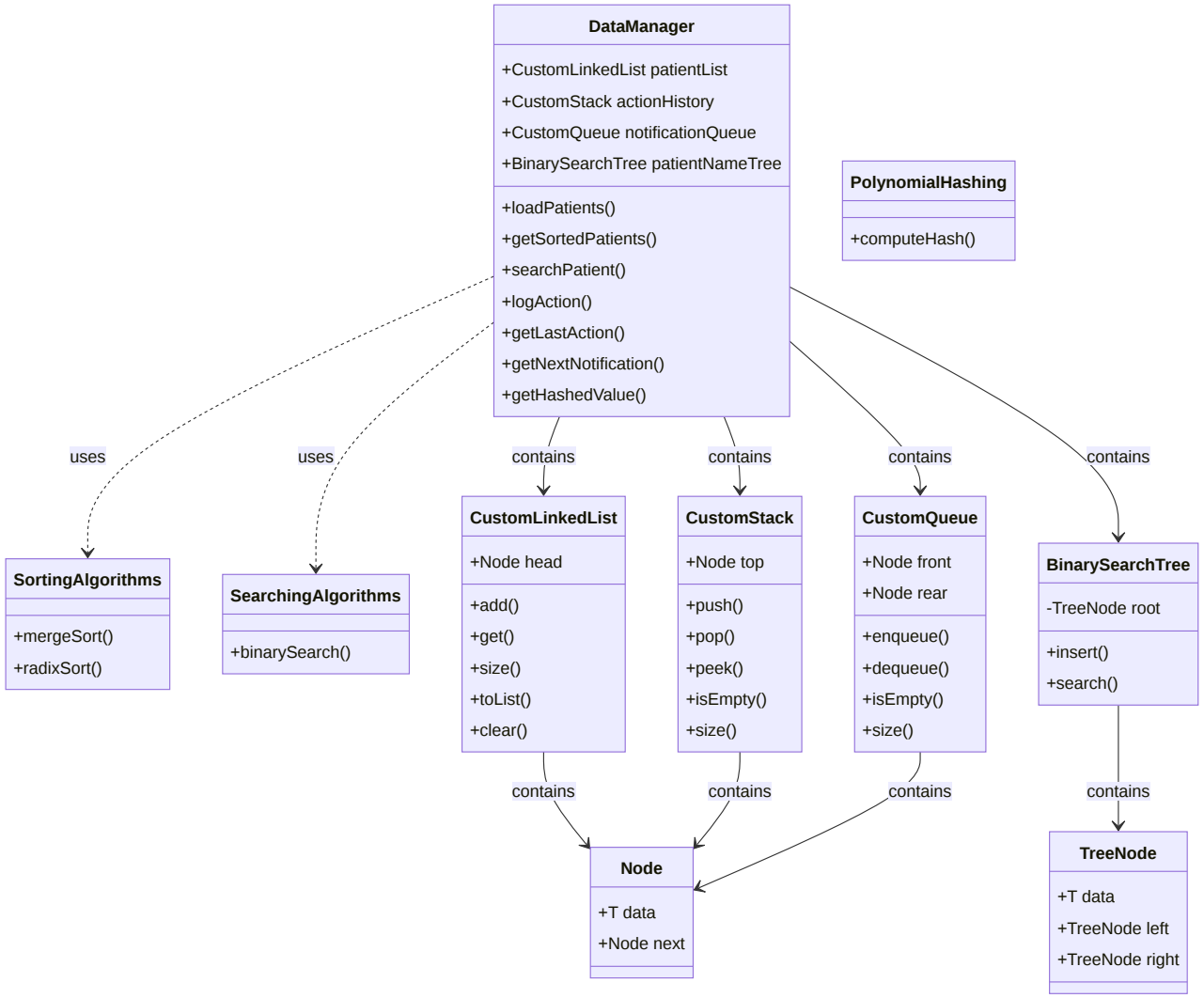
2. هيكل المشروع البرمجي (Software Project Structure)

تم تنظيم المشروع لضمان الفصل الواضح بين طبقات التطبيق المختلفة، مع إضافة حزمتين جديدتين لاستيعاب الكود المضاف الخاص بهياكل البيانات والخوارزميات:

الحزمة (Package)	الوصف	المكونات الرئيسية
`Auth`	إدارة عمليات تسجيل الدخول والتسجيل للمستخدمين.	`LoginPage.java`, `RegisterPage.java`
`Components`	مكونات واجهة المستخدم الرسومية (GUI) القابلة لإعادة الاستخدام، مثل الجداول والنماذج الديناميكية.	`Table.java`, `DynamicForm.java`, `PageHeader.java`
`DbConfig`	طبقة الوصول إلى البيانات والربط بقاعدة البيانات، بالإضافة إلى إدارة البيانات المحملة في الذاكرة.	`CrudManager.java`, `DbOperation.java`, `DbSetup.java`, **`DataManager.java`**
`Modules`	صفحات النظام الرئيسية التي تمثل واجهات المستخدم لإدارة الكيانات المختلفة (المرضى، الأطباء، الأدوية، المواعيد).	`PatientsPage.java`, `DoctorsPage.java`, `AppointmentsPage.java`, `MedicinesPage.java`
`config`	ملفات الإعدادات والثوابت العامة للمشروع، مثل الألوان والخطوط وأبعاد الشاشات.	`DesignConstants.java`, `GlobalConstants.java`, `GlobalTheming.java`
`common`	دوال مساعدة عامة تستخدم عبر المشروع، مثل تغيير حجم الصور.	`commonMethods.java`
`Main`	نقطة الدخول الرئيسية لتشغيل التطبيق.	`Main.java`
`DataStructures`	**جديد:** حزمة تحتوي على تطبيقات مخصصة لهياكل البيانات المطلوبة، مما يوفر مرونة وكفاءة في التعامل مع البيانات.	**`Node.java`**, **`CustomLinkedList.java`**, **`CustomStack.java`**, **`CustomQueue.java`**, **`BinarySearchTree.java`**, **`PolynomialHashing.java`**
`Algorithms`	**جديد:** حزمة تحتوي على تطبيقات الخوارزميات المطلوبة، مما يتيح معالجة البيانات بكفاءة عالية.	**`SortingAlgorithms.java`**, **`SearchingAlgorithms.java`**

مخطط الفئات البرمجية (Class Diagram)

يوضح المخطط التالي العلاقات بين الفئات البرمجية الرئيسية، مع التركيز على الفئات الجديدة التي تم إضافتها وكيفية تفاعلها مع `DataManager`:



3. هيكل قاعدة البيانات (Database Schema)

يعتمد المشروع على قاعدة بيانات علائقية لتخزين معلومات المستشفى. تم استنتاج هيكل الجداول والحقول من ملفات واجهة المستخدم التي تحدد `formStructure` و `COLUMNS` لكل كيان. يوضح الجدول التالي تفاصيل الجداول الرئيسية:

أ. جدول `users`

يحتوي على معلومات تسجيل الدخول للمستخدمين.

الحقل	النوع	الوصف
`id`	`INT`	مفتاح أساسي، معرف فريد للمستخدم.
`email`	`VARCHAR`	البريد الإلكتروني للمستخدم (يستخدم لتسجيل الدخول).
`password`	`VARCHAR`	كلمة المرور المشفرة للمستخدم.

ب. جدول `patients`

يحتوي على معلومات المرضى.

الحقل	النوع	الوصف
`id`	`INT`	مفتاح أساسي، معرف فريد للمريض.
`name`	`VARCHAR`	اسم المريض.
`date_of_birth`	`DATE`	تاريخ ميلاد المريض.
`gender`	`VARCHAR`	جنس المريض (ذكر/أنثى/أخرى).
`phone_number`	`VARCHAR`	رقم هاتف المريض.
`email`	`VARCHAR`	البريد الإلكتروني للمريض.
`address`	`VARCHAR`	عنوان المريض.
`diagnosis`	`VARCHAR`	تشخيص حالة المريض.
`treatment_status`	`VARCHAR`	حالة العلاج (جارٍ، مكتمل، معلق، متوقف).
`status`	`VARCHAR`	حالة المريض في النظام (نشط/غير نشط).

ج. جدول `doctors`

يحتوي على معلومات الأطباء.

الحقل	النوع	الوصف
`id`	`INT`	مفتاح أساسي، معرف فريد للطبيب.
`name`	`VARCHAR`	اسم الطبيب.
`specialty`	`VARCHAR`	تخصص الطبيب (قلب، أعصاب، عظام، أطفال، أخرى).
`phone_number`	`VARCHAR`	رقم هاتف الطبيب.
`email`	`VARCHAR`	البريد الإلكتروني للطبيب.
`gender`	`VARCHAR`	جنس الطبيب (ذكر/أنثى).
`years_of_experience`	`INT`	سنوات الخبرة للطبيب.
`status`	`VARCHAR`	حالة الطبيب في النظام (نشط/غير نشط).
`start_time`	`TIME`	وقت بدء دوام الطبيب.
`end_time`	`TIME`	وقت انتهاء دوام الطبيب.

د. جدول `appointments`

يحتوي على معلومات المواعيد.

الحقل	النوع	الوصف
`id`	`INT`	مفتاح أساسي، معرف فريد للموعد.
`patient_name`	`VARCHAR`	اسم المريض المرتبط بالموعد.
`doctor_name`	`VARCHAR`	اسم الطبيب المرتبط بالموعد.
`appointment_date`	`DATE`	تاريخ الموعد.
`appointment_time`	`TIME`	وقت الموعد.
`reason`	`VARCHAR`	سبب الموعد.
`status`	`VARCHAR`	حالة الموعد (مجدول، مكتمل، ملغى، لم يحضر).

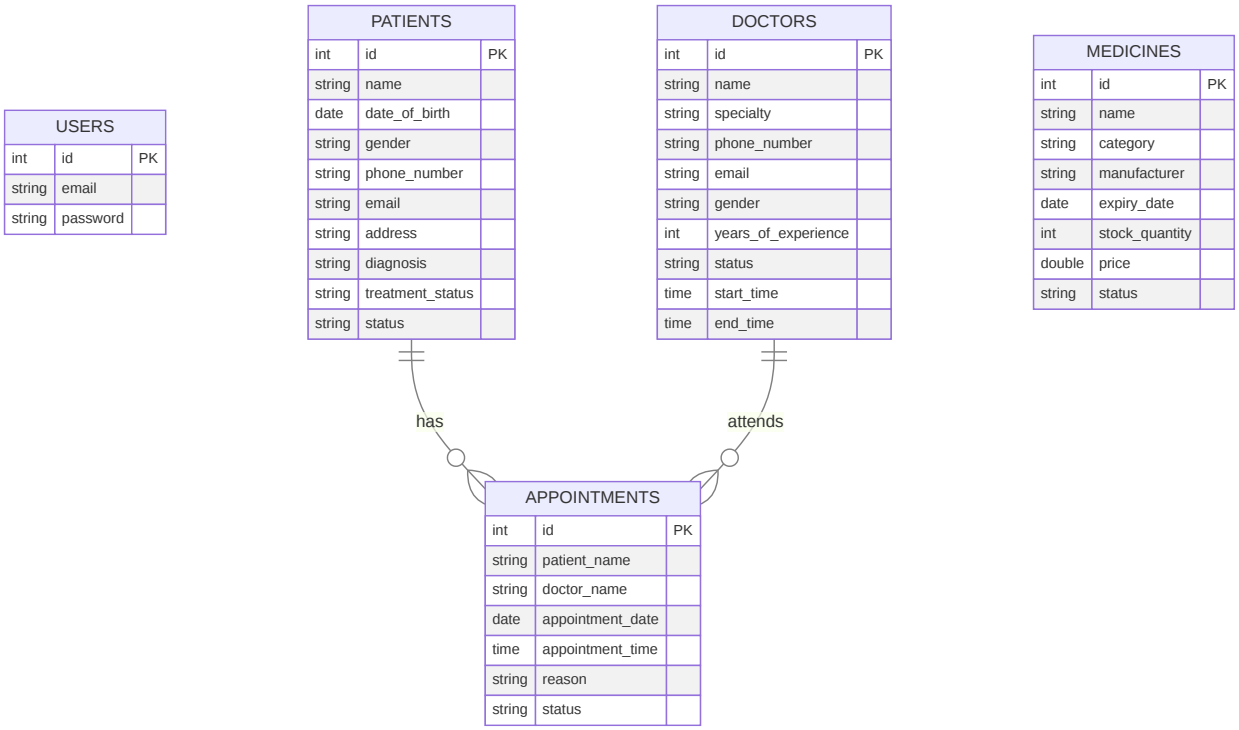
هـ. جدول `medicines`

يحتوي على معلومات الأدوية.

الحقل	النوع	الوصف
`id`	`INT`	مفتاح أساسي، معرف فريد للدواء.
`name`	`VARCHAR`	اسم الدواء.
`category`	`VARCHAR`	فئة الدواء.
`manufacturer`	`VARCHAR`	الشركة المصنعة للدواء.
`expiry_date`	`DATE`	تاريخ انتهاء صلاحية الدواء.
`stock_quantity`	`INT`	الكمية المتوفرة في المخزون.
`price`	`DOUBLE`	سعر الدواء.
`status`	`VARCHAR`	حالة الدواء (متوفر/غير متوفر).

مخطط علاقات الكيانات (ER Diagram)

يوضح المخطط التالي العلاقات بين الجداول الرئيسية في قاعدة البيانات:



4. الشرح التفصيلي لهياكل البيانات المضافة

تم تطبيق هياكل البيانات التالية لتعزيز كفاءة إدارة البيانات داخل النظام:

أ. Arrays (المصفوفات)

المفهوم: هي هياكل بيانات أساسية لتخزين مجموعة من العناصر من نفس النوع في مواقع ذاكرة متجاورة، مما يتيح الوصول المباشر والسريع للعناصر باستخدام الفهرس. **التطبيق في المشروع:** بشكل أساسي في المشروع لتمثيل البيانات (`Object[][]`) تُستخدم المصفوفات ثنائية الأبعاد قبل عرضها في الجداول. هذه `CrudManager.read()` المسترجعة من قاعدة البيانات بواسطة الصيغة هي التي يتم تمريرها إلى خوارزميات الترتيب والبحث (`SortingAlgorithms`, `SearchingAlgorithms`) للتعامل معها بكفاءة عالية.

ب. Linked List (القائمة المتصلة)

المفهوم: هيكل بيانات ديناميكي يتكون من عقد (Nodes)، حيث تحتوي كل عقدة على البيانات وعنوان العقدة التالية. تسمح هذه البنية بتغيير حجمها ديناميكياً وتسهل عمليات الإضافة والحذف في أي موقع. **التطبيق في المشروع** (**`DataStructures/CustomLinkedList.java`**):

- تُستخدم `DbConfig.DataManager` بعد تحميلها من قاعدة البيانات لتخزين سجلات المرضى في الذاكرة داخل كلاس.

- تسمح هذه البنية بتجنب قيود الحجم الثابت للمصفوفات وتسهل عمليات الإضافة والحذف المنطقية للمرضى في بيئة الذاكرة، مما يوفر مرونة أكبر في إدارة قائمة المرضى النشطين.

ج. Stack (المكدس)

المفهوم: هيكل بيانات يتبع مبدأ "الداخل أخيراً يخرج أولاً" (LIFO - Last In, First Out)، حيث يتم إضافة العناصر وحذفها من نفس الطرف (القمة). **التطبيق في المشروع (`DataStructures/CustomStack.java`):**

- **سجل الإجراءات (Action History)** يُستخدم لتسجيل داخل كلاس `DbConfig.DataManager`.
- عندما يقوم المستخدم بإجراء عملية مهمة (مثل إضافة مريض، تعديل بيانات طبيب، حذف موعد)، يتم "دفع" (`push`) وصف الإجراء إلى المكدس. يمكن استرجاع آخر إجراء تم تنفيذه بسهولة باستخدام عملية "نظرة خاطفة" (`peek`)، مما يوفر تتبعاً للإجراءات الحديثة.

د. Queue (الطابور)

المفهوم: هيكل بيانات يتبع مبدأ "الداخل أولاً يخرج أولاً" (FIFO - First In, First Out)، حيث يتم إضافة العناصر من طرف (الخلف) وحذفها من الطرف الآخر (الأمام). **التطبيق في المشروع (`DataStructures/CustomQueue.java`):**

- **قائمة الإشعارات (Notifications)** يُستخدم لإدارة داخل كلاس `DbConfig.DataManager`.
- يتم "إدخال" (`enqueue`) الإشعارات الجديدة إلى نهاية الطابور، ويتم "إخراج" (`dequeue`) الإشعارات القديمة من مقدمة الطابور لمعالجتها وعرضها للمستخدم بالترتيب الزمني الذي وصلت به، مما يضمن معالجة الإشعارات بشكل عادل ومنظم.

هـ. Hashing (التجزئة)

المفهوم: عملية تحويل البيانات المدخلة (مثل سلسلة نصية أو مفتاح) إلى قيمة رقمية قصيرة ذات طول ثابت (Hash Value) باستخدام دالة تجزئة. تُستخدم بشكل شائع في جداول التجزئة للبحث السريع وفي التشفير. **التطبيق في المشروع (`DataStructures/PolynomialHashing.java`):**

- تم تطبيق دالة **Polynomial Hashing** كنموذج تعليمي يوضح كيفية إنشاء دالة تجزئة فعالة.
- لحساب قيمة التجزئة (`Auth/LoginPage.java`) تم دمجها في صفحة تسجيل الدخول لكلمة المرور المدخلة وعرضها في سجل النظام (System Console). هذا يوضح مبدأ

استخدام التجزئة في تمثيل كلمات المرور (عادةً ما يتم تخزين التجزئة بدلاً من كلمة المرور الأصلية لأسباب أمنية) أو في التحقق من سلامة البيانات.

g. Trees (الأشجار)

المفهوم: هيكل بيانات غير خطي يتكون من عقد متصلة، حيث يتميز كل عنصر بأنه إما جذر أو ابن لعنصر آخر. **التطبيق في المشروع (`DataStructures/BinarySearchTree.java`):**

- تم تطبيق **شجرة البحث الثنائية (BST - Binary Search Tree)** لتخزين أسماء المرضى.
- عند تحميل بيانات المرضى، يتم إدخال أسماء المرضى في هذه الشجرة. تسمح هذه البنية بإجراء عمليات بحث وإضافة وحذف بكفاءة عالية (O(log n) في المتوسط)، مما يعزز سرعة التحقق من وجود اسم مريض معين في النظام أو استرجاع معلوماتهم بناءً على الاسم.

5. الشرح التفصيلي للخوارزميات المضافة

تم تطبيق الخوارزميات التالية لتمكين عمليات ترتيب وبحث سريعة وفعالة على بيانات النظام، مما يحسن من تجربة المستخدم وأداء التطبيق:

أ. Merge Sort (الترتيب بالدمج)

المفهوم: خوارزمية ترتيب تعتمد على مبدأ "فرق تسد" (Divide and Conquer). تقوم بتقسيم المصفوفة بشكل متكرر إلى نصفين حتى تصل إلى عناصر فردية، ثم تقوم بدمج هذه الأجزاء المرتبة في جميع الحالات (أسوأ، متوسط، أفضل). تتميز بثبات أدائها (O(n log n)) لإنشاء مصفوفة مرتبة بالكامل. **التطبيق في المشروع (`Algorithms/SortingAlgorithms.java`):**

- تُستخدم كخوارزمية ترتيب عامة لترتيب سجلات المرضى (أو أي بيانات جدول أخرى) حسب أي عمود نصي أو رقمي (مثل الاسم، تاريخ الميلاد، التخصص).
- لتقسيم المصفوفة إلى نصفين **الاستدعاء الذاتي (Recursion)** تعتمد بشكل أساسي على بشكل متكرر قبل عملية الدمج.

ب. Radix Sort (الترتيب الجذري)

المفهوم: خوارزمية ترتيب غير مقارنة (Non-Comparison Sort) تعمل عن طريق معالجة الأرقام من خلال أرقامها الفردية (أو خاناتها). وهي فعالة جداً لترتيب الأعداد الصحيحة أو السلاسل النصية ذات الطول الثابت. **التطبيق في المشروع (`Algorithms/SortingAlgorithms.java`):**

- **الهوية** تُستخدم لترتيب سجلات المرضى (أو الأطباء، المواعيد، الأدوية) حسب **(ID)**، حيث أن الهوية هي عدد صحيح.
- هو عدد الخانات القصوى، مما يجعلها k هو عدد العناصر و n حيث ،O(nk)) توفر كفاءة عالية أسرع بكثير من خوارزميات الترتيب المقارنة عند التعامل مع بيانات الهوية الرقمية الكبيرة.

ج. Binary Search (البحث الثنائي)

المفهوم: خوارزمية بحث سريعة وفعالة تعمل على مصفوفات مرتبة. تقوم بتقسيم مجال البحث إلى نصفين في كل خطوة، مما يقلل بشكل كبير من عدد المقارنات المطلوبة للعثور على العنصر المستهدف. كفاءتها الزمنية هي O(log n). **التطبيق في المشروع** (`Algorithms/SearchingAlgorithms.java`):

- لتمكين المستخدم من البحث (`Components/Table.java`) تم دمجها في واجهة الجدول عن مريض معين باستخدام الهوية (ID).
- على عمود Radix Sort باستخدام) قبل إجراء البحث الثنائي، يتم التأكد من أن البيانات مرتبة لتقليل مجال البحث، مما يضمن **(Recursion)** تعتمد الخوارزمية على **الاستدعاء الذاتي** .(ID سرعة بحث فائقة حتى في مجموعات البيانات الكبيرة.

د. Recursion (الاستدعاء الذاتي)

المفهوم: تقنية برمجية قوية تقوم فيها الدالة باستدعاء نفسها لحل مشكلة أكبر عن طريق تقسيمها إلى نسخ أصغر من نفس المشكلة حتى تصل إلى حالة أساسية يمكن حلها مباشرة.
التطبيق في المشروع:

- **Merge Sort:** تُستخدم لتقسيم المصفوفة بشكل متكرر إلى أجزاء أصغر قبل دمجها.
- **Binary Search:** تُستخدم لتقسيم مجال البحث بشكل متكرر في كل خطوة.
- هذا الاستخدام يوضح كيفية تصميم خوارزميات فعالة لحل المشكلات المعقدة بطريقة أنيقة وموجزة.

6. طبقة إدارة البيانات (DataManager)

ليكون بمثابة العمود الفقري لدمج هياكل `DbConfig.DataManager.java` تم إنشاء كلاس البيانات والخوارزميات مع بقية النظام. يعمل هذا الكلاس كطبقة تجريد بين واجهة المستخدم وقاعدة البيانات، ويدير البيانات في الذاكرة باستخدام هياكل البيانات الجديدة.

الوظيفة (Method)	الغرض	هياكل البيانات/الخوارزميات المستخدمة
`loadPatients()`	تحميل بيانات المرضى من قاعدة البيانات إلى الذاكرة، وتعبئة هياكل البيانات الداخلية.	`CustomLinkedList`, `BinarySearchTree`
`getSortedPatients()`	استرجاع قائمة المرضى مرتبة بناءً على معيار معين (مثل ID أو الاسم).	`Merge Sort`, `Radix Sort`
`searchPatient()`	البحث عن مريض معين في البيانات المحملة.	`Binary Search`
`logAction()`	تسجيل إجراءات المستخدم لتتبع النشاط.	`CustomStack`, `CustomQueue`
`getLastAction()`	استرجاع آخر إجراء تم تسجيله.	`CustomStack`
`getNextNotification()`	استرجاع الإشعار التالي في قائمة الانتظار.	`CustomQueue`
`getHashedValue()`	حساب قيمة التجزئة لسلسلة نصية.	`PolynomialHashing`

7. التكامل مع واجهة المستخدم (UI Integration)

ليعكس التكامل مع هياكل البيانات والخوارزميات `Components.Table.java` تم تعديل كلاس الجديدة:

- **تحميل البيانات:** عند تهيئة الجدول الخاص بالمرضى، يقوم `Table` باستدعاء `DataManager.loadPatients()` لتحميل البيانات إلى الذاكرة.
- **الترتيب الافتراضي:** يتم عرض بيانات المرضى مرتبة افتراضياً حسب `ID` باستخدام `Radix Sort` عبر `DataManager.getSortedPatients()`.
- **وظيفة البحث:** تم إضافة حقل بحث وزر "Search ID (Binary)" في واجهة الجدول. عند إدخال `ID` والضغط على الزر، يتم استخدام `SearchingAlgorithms.binarySearch()` (بعد التأكد من ترتيب البيانات) للعثور على المريض وتحديد صفه في الجدول.
- **تسجيل الدخول:** تم إضافة استدعاء لـ `Auth.LoginPage.java`، في `PolynomialHashing.computeHash()` لتوضيح كيفية استخدام دالة التجزئة مع كلمات المرور.