A MAJOR PROJECT REPORT

ON

A.I. CHATBOTS FOR ADMISSION COUSELLING

SUBMITTED
BY

HITESHWAR SINGH (20104040)
&
GULSHAN (20104034)
&
DIVYA GUPTA (20104030)
&
DAKSH KHATKAR (20104026)

B.TECH VIII SEM (ECE)

UNDER THE SUPERVISION
OF
Dr. Arun K Khosla
Professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
Dr. B. R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY,
JALANDHAR

CANDIDATES'S DECLARATION

I hereby declared that the work which is being presented in the major project report entitled, "***A.I. Chatbots for Admission Counselling***" being submitted by us in the partial fulfillment of the requirement for the award of degree of ***Bachelor of Technology in Electronics and Communication Engineering*** to ***Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY, JALANDHAR*** is an authentic record of our own work carried out under guidance of ***Dr. Arun K Khosla,*** Department of Electronics & Communication Engineering.

The matters embodied in this record have not been submitted by us for the award of any other degree or diploma.

DATE: 28/05/2024

Hiteshwar Singh (20104040)
Gulshan (20104034)
Divya Gupta (20104030)
Daksh Khatkar (20104026)

<u>CERTIFICATE</u>

This is to certify that the major project work entitled, ***"A.I. Chatbots for Admission Counselling"*** submitted by ***Daksh Khatkar (20104026)*** in partial fulfillment for the award of degree of ***Bachelor of Technology in Electronics and Communication Engineering*** to ***Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY, JALANDHAR*** has been carried out under my supervision. This work has not been submitted partially or in full to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor ……………………..

Name of Supervisor ……………………..

Designation ……………………..

Date ……………………..

ACKNOWLEDGEMENT

We would like to extend our deepest gratitude to those who have supported and guided us throughout the development of this project.

First and foremost, we express our sincere appreciation to Dr. Arun K Khosla, our professor, for their unwavering support, insightful feedback, and invaluable guidance. Their expertise and encouragement have been instrumental in the successful completion of this project. Their dedication to fostering a positive and intellectually stimulating environment has inspired us to strive for excellence.

We are also profoundly grateful to the members of the cohort team who will be judging our project. Your commitment to evaluating our work and providing constructive feedback is greatly appreciated. Your expertise and insights are crucial to our growth and learning, and we look forward to your evaluation and suggestions.

Finally, we wish to thank our peers and fellow team members, without whom this project would not have been possible. Your collaboration, dedication, and hard work have been exemplary, and it has been a privilege to work alongside such a talented and motivated group.

Thank you all for your contributions and support.

Sincerely,
Hiteshwar Singh
Gulshan
Divya Gupta
Daksh Khatkar

ABSTRACT

In the rapidly evolving landscape of educational technology, AI-driven chatbots are emerging as vital tools in enhancing student support services.

Our project explores the development and implementation of an AI chatbot designed specifically for admission counseling. This intelligent system aims to streamline the admission process by providing prospective students with personalized guidance, real-time information, and prompt responses to their queries.

The chatbot leverages advanced natural language processing (NLP) techniques to understand and interact with users, ensuring accurate and contextually relevant communication. It is equipped to handle a wide range of inquiries, from general admission requirements to detailed program-specific information. Additionally, the chatbot offers features such as application status tracking, deadline reminders, and document submission guidance, thereby reducing the administrative burden on human counselors and improving the overall efficiency of the admission process.

Our implementation involved rigorous testing and refinement to ensure the chatbot's reliability, user-friendliness, and effectiveness. Preliminary results indicate a significant enhancement in user satisfaction and engagement, suggesting that AI chatbots can play a crucial role in transforming traditional admission counseling services. This report details the development process, key features, challenges encountered, and the potential impact of AI chatbots in educational institutions.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Introduction

Project Overview

In recent years, the integration of artificial intelligence (AI) into various sectors has revolutionized how services are delivered. One such innovative application is in the domain of education, particularly in student admission counselling. Our project aims to harness the power of AI chatbots to streamline and enhance the admission counselling process for students. By leveraging advanced technologies such as the Gemini API and modern development frameworks like Kotlin Multiplatform and Jetpack Compose, we have developed a sophisticated and user-friendly chatbot tailored to meet the needs of prospective students.

Motivation and Objectives

The admission process can often be daunting and overwhelming for students due to the plethora of information available and the complexity of application procedures. Traditional methods of counselling, while effective, may not always be accessible or efficient. The motivation behind our project is to provide an accessible, reliable, and efficient solution that can assist students in navigating the admission process with ease.

Our primary objectives are:

1. To develop an AI chatbot that can answer frequently asked questions, provide information about various courses and institutions, and guide students through the application process.
2. To ensure the chatbot is easily accessible across different platforms, enhancing its usability and reach.
3. To incorporate a user-friendly interface that simplifies interactions and improves the overall user experience.

Technologies Used

To achieve these objectives, our project utilizes the following technologies:

Gemini API

The Gemini API serves as the backbone of our AI chatbot, providing robust and advanced AI capabilities. It enables the chatbot to understand and process natural language inputs, ensuring accurate and contextually appropriate responses.

Kotlin Multiplatform

Kotlin Multiplatform allows us to write code that can run on multiple platforms (Android, iOS, web) with minimal modifications. This ensures that our chatbot is accessible to a wide range of users, regardless of the device or operating system they are using.

Jetpack Compose

Jetpack Compose is used for building the user interface of the chatbot. Its declarative approach to UI development allows us to create a responsive and dynamic interface that enhances the user experience. Jetpack Compose's integration with Kotlin ensures seamless development and better performance.

Expected Impact

By integrating an AI-driven chatbot into the admission counseling process, we anticipate a significant improvement in the accessibility and efficiency of the service. Students will be able to obtain instant answers to their queries, receive personalized guidance, and navigate the admission process more confidently. Additionally, educational institutions can benefit from reduced workload on their counseling staff and improved engagement with prospective students.

In conclusion, our project represents a step forward in leveraging AI to address real-world challenges in the education sector. We believe that our chatbot will not only facilitate the admission process for students but also set a precedent for future applications of AI in education.

Literature Review

Drawing from extensive systematic literature reviews, as summarized in Table 1, AI chatbots possess the potential to profoundly influence diverse aspects of education. They contribute to advancements in both teaching and learning processes. However, it is essential to address concerns regarding the irrational use of technology and the challenges that education systems encounter while striving to harness its capacity and make the best use of it.

Title (citation)	Summary of main Findings
A Meta-Analysis and Systematic Review of the Effect of Chatbot Technology Use in Sustainable Education (Deng & Yu, 2023)	Chatbot technology demonstrated a substantial impact on overall learning outcomes, regardless of factors such as intervention duration and learning content When it comes to specific aspects of learning, chatbots exhibited significant enhancements in knowledge retention The research revealed nonsignificant effect of chatbots on critical thinking, learning engagement, and motivation
Interacting with educational chatbots: A systematic review (Kuhail et al., 2023)	Chatbots are predominantly developed on web platforms to teach various subjects Most chatbots stick to preplanned conversation paths. Some of them utilize personalized learning methods catered to individual student requirements. Some chatbots integrate experiential and collaborative learning, along with other design principles Developing chatbots faces challenges and limitations, including a lack of adequate training datasets
Artificial Intelligence in Education: A Review (Krstić et al., 2022)	AI revolutionizes the methods of instruction employed by educators and the approaches to knowledge acquisition adopted by students The integration of AI into educational systems leads to the advancement of teaching and learning programs AI is bringing about significant transformations in the field of education, while recognizing that it will not entirely supplant the conventional education system
Chatbots applications in education: A systematic review (Okonkwo & Ade-Ibijola, 2021)	Chatbots are used in education for teaching, administration, assessment, advisory, and research Chatbots have the potential to enhance learning outcomes

Title (citation)	Summary of main Findings
	through immediate assistance, quick access to information, and increased student motivation Implementing Chatbot technology in education faces challenges like ethics, evaluation methods, user attitudes, programming complexities, and data integration issues
Are we there yet?-a systematic literature review on chatbots in education (Wollny et al., 2021)	Chatbots possess the capacity to serve as a valuable educational tool, capable of enhancing student learning outcomes across various domains such as knowledge acquisition, problem-solving skills, self-regulation, and motivation In the realm of education, chatbots have been employed for diverse purposes, including the dissemination of information, responding to inquiries, offering feedback, providing tutoring support, gamifying the learning experience, and facilitating social interaction Incorporating chatbots into education comes with challenges that need to be addressed, including the development of high-quality chatbot systems, the establishment of effective evaluation methods, and the resolution of ethical concerns associated with their use
AI in Education: A Systematic Literature Review (Tahiru, 2021)	AI presents extensive opportunities in education for teachers and students to harness and attain its necessary advantages AI is already being implemented in education and various other sectors in developed nations There are challenges involved in adopting AI in education, including ethical concerns, technological factors such as data usage and ownership, and organizational considerations like the potential replacement of humans by AI
Artificial Intelligence in Education: A Review (Chen et al., 2020)	AI Chatbots fulfill instructors' duties and responsibilities, acting as virtual assistants AI Chatbots enhance the effectiveness and efficiency of teachers, leading to a higher quality of instruction AI Chatbots offer students enhanced learning experiences, contributing to improved educational outcomes
Rediscovering the use of chatbots in education: A systematic	Chatbots are increasingly used in education for personalized tutoring, homework help, concept learning, test preparation,

Title (citation)	Summary of main Findings
literature review (Pérez et al., 2020)	collaboration, and mental health support Incorporating chatbots in education benefits student engagement, learning outcomes, and stress reduction Challenges in implementing chatbots in education include the need for accurate information, potential bias, and ensuring user acceptance
Artificial intelligence in higher education: the state of the field (Crompton & Burke, 2023)	AI has the potential to make education more accessible, affordable, and effective AI is being used to personalize learning, provide support to students, and automate administrative tasks. AI is also being used to improve research and create new educational opportunities Challenges associated with AI in education include the need for data privacy and security, the potential for bias, and the need for educators to be trained in how to use AI effectively

It is evident that chatbot technology has a significant impact on overall learning outcomes. Specifically, chatbots have demonstrated significant enhancements in learning achievement, explicit reasoning, and knowledge retention. The integration of chatbots in education offers benefits such as immediate assistance, quick access to information, enhanced learning outcomes, and improved educational experiences. However, there have been contradictory findings related to critical thinking, learning engagement, and motivation. Deng and Yu (2023) found that chatbots had a significant and positive influence on numerous learning-related aspects but they do not significantly improve motivation among students. Contrary, Okonkwo and Ade-Ibijola (Okonkwo & Ade-Ibijola, 2021), as well as (Wollny et al., 2021) find that using chatbots increases students' motivation.

In terms of application, chatbots are primarily used in education to teach various subjects, including but not limited to mathematics, computer science, foreign languages, and engineering. While many chatbots follow predetermined conversational paths, some employ personalized learning approaches tailored to individual student needs, incorporating experiential and collaborative learning principles. Challenges in chatbot development include insufficient training datasets, a lack of emphasis on usability heuristics, ethical concerns, evaluation methods, user attitudes, programming complexities, and data integration issues.

Although existing systematic reviews have provided valuable insights into the impact of chatbot technology in education, it's essential to acknowledge that the field of chatbot development is continually emerging and requires timely, and updated analysis to ensure that the information and assessments reflect the most recent advancements, trends, or developments in chatbot technology. The latest chatbot models have showcased remarkable capabilities in natural language processing and generation. Additional research is required to investigate the role and potential of these newer chatbots in the field of education. Therefore, our paper focuses on reviewing and discussing the findings of these new-generation chatbots' use in education, including their benefits and challenges from the perspectives of both educators and students.

There are a few aspects that appear to be missing from the existing literature reviews:

(a) The existing findings focus on the immediate impact of chatbot usage on learning outcomes. Further research may delve into the enduring impacts of integrating chatbots in education, aiming to assess their sustainability and the persistence of the observed advantages over the long term.

(b) The studies primarily discuss the impact of chatbots on learning outcomes as a whole, without delving into the potential variations based on student characteristics. Investigating how different student groups, such as age, prior knowledge, and learning styles, interact with chatbot technology could provide valuable insights.

(c) Although the studies highlight the enhancements in certain learning components, further investigation could explore the specific pedagogical strategies employed by chatbots to achieve these outcomes. Understanding the underlying mechanisms and instructional approaches utilized by chatbots can guide the development of more effective and targeted educational interventions. (d) While some studies touch upon user attitudes and acceptance, further research can delve deeper into the user experience of interacting with chatbots in educational settings. This includes exploring factors such as usability, perceived usefulness, satisfaction, and preferences of students and teachers when using chatbot technology.

Addressing these gaps in the existing literature would significantly benefit the field of education. Firstly, further research on the impacts of integrating chatbots can shed light on their long-term sustainability and how their advantages persist over time. This knowledge is crucial for educators and policymakers to make informed decisions about the continued integration of chatbots into educational systems. Secondly, understanding how different student characteristics interact with chatbot technology can help tailor educational interventions to individual needs, potentially optimizing the learning experience. Thirdly, exploring the specific pedagogical strategies employed by chatbots to enhance learning components can inform the development of more effective educational tools and methods. Lastly, a deeper exploration of the user experience with chatbots, encompassing usability, satisfaction, and preferences, can provide valuable insights into enhancing user engagement and overall satisfaction, thus guiding the future design and implementation of chatbot technology in education.

Abstract

AI chatbots shook the world not long ago with their potential to revolutionize education systems in a myriad of ways. AI chatbots can provide immediate support by answering questions, offering explanations, and providing additional resources. Chatbots can also act as virtual teaching assistants, supporting educators through various means. In this paper, we try to understand the full benefits of AI chatbots in education, their opportunities, challenges, potential limitations, concerns, and prospects of using AI chatbots in educational settings. We conducted an extensive search across various academic databases, and after applying specific predefined criteria, we selected a final set of 67 relevant studies for review. The research findings emphasize the numerous benefits of integrating AI chatbots in education, as seen from both students' and educators' perspectives. We found that students primarily gain from AI-powered chatbots in three key areas: homework and study assistance, a personalized learning experience, and the development of various skills. For educators, the main advantages are the time-saving assistance and improved pedagogy. However, our research also emphasizes significant challenges and critical factors that educators need to handle diligently. These include

concerns related to AI applications such as reliability, accuracy, and ethical considerations.

Introduction

The traditional education system faces several issues, including overcrowded classrooms, a lack of personalized attention for students, varying learning paces and styles, and the struggle to keep up with the fast-paced evolution of technology and information. As the educational landscape continues to evolve, the rise of AI-powered chatbots emerges as a promising solution to effectively address some of these issues. Some educational institutions are increasingly turning to AI-powered chatbots, recognizing their relevance, while others are more cautious and do not rush to adopt them in modern educational settings. Consequently, a substantial body of academic literature is dedicated to investigating the role of AI chatbots in education, their potential benefits, and threats. AI-powered chatbots are designed to mimic human conversation using text or voice interaction, providing information in a conversational manner. Chatbots' history dates back to the 1960s and over the decades chatbots have evolved significantly, driven by advancements in technology and the growing demand for automated communication systems. Created by Joseph Weizenbaum at MIT in 1966, ELIZA was one of the earliest chatbot programs (Weizenbaum, 1966). ELIZA could mimic human-like responses by reflecting user inputs as questions. Another early example of a chatbot was PARRY, implemented in 1972 by psychiatrist Kenneth Colby at Stanford University (Colby, 1981). PARRY was a chatbot designed to simulate a paranoid patient with schizophrenia. It engaged in text-based conversations and demonstrated the ability to exhibit delusional behavior, offering insights into natural language processing and AI. Developed by Richard Wallace in 1995, ALICE (Artificial Linguistic Internet Computer Entity) was an early example of a chatbot using natural language processing techniques that won the Loebner Prize Turing Test in 2000–2001 (Wallace, 1995), which challenged chatbots to convincingly simulate human-like conversation. Later in 2001 ActiveBuddy, Inc. developed the chatbot SmarterChild that operated on instant messaging platforms such as AOL Instant Messenger and MSN Messenger (Hoffer et al., 2001). SmarterChild was a chatbot that could carry on conversations with users about a variety of topics. It was also able to learn from its interactions with users, which made it more and more sophisticated over time. In 2011 Apple introduced Siri as a voice-activated personal assistant for its iPhone (Aron, 2011). Although not strictly a chatbot, Siri showcased the potential of conversational AI by understanding and responding to voice commands, performing tasks, and providing information. In the same year, IBM's Watson gained fame by defeating human champions in the quiz show Jeopardy (Lally & Fodor, 2011). It demonstrated the power of natural language processing and machine learning algorithms in understanding complex questions and providing accurate answers. More recently, in 2016, Facebook opened its Messenger platform for chatbot development, allowing businesses to create AI-powered conversational agents to interact with users. This led to an explosion of chatbots on the platform, enabling tasks like customer support, news delivery, and e-commerce (Holotescu, 2016). Google Duplex, introduced in May 2018, was able to make phone calls and carry out conversations on behalf of users. It showcased the potential of chatbots to handle complex, real-time interactions in a human-like manner (Dinh & Thai, 2018; Kietzmann et al., 2018).

More recently, more sophisticated and capable chatbots amazed the world with their abilities. Among them, ChatGPT and Google Bard are among the most profound AI-powered chatbots. ChatGPT is an artificial intelligence chatbot developed by OpenAI. It was first announced in November 2022 and is available to the general public. ChatGPT’s rival Google Bard chatbot, developed by Google AI, was first announced in May 2023. Both Google Bard and ChatGPT are sizable language model chatbots that undergo training on extensive datasets of text and code. They possess the ability to generate text, create diverse creative content, and provide informative answers to questions, although their accuracy may not always be perfect. The key difference is that Google Bard is trained on a dataset that includes text from the internet, while ChatGPT is trained on a dataset that includes text from books and articles. This means that Google Bard is more likely to be up-to-date on current events, while ChatGPT is more likely to be accurate in its responses to factual questions (AlZubi et al., 2022; Rahaman et al., 2023; Rudolph et al., 2023).

Chatbots are now used across various sectors, including education. Most of the latest intelligent AI chatbots are web-based platforms that adapt to the behaviors of both instructors and learners, enhancing the educational experience (Chassignol et al., 2018; Devedzic, 2004; Kahraman et al., 2010; Peredo et al., 2011). AI chatbots have been applied in both instruction and learning within the education sector. Chatbots specialize in personalized tutoring, homework help, concept learning, standardized test preparation, discussion and collaboration, and mental health support. Some of the most popular AI-based tools /chatbots used in education are:

Bard, introduced in 2022, is a large language model chatbot created by Google AI. Its capabilities include generating text, language translation, producing various types of creative content, and providing informative responses to questions. (Rudolph et al., 2023). Bard is still under development, but it has the potential to be a valuable tool for education.

ChatGPT, launched in 2022 by OpenAI, is a large language model chatbot that can generate text, produce diverse creative content, and deliver informative answers to questions (Dergaa et al., 2023; Khademi, 2023; Rudolph et al., 2023). However, as discussed in the results section of this paper, there are numerous concerns related to the use of ChatGPT in education, such as accuracy, reliability, ethical issues, etc.

Ada, launched in 2017, is a chatbot that is used to provide personalized tutoring to students. It can answer questions, provide feedback, and facilitate individualized learning for students (Kabiljo et al., 2020; Konecki et al., 2023). However, the Ada chatbot has limitations in understanding complex queries. It could misinterpret context and provide inaccurate responses

Replika, launched in 2017, is an AI chatbot platform that is designed to be a friend and companion for students. It can listen to students' problems, offer advice, and help them feel less alone (Pentina et al., 2023; Xie & Pentina, 2022). However, given the personal nature of conversations with Replika, there are valid concerns regarding data privacy and security.

Socratic, launched in 2013, had the goal of creating a community that made learning accessible to all students. Currently, Socratic is an AI-powered educational platform that was acquired by Google in 2018. While not a chatbot per se, it has a chatbot-like interface and functionality designed to assist students in learning new concepts (Alsanousi et al., 2023; Moppel, 2018; St-Hilaire et al., 2022). Like with other chatbots, a concern arises where students might excessively rely on Socratic for learning. This could lead to a diminished emphasis on critical thinking, as students may opt to use the platform to obtain answers without gaining a genuine understanding of the underlying concepts.

Habitica, launched in 2013, is used to help students develop good study habits. It gamifies the learning process, making it more fun and engaging for students. Students can use Habitica to manage their academic tasks, assignments, and study schedules. By turning their to-do list into a game-like experience, students are motivated to complete their tasks and build productive habits (Sales & Antunes, 2021; Zhang, 2023). However, the gamified nature of Habitica could inadvertently introduce distractions, especially for students who are easily drawn into the gaming aspect rather than focusing on their actual academic responsibilities.

Piazza launched in 2009, is used to facilitate discussion and collaboration in educational settings, particularly in classrooms and academic institutions. It provides a space for students and instructors to engage in discussions, ask questions, and share information related to course content and assignments (Ruthotto et al., 2020; Wang et al., 2020). Because discussions on Piazza are user-generated, the quality and accuracy of responses can vary. This variability may result in situations where students do not receive accurate and helpful information.

We will likely see even more widespread adoption of chatbots in education in the years to come as technology advances further. Chatbots have enormous potential to improve teaching and learning. A large body of literature is devoted to exploring the role, challenges, and opportunities of chatbots in education. This paper gathers and synthesizes this vast amount of literature, providing a comprehensive understanding of the current research status concerning the influence of chatbots in education. By conducting a systematic review, we seek to identify common themes, trends, and patterns in the impact of chatbots on education and provide a holistic view of the research, enabling researchers, policymakers, and educators to make evidence-based decisions. One of the main objectives of this paper is to identify existing research gaps in the literature to pinpoint areas where further investigation is needed, enabling researchers to contribute to the knowledge base and guide future research efforts. Firstly, we aim to understand the primary advantages of incorporating AI chatbots in education, focusing on the perspectives of students. Secondly, we seek to explore the key advantages of integrating AI chatbots from the standpoint of educators. Lastly, we endeavor to comprehensively analyze the major concerns expressed by scholars regarding the integration of AI chatbots in educational settings. Corresponding research questions are formulated in the section below. Addressing these research questions, we aim to contribute valuable insights that shed light on the potential benefits and challenges associated with the utilization of AI chatbots in the field of education.

The paper follows a structured outline comprising several sections. Initially, we provide a summary of existing literature reviews. Subsequently, we delve into the methodology, encompassing aspects such as research questions, the search process, inclusion and exclusion criteria, as well as the data extraction strategy. Moving on, we present a comprehensive analysis of the results in the subsequent section. Finally, we conclude by addressing the limitations encountered during the study and offering insights into potential future research directions.

Methodology

A systematic review follows a rigorous methodology, including predefined search criteria and systematic screening processes, to ensure the inclusion of relevant studies. This comprehensive approach ensures that a wide range of research is considered, minimizing the risk of bias and providing a comprehensive overview of the impact of AI in education. Firstly, we define the research questions and corresponding search strategies and then we filter the search results based on predefined inclusion and exclusion criteria. Secondly, we study selected articles and synthesize results and lastly, we report and discuss the findings. To improve the clarity of the discussion section, we employed Large Language Model (LLM) for stylistic suggestions.

Research questions

Considering the limitations observed in previous literature reviews, we have developed three research questions for further investigation:

What are the key advantages of incorporating AI chatbots in education from the viewpoint of students?

What are the key advantages of integrating AI chatbots in education from the viewpoint of educators?
What are the main concerns raised by scholars regarding the integration of AI chatbots in education?
Exploring the literature that focuses on these research questions, with specific attention to contemporary AI-powered chatbots, can provide a deeper understanding of the impact, effectiveness, and potential limitations of chatbot technology in education while guiding its future development and implementation. This paper will help to better understand how educational chatbots can be effectively utilized to enhance education and address the specific needs and challenges of students and educators.

Search Process

The search for the relevant literature was conducted in the following databases: ACM Digital Library, Scopus, IEEE Xplore, and Google Scholar. The search string was created using Boolean operators, and it was structured as follows: ("Education" or "Learning" or "Teaching") and ("Chatbot" or "Artificial intelligence" or "AI" or "ChatGPT"). Initially, the search yielded a total of 563 papers from all four databases. Search filters were applied based on predefined inclusion and exclusion criteria, followed by a rigorous data extraction strategy as explained below.

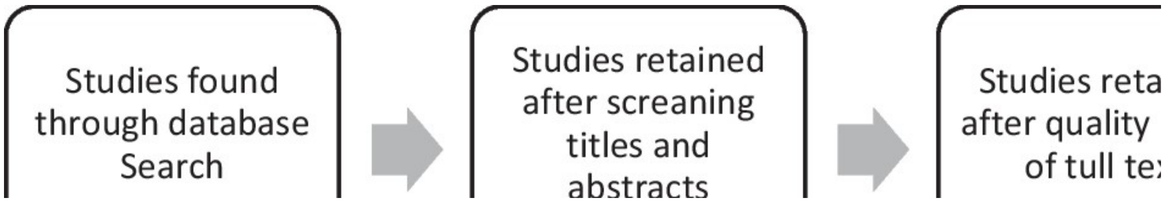
Inclusion and Exclusion Criteria

In our review process, we carefully adhered to the inclusion and exclusion criteria specified in Table 2. Criteria were determined to ensure the studies chosen are relevant to the research question (content, timeline) and maintain a certain level of quality (literature type) and consistency (language, subject area).

Criterion	Inclusion	Exclusion
Literature Type	Peer Reviewed Journals, books and book chapters	Conference proceedings, journal articles that are not published in Peer Reviewed Journals
Content	Discusses at least one educational chatbot or reviews articles on this topic	PhD thesis, Technical papers, tutorial, poster
Language	English	Non-English
Timeline	2018–2023	Before 2018
Subject Area	Artificial Intelligence (AI), chatbots and Education	Neither Artificial Intelligence (AI) nor Education

Data Extraction Strategy

All three authors collaborated to select the articles, ensuring consistency and reliability. Each article was reviewed by at least two co-authors. The article selection process involved the following stages: Initially, the authors reviewed the studies' metadata, titles, abstracts, keywords and eliminated articles that were not relevant to research questions. This reduced the number of studies to 139. Next, the authors evaluated the quality of the studies by assessing research methodology, sample size, research design, and clarity of objectives, further refining the selection to 85 articles. Finally, the authors thoroughly read the entire content of the articles. Studies offering limited empirical evidence related to our research questions were excluded. This final step reduced the number of papers to 67. Figure 1 presents the article selection process.


Studies found through database Search
Studies retained after screaning titles and abstracts
Studies reta after quality of tull te:

Results

In this section, we present the results of the reviewed articles, focusing on our research questions, particularly with regard to ChatGPT. ChatGPT, as one of the latest AI-powered chatbots, has gained significant attention for its potential applications in education. Within just eight months of its launch in 2022, it has already amassed over 100 million users, setting new records for user and traffic growth. ChatGPT stands out among AI-powered chatbots used in education due to its advanced natural language processing capabilities and sophisticated language generation, enabling more natural and human-like conversations. It excels at capturing and retaining contextual information throughout interactions, leading to more coherent and contextually relevant conversations. Unlike some educational chatbots that follow predetermined paths or rely on predefined scripts, ChatGPT is capable of engaging in open-ended dialogue and adapting to various user inputs. Its adaptability allows it to write articles, stories, and poems, provide summaries, accommodate different perspectives, and even write and debug computer code, making it a valuable tool in educational settings (Baidoo-Anu & Owusu Ansah, 2023; Tate et al., 2023; Williams, 2023).

Advantages for Students

The integration of chatbots and virtual assistants into educational settings has the potential to transform support services, improve accessibility, and contribute to more efficient and effective learning environments (Chen et al., 2023; Essel et al., 2022). AI tools have the potential to improve student success and engagement, particularly among those from disadvantaged backgrounds (Sullivan et al., 2023). However, the existing literature highlights an important gap in the discussion from a student's standpoint. A few existing research studies addressing the student's perspective of using ChatGPT in the learning process indicate that students have a Research question 1. What are the key advantages of incorporating AI chatbots in education from the viewpoint of students? positive view of ChatGPT, appreciate its capabilities, and find it helpful for their studies and work (Kasneci et al., 2023; Shoufan, 2023). Students acknowledge that ChatGPT's answers are not always accurate and emphasize the need for solid background knowledge

to utilize it effectively, recognizing that it cannot replace human intelligence (Shoufan, 2023). Common most important benefits identified by scholars are:
Homework and Study Assistance. AI-powered chatbots can provide detailed feedback on student assignments, highlighting areas of improvement and offering suggestions for further learning (Celik et al., 2022). For example, ChatGPT can act as a helpful study companion, providing explanations and clarifications on various subjects. It can assist with homework questions, offering step-by-step solutions and guiding students through complex problems (Crawford et al., 2023; Fauzi et al., 2023; Lo, 2023; Qadir, 2023; Shidiq, 2023). According to Sedaghat (2023) experiment, ChatGPT performed similarly to third-year medical students on medical exams, and could write quite impressive essays. Students can also use ChatGPT to quiz themselves on various subjects, reinforcing their knowledge and preparing for exams (Choi et al., 2023; Eysenbach, 2023; Sevgi et al., 2023; Thurzo et al., 2023).
Flexible personalized learning. AI-powered chatbots in general are now able to provide individualized guidance and feedback to students, helping them navigate through challenging concepts and improve their understanding. These systems can adapt their teaching strategies to suit each student's unique needs (Fariani et al., 2023; Kikalishvili, 2023; Schiff, 2021). Students can access ChatGPT anytime, making it convenient. According to Kasneci et al. (2023), ChatGPT's interactive and conversational nature can enhance students' engagement and motivation, making learning more enjoyable and personalized. (Khan et al., 2023) examine the impact of ChatGPT on medical education and clinical management, highlighting its ability to offer students tailored learning opportunities.
Skills development. It can aid in the enhancement of writing skills (by offering suggestions for syntactic and grammatical corrections) (Kaharuddin, 2021), foster problem-solving abilities (by providing step-by-step solutions) (Benvenuti et al., 2023), and facilitate group discussions and debates (by furnishing discussion structures and providing real-time feedback) (Ruthotto et al., 2020; Wang et al., 2020).
It's important to note that some papers raise concerns about excessive reliance on AI-generated information, potentially leading to a negative impact on student’s critical thinking and problem-solving skills (Kasneci et al., 2023)

Advantages for Educators

Research question 2. What are the key advantages of integrating AI chatbots in education from the viewpoint of educators?

With the current capabilities of AI and its future potential, AI-powered chatbots, like ChatGPT, can have a significant impact on existing instructional practices. Major benefits from educators' viewpoint identified in the literature are:

Time-Saving Assistance. AI chatbot administrative support capabilities can help educators save time on routine tasks, including scheduling, grading, and providing information to students, allowing them to allocate more time for instructional planning and student engagement. For example, ChatGPT can successfully generate various types of questions and answer keys in different disciplines. However, educators should exercise critical evaluation and customization to suit their unique teaching contexts. The expertise, experience, and comprehension of the teacher are essential in making informed pedagogical choices, as AI is not yet capable of replacing the role of a science teacher (Cooper, 2023).

Improved pedagogy. Educators can leverage AI chatbots to augment their instruction and provide personalized support. According to Herft (2023), there are various ways in which teachers can utilize ChatGPT to enhance their pedagogical approaches and assessment methods. For instance, Educators can leverage the capabilities of ChatGPT to generate open-ended question prompts that align precisely with the targeted learning objectives and success criteria of the instructional unit. By doing so, teachers can tailor educational content to cater to the distinct needs, interests, and learning preferences of each student, offering personalized learning materials and activities (Al Ka'bi, 2023; Fariani et al., 2023).

Concerns raised by scholars

Research question 3. What are the main concerns raised by scholars regarding the integration of AI chatbots in education?

Scholars' opinions on using AI in this regard are varied and diverse. Some see AI chatbots as the future of teaching and learning, while others perceive them as a potential threat. The main arguments of skeptical scholars are threefold:

Reliability and Accuracy. AI chatbots may provide biased responses or non-accurate information (Kasneci et al., 2023; Sedaghat, 2023). If the chatbot provides incorrect information or guidance, it could mislead students and hinder their learning progress. According to Sevgi et al. (2023), although ChatGPT exhibited captivating and thought-provoking answers, it should not be regarded as a reliable information source. This point is especially important for medical education. Within the field of medical education, it is crucial to guarantee the reliability and accuracy of the information chatbots provide (Khan et al., 2023). If the training data used to develop an AI chatbot contains biases, the chatbot may inadvertently reproduce those biases in its responses, potentially including skewed perspectives, stereotypes, discriminatory language, or biased recommendations. This is of particular concern in an educational context.

Fair assessments. One of the challenges that educators face with the integration of Chatbots in education is the difficulty in assessing students' work, particularly when it comes to written assignments or responses. AI-generated text detection, while continually improving, is not yet foolproof and can produce false negatives or positives. This creates uncertainty and can undermine the credibility of the assessment process. Educators may struggle to discern whether the responses are genuinely student-generated or if they have been provided by an AI, affecting the accuracy of grading and feedback. This raises concerns about academic integrity and fair assessment practices (AlAfnan et al., 2023; Kung et al., 2023).

Ethical issues. The integration of AI chatbots in education raises several ethical implications, particularly concerning data privacy, security, and responsible AI use. As AI chatbots interact with students and gather data during conversations, necessitating the establishment of clear guidelines and safeguards. For example, medical education frequently encompasses the acquisition of knowledge pertaining to delicate and intimate subjects, including patient confidentiality and ethical considerations within the medical field and thus ethical and proper utilization of chatbots holds significant importance (Masters, 2023; Miao & Ahn, 2023; Sedaghat, 2023; Thurzo et al., 2023).

For these and other geopolitical reasons, ChatGPT is banned in countries with strict internet censorship policies, like North Korea, Iran, Syria, Russia, and China. Several nations prohibited the usage of the application due to privacy apprehensions. Meanwhile, North Korea, China, and Russia, in particular, contended that the U.S. might employ ChatGPT for disseminating misinformation. Conversely, OpenAI restricts access to ChatGPT in certain countries, such as Afghanistan and Iran, citing geopolitical constraints, legal considerations, data protection regulations, and internet accessibility as the basis for this decision. Italy became the first Western country to ban ChatGPT (Browne, 2023) after the country's data protection authority called on OpenAI to stop processing Italian residents' data. They claimed that ChatGPT did not comply with the European General Data Protection Regulation. However, after OpenAI clarified the data privacy issues with Italian data protection authority, ChatGPT returned to Italy. To avoid cheating on school homework and assignments, ChatGPT was also blocked in all New York school devices and networks so that students and teachers could no longer access it (Elsen-Rooney, 2023; Li et al., 2023). These examples highlight the lack of readiness to embrace recently developed AI tools. There are numerous concerns that must be addressed in order to gain broader acceptance and understanding.

To summarize, incorporating AI chatbots in education brings personalized learning for students and time efficiency for educators. Students benefit from flexible study aid and skill development. However, concerns arise regarding the accuracy of information, fair assessment practices, and ethical considerations. Striking a balance between these advantages and concerns is crucial for responsible integration in education.

Discussion

The integration of artificial intelligence (AI) chatbots in education has the potential to revolutionize how students learn and interact with information. One significant advantage of AI chatbots in education is their ability to provide personalized and engaging learning experiences. By tailoring their interactions to individual students' needs and preferences, chatbots offer customized feedback and instructional support, ultimately enhancing student engagement and information retention. However, there are potential difficulties in fully replicating the human educator experience with chatbots. While they can provide customized instruction, chatbots may not match human instructors' emotional support and mentorship. Understanding the importance of human engagement and expertise in education is crucial. A teacher's role encompasses more than just sharing knowledge.

They offer students guidance, motivation, and emotional support—elements that AI cannot completely replicate.

We find that AI chatbots may benefit students as well as educators in various ways, however, there are significant concerns that need to be addressed in order to harness its capabilities effectively. Specifically, educational institutions should implement preventative measures. This includes (a) creating awareness among students, focusing on topics such as digital inequality, the reliability and accuracy of AI chatbots, and associated ethical considerations; and (b) offering regular professional development training for educators. This training should initially focus on enabling educators to integrate diverse in-class activities and assignments into the curriculum, aimed at nurturing students' critical thinking and problem-solving skills while ensuring fair performance evaluation. Additionally, this training should cover educating educators about the capabilities and potential educational uses of AI chatbots, along with providing them with best practices for effectively integrating these tools into their teaching methods.

As technology continues to advance, AI-powered educational chatbots are expected to become more sophisticated, providing accurate information and offering even more individualized and engaging learning experiences. They are anticipated to engage with humans using voice recognition, comprehend human emotions, and navigate social interactions. Consequently, their potential impact on future education is substantial. This includes activities such as establishing educational objectives, developing teaching methods and curricula, and conducting assessments (Latif et al., 2023). Considering Microsoft's extensive integration efforts of ChatGPT into its products (Rudolph et al., 2023; Warren, 2023), it is likely that ChatGPT will become widespread soon. Educational institutions may need to rapidly adapt their policies and practices to guide and support students in using educational chatbots safely and constructively manner (Baidoo-Anu & Owusu Ansah, 2023). Educators and researchers must continue to explore the potential benefits and limitations of this technology to fully realize its potential.

Conclusion

The widespread adoption of chatbots and their increasing accessibility has sparked contrasting reactions across different sectors, leading to considerable confusion in the field of education. Among educators and learners, there is a notable trend—while learners are excited about chatbot integration, educators’ perceptions are particularly critical. However, this situation presents a unique opportunity, accompanied by unprecedented challenges. Consequently, it has prompted a significant surge in research, aiming to explore the impact of chatbots on education.

In this article, we present a systematic review of the latest literature with the objective of identifying the potential advantages and challenges associated with integrating chatbots in education. Through this review, we have been able to highlight critical gaps in the existing research that warrant further in-depth investigation. Addressing these gaps will be instrumental in optimizing the implementation of chatbots and harnessing their full potential in the educational landscape, thereby benefiting both educators and students alike. Further research will play a vital role in comprehending the long-term impact, variations based on student characteristics, pedagogical strategies, and the user experience associated with integrating chatbots in education.

From the viewpoint of educators, integrating AI chatbots in education brings significant advantages. AI chatbots provide time-saving assistance by handling routine administrative tasks such as scheduling, grading, and providing information to students, allowing educators to focus more on instructional planning and student engagement. Educators can improve their pedagogy by leveraging AI chatbots to augment their instruction and offer personalized support to students. By customizing educational content and generating prompts for open-ended questions aligned with specific learning objectives, teachers can cater to individual student needs and enhance the learning experience. Additionally, educators can use AI chatbots to create tailored learning materials and activities to accommodate students' unique interests and learning styles.

Incorporating AI chatbots in education offers several key advantages from students' perspectives. AI-powered chatbots provide valuable homework and study assistance by offering detailed feedback on assignments, guiding students through complex problems, and providing step-by-step solutions. They also act as study companions, offering explanations and clarifications on various subjects. They can be used for self-quizzing to reinforce knowledge and prepare for exams. Furthermore, these chatbots facilitate flexible personalized learning, tailoring their teaching strategies to suit each student's unique needs. Their interactive and conversational nature enhances student engagement and motivation, making learning more enjoyable and personalized. Also, AI chatbots contribute to skills development by suggesting syntactic and grammatical corrections to enhance writing skills, providing problem-solving guidance, and facilitating group discussions and debates with real-time feedback. Overall, students appreciate the capabilities of AI chatbots and find them helpful for their studies and skill development, recognizing that they complement human intelligence rather than replace it.

The presence of AI chatbots also brought lots of skepticism among scholars. While some see transformative potential, concerns loom over reliability, accuracy, fair assessments, and ethical dilemmas. The fear of misinformation compromised academic integrity, and data privacy issues cast an eerie shadow over the implementation of AI chatbots. Based on the findings of the reviewed papers, it is commonly concluded that addressing some of the challenges related to the use of AI chatbots in education can be accomplished by introducing preventative measures. More specifically, educational institutions must prioritize creating awareness among students about the risks associated with AI chatbots, focusing on essential aspects like digital inequality and ethical considerations. Simultaneously, investing in the continuous development of educators through targeted training is key. Empowering educators to effectively integrate AI chatbots into their teaching methods, fostering critical thinking and fair evaluation, will pave the way for a more effective and engaging educational experience.

The implications of the research findings for policymakers and researchers are extensive, shaping the future integration of chatbots in education. The findings emphasize the need to establish guidelines and regulations ensuring the ethical development and deployment of AI chatbots in education. Policies should specifically focus on data privacy, accuracy, and transparency to mitigate potential risks and build trust within the educational community. Additionally, investing in research and development to enhance AI chatbot capabilities and address identified concerns is crucial for a seamless integration into educational systems. Researchers are strongly encouraged to fill the identified research gaps through rigorous studies that delve deeper into the impact of chatbots on education. Exploring the long-term effects, optimal integration strategies, and addressing ethical considerations should take the forefront in research initiatives.

Funding

The authors declare that this research paper did not receive any funding from external organizations. The study was conducted independently and without financial support from any source. The authors have no financial interests or affiliations that could have influenced the design, execution, analysis, or reporting of the research.

Coding and Documentation

```
<?xml version="1.0" encoding="utf-8"?>

<manifest
xmlns:android="http://schemas.android.com/apk/res/android">


    <application

        android:allowBackup="true"

        android:icon="@mipmap/ic_launcher"

        android:label="@string/app_name"

        android:roundIcon="@mipmap/ic_launcher_round"

        android:supportsRtl="true"

android:theme="@style/Theme.AppCompat.Light.NoActionBar">

        <activity

            android:exported="true"

android:configChanges="orientation|screenSize|screenLayout|
keyboardHidden|mnc|colorMode|density|fontScale|fontWeightAd
justment|keyboard|layoutDirection|locale|mcc|navigation|sma
llestScreenSize|touchscreen|uiMode"

            android:name=".MainActivity"

        >

            <intent-filter>

                <action
android:name="android.intent.action.MAIN" />


                <category
android:name="android.intent.category.LAUNCHER" />

            </intent-filter>
        </activity>
    </application>

</manifest>

package com.myapplication

import android.app.Application
import di.initKoin

class AndroidApp : Application() {
    override fun onCreate() {
        super.onCreate()
        initKoin()
    }
}
package com.myapplication

import MainView
import android.os.Bundle
import androidx.activity.compose.setContent
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MainView()
        }
    }
}
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.application

fun main() = application {
    Window(onCloseRequest = ::exitApplication) {
        MainView()
    }
}
package data.network.dto

import io.ktor.util.encodeBase64
import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable



@Serializable
data class Request(
    @SerialName("contents") private val contents:
List<RequestContent>
) {
    class RequestBuilder {
        private val parts = mutableListOf<RequestPart>()

        private fun addPart(part: RequestPart) {
            parts.add(part)
        }

        fun addText(text: String): RequestBuilder {
            addPart(RequestPart(text = text))
            return this
        }

        fun addImage(image: ByteArray, mimeType: String =
"image/png"): RequestBuilder {
            val imagePart = RequestPart(requestInlineData =
RequestInlineData(mimeType, image.encodeBase64()))
            addPart(imagePart)
            return this
        }

        fun addImages(images: List<ByteArray>, mimeType:
String = "image/png"): RequestBuilder {
            images.forEach {
                val part = RequestPart(requestInlineData =
RequestInlineData(mimeType, it.encodeBase64()))
                addPart(part)
            }
            return this
        }
        fun build(role: String? = null): Request {
            return Request(
                contents = listOf(
                    RequestContent(parts = parts, role =
role)
                )
            )
        }
    }
}

@Serializable
data class RequestContent(
    @SerialName("parts") val parts: List<RequestPart> =
emptyList(),
    @SerialName("role") val role: String? = null,
)

@Serializable
data class RequestPart(
    @SerialName("text") val text: String? = null,
    @SerialName("inlineData") val requestInlineData:
RequestInlineData? = null,
)

@Serializable
data class RequestInlineData(
    @SerialName("mimeType") val mimeType: String,
    @SerialName("data") val data: String,
)
package data.network.dto

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable



@Serializable
data class Response(
    @SerialName("candidates") val candidates:
List<Candidate> = emptyList(),
    @SerialName("promptFeedback") val promptFeedback:
PromptFeedback? = null,
    @SerialName("error") val error: Error? = null,
) {
    fun getText(): String? =

candidates.firstOrNull()?.content?.parts?.firstOrNull()?.te
xt
}

@Serializable
data class Candidate(
    @SerialName("index") val index: Int,
    @SerialName("content") val content: Content? = null,
    @SerialName("finishReason") val finishReason: String? =
null,
    @SerialName("safetyRatings") val safetyRatings:
List<SafetyRating> = emptyList()
)


@Serializable
data class Content(
    @SerialName("parts") val parts: List<Part> =
emptyList(),
    @SerialName("role") val role: String? = null,
)


@Serializable
data class Part(
    @SerialName("text") val text: String? = null,
)


@Serializable
data class PromptFeedback(
    @SerialName("safetyRatings") val safetyRatings:
List<SafetyRating> = emptyList()
)


@Serializable
data class SafetyRating(
    @SerialName("category") val category: String,
    @SerialName("probability") val probability: String
```

```
)

@Serializable
data class Error(
    @SerialName("code") val code: Int,
    @SerialName("message") val message: String,
    @SerialName("status") val status: String,

    )
package data.network

import data.network.dto.Request
import data.network.dto.Response
import io.ktor.client.HttpClient
import io.ktor.client.plugins.HttpTimeout
import
io.ktor.client.plugins.contentnegotiation.ContentNegotiatio
n
import io.ktor.client.plugins.logging.DEFAULT
import io.ktor.client.plugins.logging.LogLevel
import io.ktor.client.plugins.logging.Logger
import io.ktor.client.plugins.logging.Logging
import io.ktor.client.request.post
import io.ktor.client.statement.bodyAsText
import io.ktor.serialization.kotlinx.json.json
import io.ktor.util.InternalAPI
import kotlinx.serialization.ExperimentalSerializationApi
import kotlinx.serialization.encodeToString
import kotlinx.serialization.json.Json


const val BASE_URL =
"https://generativelanguage.googleapis.com"
const val TIMEOUT = 30000L


@OptIn(ExperimentalSerializationApi::class,
InternalAPI::class)
class GeminiService {


    // region Setup
    private val client = HttpClient {
        install(ContentNegotiation) {
            json(Json {
                isLenient = true
                explicitNulls = false
                encodeDefaults = true
                ignoreUnknownKeys = true
            })
        }
        install(HttpTimeout) {
            connectTimeoutMillis = TIMEOUT
            socketTimeoutMillis = TIMEOUT
            requestTimeoutMillis = TIMEOUT
        }
        install(Logging) {
            logger = Logger.DEFAULT
            level = LogLevel.ALL
        }
    }
    // endregion

    // region API key

    // Enter your personal api key here
    private var apiKey: String = ""

    fun getApiKey(): String {
        return apiKey
    }

    fun setApiKey(key: String) {
        apiKey = key
    }

    // endregion

    // region API calls
    suspend fun generateContent(prompt: String): Response {
        return
makeApiRequest("$BASE_URL/v1beta/models/gemini-
pro:generateContent?key=$apiKey") {
            addText(prompt)
        }
    }

    suspend fun generateContentWithMedia(prompt: String,
images: List<ByteArray>): Response {
        return
makeApiRequest("$BASE_URL/v1beta/models/gemini-pro-
vision:generateContent?key=$apiKey") {
            addText(prompt)
            addImages(images)
        }
    }

    private suspend fun makeApiRequest(url: String,
requestBuilder: Request.RequestBuilder.() -> Unit):
Response {
        val request =
Request.RequestBuilder().apply(requestBuilder).build()

        val response: String = client.post(url) {
            body = Json.encodeToString(request)
        }.bodyAsText()

        return Json.decodeFromString(response)
    }

    // endregion

}
```

```
package data.preferences


class GeminiTalkerPref(private val preferencesStorage:
PreferencesStorage) {
//    private val preferencesStorage: PreferencesStorage =
PreferencesStorage()


    fun save(key: String, value: String) {
        preferencesStorage.putString(key, value)
    }


    fun load(key: String, default: String = ""): String? {
        return preferencesStorage.getString(key, default)
    }
}
package data.preferences



expect class PreferencesStorage {
    fun getString(key: String, defaultValue: String = ""):
String
    fun putString(key: String, value: String)
}
package data.preferences


object PrefKey {
    const val GEMINI_PRO_API_KEY: String =
"geminiProApiKey"
```

```
    const val GEMINI_TALKER_PREF: String =
"geminiTalkerPref"

}

package data.repository


import data.network.GeminiService

import domain.model.Status

import domain.repository.GeminiRepository

import io.ktor.utils.io.errors.IOException


class GeminiRepositoryImpl : GeminiRepository {


    private val geminiService = GeminiService()


    override suspend fun generate(prompt: String, images:
List<ByteArray>): Status {

        return try {

            val response = when {

                images.isEmpty() ->
geminiService.generateContent(prompt)

                else ->
geminiService.generateContentWithMedia(prompt, images)

            }


            val status = response.error?.let {

                Status.Error(it.message)

            } ?: response.getText()?.let {

                Status.Success(it)
            } ?: Status.Error("An error occurred, please
retry.")

            status

        } catch (e: IOException) {
            Status.Error("Unable to connect to the server.
Please check your internet connection and try again.")
        } catch (e: Exception) {
            Status.Error("An error occurred, please
retry.")
        }
    }

    override fun getApiKey(): String {
        return geminiService.getApiKey()
    }

    override fun setApiKey(key: String) {
        geminiService.setApiKey(key)
    }


}
package di

import org.koin.core.context.startKoin
import org.koin.dsl.module
```

```
val dataModule = module {
//    single {
//        val json = Json { ignoreUnknownKeys = true }
//        HttpClient {
//            install(ContentNegotiation) {
//                // TODO Fix API so it serves
application/json
//                json(json, contentType = ContentType.Any)
//            }
//        }
//    }

//    single<GeminiRepository> {
GeminiRepositoryImpl(get()) }
//    single { GeminiService(get()) }
}

//val screenModelsModule = module {
//    factoryOf(::ChatViewModel)
//}

fun initKoin() {
    startKoin {
        modules(
            dataModule,
//            screenModelsModule,
        )
    }
}

package domain.model

import kotlinx.datetime.Clock
import kotlinx.datetime.TimeZone
import kotlinx.datetime.toLocalDateTime

data class Message(
    val sender: Sender,
    val text: String,
    val images: List<ByteArray> = emptyList(),
    val isLoading: Boolean = false,
) {
    val time: String
        get() = currentTime()

    val isBotMessage: Boolean
        get() = sender == Sender.Bot

    private fun currentTime(): String {
        val datetime =
Clock.System.now().toLocalDateTime(TimeZone.currentSystemDe
fault())
```

```
        val hour = if (datetime.hour < 10)
"0${datetime.hour}" else datetime.hour

        val minute = if (datetime.minute < 10)
"0${datetime.minute}" else datetime.minute

        return "${hour}:${minute}"

    }

}


enum class Sender {

    User,

    Bot;


    override fun toString(): String {

        return when (this) {

            User -> "You"

            Bot -> "GeminiTalker"

        }

    }

}

package domain.model

sealed class Status {

    data object Idle : Status()

    class Success(val data: String) : Status()

    class Error(val message: String) : Status()

    data object Loading : Status()

}

package domain.repository
```

```
import domain.model.Status


interface GeminiRepository {
    suspend fun generate(prompt: String, images:
List<ByteArray> = emptyList()): Status


    fun getApiKey(): String


    fun setApiKey(key: String)
}


package presentation.theme



import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Shapes
import androidx.compose.material3.Surface
import androidx.compose.material3.Typography
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
```

```kotlin
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

private val LightColorScheme = lightColorScheme(
    tertiaryContainer = Gray400,
    onPrimaryContainer = Color.Black.copy(alpha = 0.8f),
    onSecondaryContainer = Color.LightGray,
    surface = Gray400
)

private val DarkColorScheme = darkColorScheme(
    primaryContainer = SurfaceDark,
    tertiaryContainer = SurfaceDark,
    onPrimaryContainer = Color.White,
    onSecondaryContainer = Gray700,
    surface = BackgroundDark
)

private val shapes = Shapes(
    extraSmall = RoundedCornerShape(3.dp),
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(8.dp),
    large = RoundedCornerShape(16.dp),
    extraLarge = RoundedCornerShape(32.dp)
)
```

```
private val typography = Typography(
    bodyMedium = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 16.sp
    )
)

@Composable
internal fun GeminiTalkerTheme(content: @Composable () ->
Unit) {
    MaterialTheme(
        colorScheme = if (isSystemInDarkTheme())
DarkColorScheme else LightColorScheme,
        typography = typography,
        shapes = shapes,
        content = { Surface(content = content) }
    )
}

@Composable
internal expect fun SystemAppearance(isDark: Boolean)
package presentation.ui.component

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.wrapContentWidth
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowForward
import androidx.compose.material.icons.outlined.Add
import androidx.compose.material.icons.rounded.Close
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
```

```kotlin
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.draw.shadow
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.ImageBitmap
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.IntOffset
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.preat.peekaboo.image.picker.SelectionMode
import
com.preat.peekaboo.image.picker.rememberImagePickerLauncher
import com.preat.peekaboo.image.picker.toImageBitmap
import domain.model.Status
import presentation.theme.Gray700

/**
 * TODO : allow sending attachments without text
 */
@Composable
fun CustomBottomBar(
    modifier: Modifier = Modifier,
    status: Status,
    onSendClick: (String, List<ByteArray>) -> Unit
) {
    val textState = remember { mutableStateOf("") }
    val images = remember {
mutableStateOf(listOf<ByteArray>()) }

    val scope = rememberCoroutineScope()
    val multipleImagePicker = rememberImagePickerLauncher(
        selectionMode = SelectionMode.Multiple(),
        scope = scope,
        onResult = { images.value = it }
    )
    Column {
        LazyRow {
            items(images.value.size) { index ->
                val bitmap =
images.value[index].toImageBitmap()
                ImageAttachment(
                    bitmap = bitmap,
                    onCloseClick = {
                        val mutableImages =
images.value.toMutableList()
                        mutableImages.removeAt(index)
                        images.value = mutableImages
                    }
                )

            }
        }
        TextField(
            value = textState.value,
            onValueChange = { textState.value = it },
            maxLines = 3,
            placeholder = {
                Text(
                    text = "Type a message...",
                    style = TextStyle(
                        fontSize = 14.sp,
                        color = Gray700,
                    ),
                    textAlign = TextAlign.Center
                )
            },
            colors = TextFieldDefaults.colors(
                focusedContainerColor =
MaterialTheme.colorScheme.tertiaryContainer,
                unfocusedContainerColor =
MaterialTheme.colorScheme.tertiaryContainer,
                disabledContainerColor =
MaterialTheme.colorScheme.tertiaryContainer,
                focusedIndicatorColor = Color.Transparent,
                unfocusedIndicatorColor =
Color.Transparent,
            ),
            trailingIcon = {
                Button(
                    onClick = {
                        onSendClick(textState.value,
images.value)
                        images.value = emptyList()
                        textState.value = ""
                    },
                    enabled = textState.value.isNotBlank()
&& status != Status.Loading,
                    content = {
                        if (status is Status.Loading)
                            CircularProgressIndicator(
                                modifier =
Modifier.size(20.dp),
                                color =
MaterialTheme.colorScheme.onPrimary
                            )
                        else {
                            Icon(
                                Icons.Default.ArrowForward,
                                contentDescription = null,
                                modifier =
Modifier.rotate(-90.0F).size(20.dp),
                            )
                        }
                    },
                    colors = ButtonDefaults.buttonColors(
                        contentColor =
MaterialTheme.colorScheme.onPrimary,
                        containerColor =
MaterialTheme.colorScheme.primary,
                        disabledContainerColor =
MaterialTheme.colorScheme.onSecondaryContainer
                    ),
                    shape = RoundedCornerShape(30),
                    modifier = Modifier.padding(horizontal
= 10.dp)
                )
            },
            leadingIcon = {
                IconButton(
                    onClick = {
                        multipleImagePicker.launch()
                    },
                    content = {
                        Icon(
                            Icons.Outlined.Add,
                            contentDescription = null,
                            modifier = Modifier,
                            tint =
MaterialTheme.colorScheme.onSecondaryContainer
                        )
                    },
                )
            },
            modifier = modifier,
            shape = RoundedCornerShape(24),
        )
    }
}

@Composable
private fun ImageAttachment(bitmap: ImageBitmap,
onCloseClick: () -> Unit = {}) {

    val iconSize = 20.dp
    val offsetInPx = LocalDensity.current.run { (iconSize /
2).roundToPx() }

    Box(modifier = Modifier.padding((iconSize / 3))) {
        Image(
            bitmap = bitmap,
            contentDescription = null,
            contentScale = ContentScale.Fit,
            modifier = Modifier
                .height(80.dp)
                .wrapContentWidth()
                .shadow(1.dp, RoundedCornerShape(12.dp)),
        )

        IconButton(
            onClick = {
                onCloseClick()
            },
            modifier = Modifier
                .offset {
                    IntOffset(x = +offsetInPx, y = -
offsetInPx)
                }
                .padding(5.dp)
                .clip(CircleShape)

.background(MaterialTheme.colorScheme.primaryContainer)
                .size(iconSize)
                .align(Alignment.TopEnd)
        ) {
            Icon(
                imageVector = Icons.Rounded.Close,
                contentDescription = "",
                tint =
MaterialTheme.colorScheme.onPrimaryContainer,
                modifier = Modifier.size(12.dp)
            )
        }
    }
}

package presentation.ui.component

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Key
import androidx.compose.material.icons.filled.Visibility
import androidx.compose.material.icons.filled.VisibilityOff
import androidx.compose.material.icons.rounded.Close
import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle

import androidx.compose.ui.text.font.FontWeight

import
androidx.compose.ui.text.input.PasswordVisualTransformation

import androidx.compose.ui.text.input.VisualTransformation

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.compose.ui.window.DialogProperties

import presentation.theme.Gray700


@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun CustomDialog(

    value: String,

    onVisibilityChanged: (Boolean) -> Unit,

    onSaveClicked: (String) -> Unit) {


    val keyVisibility = remember { mutableStateOf(false) }

    val textField = remember { mutableStateOf(value) }


    AlertDialog(

        onDismissRequest = { onVisibilityChanged(false) },

        properties = DialogProperties(

            dismissOnBackPress = true,

            dismissOnClickOutside = true,

        )
    ) {
        Card(
            modifier = Modifier,
            shape = RoundedCornerShape(20.dp),
            elevation = CardDefaults.cardElevation(4.dp),
            colors = CardDefaults.cardColors(
                containerColor =
MaterialTheme.colorScheme.primaryContainer,
            )
        ) {
            Column(
                modifier = Modifier.padding(20.dp)
            ) {
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement =
Arrangement.SpaceBetween,
                    verticalAlignment =
Alignment.CenterVertically
                ) {
                    Text(
                        text = "Enter your personal API
Key",
                        style = TextStyle(
                            fontSize = 16.sp,
                            fontWeight = FontWeight.Bold
                        )
                    )
```

```
                        IconButton(
                            onClick = {
onVisibilityChanged(false) },
                            content = {
                                Icon(
                                    imageVector =
Icons.Rounded.Close,
                                    contentDescription = null,
                                )
                            },
                        )
                    }

                    Spacer(modifier = Modifier.height(20.dp))

                    TextField(
                        value = textField.value,
                        onValueChange = { textField.value = it
},
                        singleLine = true,
                        placeholder = {
                            Text(
                                text = "Your API Key...",
                                style = TextStyle(
                                    fontSize = 14.sp,
                                    color = Gray700,
                                ),
                                textAlign = TextAlign.Center
                            )
                        },
                        visualTransformation =
                            if (keyVisibility.value)
    VisualTransformation.None
                            else
    PasswordVisualTransformation(),
                        colors = TextFieldDefaults.colors(
                            focusedContainerColor =
    MaterialTheme.colorScheme.surface,
                            unfocusedContainerColor =
    MaterialTheme.colorScheme.surface,
                            disabledContainerColor =
    MaterialTheme.colorScheme.surface,
                            focusedIndicatorColor =
    Color.Transparent,
                            unfocusedIndicatorColor =
    Color.Transparent,
                        ),
                        leadingIcon = {
                            Icon(
                                imageVector =
    Icons.Default.Key,
                                contentDescription = null,
                                tint =
    MaterialTheme.colorScheme.onSecondaryContainer
                            )
                        },
                        trailingIcon = {
                            IconButton(
                                onClick = { keyVisibility.value
    = !keyVisibility.value },
                                content = {
                                    Icon(
                                        imageVector =
                                            if
    (keyVisibility.value) Icons.Default.Visibility
                                            else
    Icons.Default.VisibilityOff,
                                        contentDescription =
    null,
                                        tint =
    MaterialTheme.colorScheme.onSecondaryContainer
                                    )
                                },
                            )
                        },
                        shape = RoundedCornerShape(24),
                    )

                    Spacer(modifier = Modifier.height(20.dp))

                    Button(
                        enabled = textField.value.isNotBlank()
    && textField.value != value,
                        onClick = {
                            onSaveClicked(textField.value)
                            onVisibilityChanged(false)
                        },
                        shape = RoundedCornerShape(12.dp),
                        modifier = Modifier
                            .fillMaxWidth()
                            .height(40.dp)
                    ) {
                        Text(text = "Save")
                    }

                }
            }
        }
    }
    package presentation.ui.component

    import androidx.compose.foundation.shape.RoundedCornerShape
    import androidx.compose.material3.Snackbar
    import androidx.compose.material3.SnackbarData
    import androidx.compose.runtime.Composable
    import androidx.compose.ui.graphics.Color

    @Composable
    fun CustomSnackBar(
        data: SnackbarData,
        containerColor: Color,
        contentColor: Color = Color.White,
    ) {
        Snackbar(
            containerColor = containerColor,
            contentColor = contentColor,
            dismissActionContentColor = Color.White,
            shape = RoundedCornerShape(20),
            snackbarData = data
        )
    }

    package presentation.ui.component

    import androidx.compose.foundation.Image
    import androidx.compose.foundation.background
    import androidx.compose.foundation.clickable
    import androidx.compose.foundation.layout.Column
    import androidx.compose.foundation.layout.Row
    import androidx.compose.foundation.layout.Spacer
    import androidx.compose.foundation.layout.fillMaxWidth
    import androidx.compose.foundation.layout.padding
    import androidx.compose.foundation.layout.size
    import androidx.compose.foundation.layout.width
    import androidx.compose.foundation.shape.CircleShape
    import androidx.compose.material.icons.Icons
    import androidx.compose.material.icons.filled.Key
    import androidx.compose.material3.ExperimentalMaterial3Api
    import androidx.compose.material3.Icon
    import androidx.compose.material3.IconButton
    import androidx.compose.material3.MaterialTheme
    import androidx.compose.material3.Text

    import androidx.compose.material3.TopAppBar

    import androidx.compose.material3.TopAppBarDefaults

    import androidx.compose.runtime.Composable

    import androidx.compose.ui.Modifier

    import androidx.compose.ui.draw.clip

    import androidx.compose.ui.layout.ContentScale

    import androidx.compose.ui.text.TextStyle

    import androidx.compose.ui.text.font.FontWeight

    import androidx.compose.ui.unit.dp

    import androidx.compose.ui.unit.sp

    import
    org.jetbrains.compose.resources.ExperimentalResourceApi

    import org.jetbrains.compose.resources.painterResource

    import presentation.theme.Gray700



    @OptIn(ExperimentalMaterial3Api::class,
    ExperimentalResourceApi::class)

    @Composable

    fun CustomAppBar(onActionClick: () -> Unit = {}, modifier:
    Modifier = Modifier) {

        Row(

            modifier = modifier

    .background(MaterialTheme.colorScheme.primaryContainer)

                .fillMaxWidth()

        ) {
            TopAppBar(
                modifier = Modifier
                    .padding(vertical = 5.dp)
                    .fillMaxWidth(),
                scrollBehavior =
TopAppBarDefaults.pinnedScrollBehavior(),
                title = {
                    Column {
                        Text(
                            text = "GeminiTalker",
                            style = TextStyle(
                                fontSize = 20.sp,
                                fontWeight =
FontWeight.ExtraBold,
                                color =
MaterialTheme.colorScheme.onPrimaryContainer,
                            ),
                        )
                        Text(
                            text = "Powered by Gemini Pro",
                            style = TextStyle(
                                fontSize = 13.sp,
                                fontWeight = FontWeight.Normal,
                                color = Gray700,
                            ),
                        )
                    }
                },
                navigationIcon = {
                    Row {
                        Spacer(modifier =
Modifier.width(10.dp))
                        Image(
                            painter =
painterResource("logo.png"),
                            contentDescription = "Compose
Multiplatform icon",
                            contentScale = ContentScale.Crop,
                            modifier = Modifier
                                .size(45.dp)
                                .clip(CircleShape)
                                .clickable {}
                        )
                        Spacer(modifier = Modifier.width(5.dp))
                    }
                },
                actions = {
                    IconButton(
                        onClick = {
                            onActionClick()
                        },
                        content = {
                            Icon(
                                Icons.Filled.Key,
                                contentDescription = null,
                                tint =
    MaterialTheme.colorScheme.onPrimaryContainer,
                            )
                        },
                    )
                },

                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor =
    MaterialTheme.colorScheme.primaryContainer,
                )
            )
        }
    }
    package presentation.ui.component

    import androidx.compose.animation.core.Animatable
    import
    androidx.compose.animation.core.LinearOutSlowInEasing
    import androidx.compose.animation.core.RepeatMode
    import androidx.compose.animation.core.infiniteRepeatable
    import androidx.compose.animation.core.keyframes
    import androidx.compose.foundation.background
    import androidx.compose.foundation.layout.Arrangement
    import androidx.compose.foundation.layout.Box
    import androidx.compose.foundation.layout.Row
    import androidx.compose.foundation.layout.size
    import androidx.compose.foundation.shape.CircleShape
    import androidx.compose.material3.MaterialTheme

    import androidx.compose.runtime.Composable

    import androidx.compose.runtime.LaunchedEffect

    import androidx.compose.runtime.remember

    import androidx.compose.ui.Modifier

    import androidx.compose.ui.graphics.Color

    import androidx.compose.ui.graphics.graphicsLayer

    import androidx.compose.ui.platform.LocalDensity

    import androidx.compose.ui.unit.Dp

    import androidx.compose.ui.unit.dp

    import kotlinx.coroutines.delay


    @Composable

    fun LoadingAnimation(

        modifier: Modifier = Modifier,

        circleSize: Dp = 25.dp,

        circleColor: Color = MaterialTheme.colorScheme.primary,

        spaceBetween: Dp = 10.dp,

        travelDistance: Dp = 20.dp

    ) {

        val circles = listOf(

            remember { Animatable(initialValue = 0f) },

            remember { Animatable(initialValue = 0f) },

            remember { Animatable(initialValue = 0f) }

        )


        circles.forEachIndexed { index, animatable ->
```

```
            LaunchedEffect(key1 = animatable) {
                delay(index * 100L)
                animatable.animateTo(
                    targetValue = 1f,
                    animationSpec = infiniteRepeatable(
                        animation = keyframes {
                            durationMillis = 1200
                            0.0f at 0 with
LinearOutSlowInEasing
                            1.0f at 300 with
LinearOutSlowInEasing
                            0.0f at 600 with
LinearOutSlowInEasing
                            0.0f at 1200 with
LinearOutSlowInEasing
                        },
                        repeatMode = RepeatMode.Restart
                    )
                )
            }
        }

        val circleValues = circles.map { it.value }
        val distance = with(LocalDensity.current) {
travelDistance.toPx() }

        Row(
            modifier = modifier,
            horizontalArrangement =
Arrangement.spacedBy(spaceBetween)
        ) {
            circleValues.forEach { value ->
                Box(
                    modifier = Modifier
                        .size(circleSize)
                        .graphicsLayer {
                            translationY = -value * distance
                        }
                        .background(
                            color = circleColor,
                            shape = CircleShape
                        )
                )
            }
        }

}
package presentation.ui.component

import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.expandHorizontally
import androidx.compose.animation.fadeIn
import androidx.compose.animation.scaleIn
import androidx.compose.animation.slideInHorizontally
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.wrapContentWidth
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.LocalContentColor
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.TransformOrigin
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.mikepenz.markdown.compose.Markdown
import com.mikepenz.markdown.model.markdownColor
import domain.model.Message

import presentation.theme.Gray700


@Composable

inline fun MessageBubble(message: Message, modifier:
Modifier = Modifier) {

    val bubbleColor =

        if (message.isBotMessage)
MaterialTheme.colorScheme.surface

        else MaterialTheme.colorScheme.secondaryContainer


    var visibility by remember { mutableStateOf(false) }


    LaunchedEffect(key1 = true) {

        visibility = true

    }

    AnimatedVisibility(

        visible = visibility,

        enter = slideInHorizontally()

                + expandHorizontally(expandFrom =
Alignment.Start)

                + scaleIn(transformOrigin =
TransformOrigin(0.5f, 0f))

                + fadeIn(initialAlpha = 0.3f),

    ) {

        Box(

            contentAlignment = if (!message.isBotMessage)
Alignment.CenterEnd else Alignment.CenterStart,
            modifier = modifier
                .padding(
                    start = if (message.isBotMessage) 0.dp
else 50.dp,
                    end = if (message.isBotMessage) 50.dp
else 0.dp,
                )
                .fillMaxWidth()
        ) {
            Row(verticalAlignment = Alignment.Bottom) {
                Column {
                    Box(
                        modifier = Modifier
                            .clip(
                                RoundedCornerShape(
                                    bottomStart = 20.dp,
                                    bottomEnd = 20.dp,
                                    topEnd = if
(message.isBotMessage) 20.dp else 2.dp,
                                    topStart = if
(message.isBotMessage) 2.dp else 20.dp
                                )
                            )
                            .background(color =
bubbleColor)
                            .padding(vertical = 5.dp,
horizontal = 16.dp),
                    ) {
                        Column {
                            Text(
                                text =
message.sender.toString(),
                                fontWeight =
FontWeight.Bold,
                                fontSize = 14.sp,
                                textAlign = TextAlign.Left,
                                color = if
(message.isBotMessage) MaterialTheme.colorScheme.secondary
                                else
MaterialTheme.colorScheme.primary,
                            )
                            if (message.isBotMessage &&
message.isLoading) {
                                LoadingAnimation(
                                    circleSize = 8.dp,
                                    spaceBetween = 5.dp,
                                    travelDistance = 10.dp,
                                    circleColor =
MaterialTheme.colorScheme.primary.copy(alpha = 0.7f),
                                    modifier =
Modifier.padding(top = 14.dp)
                                )
                            } else {
                                Markdown(
                                    content = message.text,
                                    colors = markdownColor(
                                        text =
LocalContentColor.current,
                                        codeText =
MaterialTheme.colorScheme.tertiaryContainer
                                    ),
                                    modifier =
Modifier.wrapContentWidth()
                                )
                            }
                            Row(
                                horizontalArrangement =
Arrangement.End,
                                modifier =
Modifier.align(Alignment.End)
                            ) {
                                Text(
                                    text = message.time,
                                    textAlign =
TextAlign.End,
                                    fontSize = 12.sp,
                                    color = Gray700
                                )
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```kotlin
package presentation.ui.component

import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.core.Animatable
import androidx.compose.animation.core.AnimationVector1D
import
androidx.compose.animation.core.FastOutLinearInEasing
import androidx.compose.animation.core.FastOutSlowInEasing
import androidx.compose.animation.core.TweenSpec
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.tween
import androidx.compose.animation.expandHorizontally
import androidx.compose.animation.fadeIn
import androidx.compose.animation.scaleIn
import androidx.compose.animation.slideInHorizontally
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.Image
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.heightIn
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.widthIn
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.graphics.Shape
import androidx.compose.ui.graphics.TransformOrigin
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.layout.onSizeChanged
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.IntOffset
import androidx.compose.ui.unit.dp
import androidx.compose.ui.zIndex
import com.preat.peekaboo.image.picker.toImageBitmap
import domain.model.Message
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.launch
```

```
private const val rotationValue = 45f

@Composable
fun MessageImagesStack(
    message: Message,
    modifier: Modifier = Modifier,
) {
    var visibility by remember { mutableStateOf(false) }

    LaunchedEffect(key1 = true) {
        visibility = true
    }
    AnimatedVisibility(
        visible = visibility,
        enter = slideInHorizontally()
                + expandHorizontally(expandFrom =
Alignment.Start)
                + scaleIn(transformOrigin =
TransformOrigin(0.5f, 0f))
                + fadeIn(initialAlpha = 0.3f),
    ) {
        Box(
            contentAlignment = if (!message.isBotMessage)
Alignment.CenterEnd else Alignment.CenterStart,
            modifier = modifier
                .padding(
                    start = if (message.isBotMessage) 0.dp
else 50.dp,
                    end = if (message.isBotMessage) 50.dp
else 0.dp,
                )
                .fillMaxWidth()
        ) {
            CardStack(
                cardCount = message.images.size,
                cardShape = RoundedCornerShape(20.dp),
                cardContent = { index ->
                    Image(
                        bitmap =
message.images[index].toImageBitmap(),
                        contentDescription = "Same Card
Type with Different Image",
                        contentScale = ContentScale.Crop,
                        modifier = Modifier
                            .heightIn(100.dp, 300.dp)
                            .widthIn(50.dp, 200.dp)
                    )
                },
                orientation = Orientation.Horizontal(
                    alignment =
HorizontalAlignment.EndToStart,
                    animationStyle =
HorizontalAnimationStyle.FromTop
                ),
            )
        }
```

```
    }
}

@Composable
fun CardStack(
    cardContent: @Composable (Int) -> Unit,
    cardCount: Int,
    cardElevation: Dp = 3.dp,
    paddingBetweenCards: Dp = 15.dp,
    animationDuration: Int = 200,
    cardShape: Shape = MaterialTheme.shapes.medium,
    cardBorder: BorderStroke? = null,
    onCardClick: ((Int) -> Unit)? = null,
    orientation: Orientation = Orientation.Vertical()
) {

    val runAnimations = animationDuration > 0
    val coroutineScope = rememberCoroutineScope()

    var selectedIndex by rememberSaveable {
mutableStateOf(0) }

    val contentAlignment = getContentAlignment(orientation)

    val rotationValue = getRotation(orientation)

    Box(contentAlignment = contentAlignment) {
        (0 until cardCount).forEachIndexed { index, _ ->
            ShowCard(
                coroutineScope,
                runAnimations,
                selectedIndex,
                index,
                cardCount,
                paddingBetweenCards,
                animationDuration,
                rotationValue,
                orientation,
                cardElevation,
                cardShape,
                cardBorder,
                onCardClick,
                { cardContent(index) },
                { selectedIndex = it })
        }
    }
}

@Composable
private fun ShowCard(
    coroutineScope: CoroutineScope,
    runAnimations: Boolean,
    selectedIndex: Int,
    index: Int,
    cardCount: Int,

    paddingBetweenCards: Dp,

    animationDuration: Int,

    rotationValue: Float,

    orientation: Orientation,

    cardElevation: Dp,

    cardShape: Shape,

    cardBorder: BorderStroke?,

    onCardClick: ((Int) -> Unit)? = null,

    composable: @Composable (Int) -> Unit,

    newIndexBlock: (Int) -> Unit

) {

    var itemPxSize = 0


    val padding = when {

        selectedIndex == index -> 0.dp

        selectedIndex < index -> ((index - selectedIndex) *
paddingBetweenCards.value).dp

        selectedIndex > index -> ((cardCount -
selectedIndex + index) * paddingBetweenCards.value).dp

        else -> throw IllegalStateException()

    }


    val paddingAnimation by animateDpAsState(padding,
tween(animationDuration, easing = FastOutSlowInEasing))

    val offsetAnimation = remember { Animatable(0f) }

    val rotateAnimation = remember { Animatable(0f) }
    val offsetValues = when (orientation) {
        is Orientation.Vertical -> {
            IntOffset(
                if (orientation.animationStyle ==
VerticalAnimationStyle.ToRight)
                    offsetAnimation.value.toInt()
                else
                    -offsetAnimation.value.toInt(), 0
            )
        }
        is Orientation.Horizontal -> {
            IntOffset(
                0, if (orientation.animationStyle ==
HorizontalAnimationStyle.FromTop)
                    -offsetAnimation.value.toInt()
                else
                    offsetAnimation.value.toInt()
            )
        }
    }

    val paddingModifier = when {
        orientation is Orientation.Vertical &&
orientation.alignment == VerticalAlignment.TopToBottom ->
PaddingValues(top = paddingAnimation)
        orientation is Orientation.Vertical &&
orientation.alignment == VerticalAlignment.BottomToTop ->
PaddingValues(bottom = paddingAnimation)
```

```
        orientation is Orientation.Horizontal &&
orientation.alignment == HorizontalAlignment.StartToEnd ->
PaddingValues(start = paddingAnimation)

        else -> PaddingValues(end = paddingAnimation)

    }


    val modifier = Modifier

        .padding(paddingModifier)

        .zIndex(-padding.value)

        .offset { offsetValues }

        .rotate(rotateAnimation.value)

        .onSizeChanged {

            itemPxSize = if (orientation is
Orientation.Vertical) {

                if (itemPxSize > it.width)

                    itemPxSize

                else

                    it.width

            } else {

                if (itemPxSize > it.height)

                    itemPxSize

                else

                    it.height

            }

        }


    Card(
        elevation =
CardDefaults.cardElevation(cardElevation),
        shape = cardShape,
        border = cardBorder,
        modifier = modifier.clickable {
            if(cardCount > 1 && selectedIndex == index) {
                onCardClick?.invoke(index)
                animateOnClick(coroutineScope, itemPxSize,
runAnimations, animationDuration, rotationValue, index,
cardCount, offsetAnimation, rotateAnimation, newIndexBlock)
            }
        }

    ) {
        composable.invoke(index)
    }
}

private fun animateOnClick(
    coroutineScope: CoroutineScope,
    pxValue: Int,
    runAnimations: Boolean,
    animationDuration: Int,
    rotationValue: Float,
    index: Int,
    cardCount: Int,
    offsetAnimation: Animatable<Float, AnimationVector1D>,
    rotateAnimation: Animatable<Float, AnimationVector1D>,
    newIndexBlock: (Int) -> Unit
) {
    val spec: TweenSpec<Float> = tween(animationDuration,
easing = FastOutLinearInEasing)

    coroutineScope.launch {
        if (runAnimations)
            offsetAnimation.animateTo(pxValue.toFloat(),
spec)

        val newIndex = if (cardCount > index + 1)
            index + 1
        else
            0

        newIndexBlock.invoke(newIndex)

        if(runAnimations) {
            rotateAnimation.animateTo(rotationValue, spec)
            launch { rotateAnimation.animateTo(0f, spec) }
            launch { offsetAnimation.animateTo(0f, spec) }
        }
    }
}


sealed class Orientation {
    data class Vertical(val alignment: VerticalAlignment =
VerticalAlignment.TopToBottom,
                        val animationStyle:
VerticalAnimationStyle = VerticalAnimationStyle.ToRight
    ) : Orientation()


    data class Horizontal(val alignment:
HorizontalAlignment = HorizontalAlignment.StartToEnd,
                          val animationStyle:
HorizontalAnimationStyle = HorizontalAnimationStyle.FromTop
    ) : Orientation()
}


enum class VerticalAlignment {
    TopToBottom,
    BottomToTop,
}


enum class HorizontalAlignment {
    StartToEnd,
    EndToStart,
}


enum class VerticalAnimationStyle {
    ToRight,
    ToLeft,
}
```

```
enum class HorizontalAnimationStyle {
    FromTop,
    FromBottom
}

private fun getContentAlignment(orientation: Orientation):
Alignment {
    return when(orientation) {
        is Orientation.Vertical -> {
            if(orientation.alignment ==
VerticalAlignment.TopToBottom)
                Alignment.TopCenter
            else
                Alignment.BottomCenter
        }

        is Orientation.Horizontal -> {
            if(orientation.alignment ==
HorizontalAlignment.StartToEnd)
                Alignment.CenterStart
            else
                Alignment.CenterEnd
        }
    }
}

private fun getRotation(orientation: Orientation): Float {
    return when(orientation) {
        is Orientation.Vertical ->
if(orientation.animationStyle ==
VerticalAnimationStyle.ToRight)
            rotationValue
        else
            -rotationValue
        is Orientation.Horizontal ->
if(orientation.animationStyle ==
HorizontalAnimationStyle.FromTop)
            -rotationValue
        else
            rotationValue
    }
}
```

```
package presentation.ui.extension

import androidx.compose.material3.SnackbarHostState
import domain.model.Status
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.launch

fun SnackbarHostState.showSnackBar(scope: CoroutineScope,
status: Status) {
    if (status is Status.Error) {
```

```
                scope.launch {
                    showSnackbar(
                        message = status.message,
                        withDismissAction = true
                    )
                }
            } else {
                this.currentSnackbarData?.dismiss()
            }
        }
        package presentation.ui.screen

        import
        androidx.compose.foundation.gestures.detectTapGestures
        import androidx.compose.foundation.layout.Arrangement
        import androidx.compose.foundation.layout.PaddingValues
        import androidx.compose.foundation.layout.Spacer
        import androidx.compose.foundation.layout.fillMaxWidth
        import androidx.compose.foundation.layout.height
        import androidx.compose.foundation.layout.padding
        import androidx.compose.foundation.lazy.LazyColumn
        import
        androidx.compose.foundation.lazy.rememberLazyListState
        import androidx.compose.material3.Scaffold
        import androidx.compose.material3.SnackbarHost
        import androidx.compose.material3.SnackbarHostState
        import androidx.compose.runtime.Composable
```

```
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.unit.dp
import domain.model.Message
import domain.model.Status
import kotlinx.coroutines.launch
import presentation.theme.LightGreen
import presentation.theme.LightRed
import presentation.ui.component.CustomAppBar
import presentation.ui.component.CustomBottomBar
import presentation.ui.component.CustomDialog
import presentation.ui.component.CustomSnackBar
import presentation.ui.component.MessageBubble
import presentation.ui.component.MessageImagesStack
import presentation.ui.extension.showSnackBar


@Composable
fun ChatScreen(viewModel: ChatViewModel = ChatViewModel())
{
    val chatUiState = viewModel.uiState
    val focusManager = LocalFocusManager.current
    val coroutineScope = rememberCoroutineScope()
    val apiKeySnackBarHostState = remember {
SnackbarHostState() }
    val errorSnackBarHostState = remember {
SnackbarHostState() }
    val showDialog = remember { mutableStateOf(false) }


    Scaffold(
        topBar = {
            CustomAppBar(onActionClick = { showDialog.value
= true })
        },
        bottomBar = {
            CustomBottomBar(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 10.dp)
                    .padding(bottom = 30.dp, top = 5.dp),
                status = chatUiState.value.status,
                onSendClick = { text, images ->
                    coroutineScope.launch {
                        viewModel.generateContent(text,
images)
                    }
                },
            )
        },
        snackbarHost = {
            SnackbarHost(errorSnackBarHostState) { data ->
```

```
                CustomSnackBar(
                    data = data,
                    containerColor = LightRed
                )
            }
            SnackbarHost(apiKeySnackBarHostState) { data ->
                CustomSnackBar(
                    data = data,
                    containerColor = LightGreen
                )
            }
        },
        modifier = Modifier.pointerInput(Unit) {
            detectTapGestures(onTap = {
focusManager.clearFocus() })
        },
    ) { paddingValues ->
        ChatList(
            modifier = Modifier.padding(paddingValues),
            messages = chatUiState.value.messages
        )

        if (showDialog.value) {
            CustomDialog(
                value = chatUiState.value.apiKey,
                onVisibilityChanged = { showDialog.value =
it },
                onSaveClicked = {
                    coroutineScope.launch {
                        viewModel.setApiKey(it)

apiKeySnackBarHostState.currentSnackbarData?.dismiss()
                        if(chatUiState.value.status is
Status.Success){

apiKeySnackBarHostState.showSnackbar(
                                message =
(chatUiState.value.status as Status.Success).data,
                                withDismissAction = true
                            )
                        }


                    }
                }
            )
        }


        errorSnackBarHostState.showSnackBar(coroutineScope,
chatUiState.value.status)
    }
}



@Composable
fun ChatList(modifier: Modifier, messages: List<Message>) {
    val listState = rememberLazyListState()

    if (messages.isNotEmpty()) {
        LaunchedEffect(messages) {
listState.animateScrollToItem(messages.lastIndex)
        }
    }
    LazyColumn(
        state = listState,
        modifier = modifier.fillMaxWidth(),
        contentPadding = PaddingValues(10.dp),
        verticalArrangement = Arrangement.spacedBy(10.dp)
    ) {
        items(messages.size) {
            val message = messages[it]
            if (message.images.isNotEmpty()) {
                MessageImagesStack(message = message)
                Spacer(modifier = Modifier.height(4.dp))
            }
            MessageBubble(message = message)
        }
    }
}
package presentation.ui.screen

import domain.model.Message
import domain.model.Status

data class ChatUiState(
    val messages: List<Message> = emptyList(),
    val status: Status = Status.Idle,
    val apiKey: String = ""
)
package presentation.ui.screen

import androidx.compose.runtime.State
import androidx.compose.runtime.mutableStateOf
import data.repository.GeminiRepositoryImpl
import dev.icerock.moko.mvvm.viewmodel.ViewModel
import domain.repository.GeminiRepository
import domain.model.Message
import domain.model.Sender
import domain.model.Status
import kotlinx.coroutines.launch

class ChatViewModel : ViewModel() {

    private val geminiRepository: GeminiRepository =
GeminiRepositoryImpl()

    private val _uiState = mutableStateOf(ChatUiState())
    val uiState: State<ChatUiState> = _uiState
    init {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(apiKey =
geminiRepository.getApiKey())
        }
    }

    fun setApiKey(key: String) {
        geminiRepository.setApiKey(key)
        _uiState.value = _uiState.value.copy(apiKey = key,
status = Status.Success("API key updated successfully."))
    }

    fun generateContent(message: String, images:
List<ByteArray> = emptyList()) {
        viewModelScope.launch {
            addToMessages(message, images, Sender.User)
            addToMessages("", emptyList(), Sender.Bot,
true)

            when (val response =
geminiRepository.generate(message, images)) {
                is Status.Success ->
updateLastBotMessage(response.data, response)
                is Status.Error ->
updateLastBotMessage(response.message, response)
                else -> {}
            }
        }
    }


    private fun updateLastBotMessage(text: String, status:
Status) {
        val messages =
_uiState.value.messages.toMutableList()
        if (messages.isNotEmpty() && messages.last().sender
== Sender.Bot) {
            val last = messages.last()
            val updatedMessage = last.copy(text = text,
isLoading = status == Status.Loading)
            messages[messages.lastIndex] = updatedMessage
            _uiState.value = _uiState.value.copy(
                messages = messages,
                status = status
            )
        }
    }


    private fun addToMessages(
        text: String,
        images: List<ByteArray>,
        sender: Sender,
        isLoading: Boolean = false
    ) {
        val message = Message(sender, text, images,
isLoading)
        _uiState.value = _uiState.value.copy(
            messages = _uiState.value.messages + message,
            status = if (isLoading) Status.Loading else
Status.Idle
        )
    }
}

import androidx.compose.runtime.Composable
import presentation.theme.GeminiTalkerTheme
import presentation.ui.screen.ChatScreen

@Composable
fun App() {
    GeminiTalkerTheme {
        ChatScreen()
    }
}

expect fun getPlatformName(): String
package data.preferences

import java.io.File
import java.io.FileInputStream
import java.io.FileOutputStream
import java.util.Properties

actual class PreferencesStorage {
    private val properties = Properties()
    private val propertiesFile =
File("GeminiTalker.properties")

    init {
        if (propertiesFile.exists()) {

properties.load(FileInputStream(propertiesFile))
        }
    }

    actual fun getString(key: String, defaultValue:
String): String =
        properties.getProperty(key, defaultValue)

    actual fun putString(key: String, value: String) {
        properties.setProperty(key, value)
        properties.store(FileOutputStream(propertiesFile),
null)
    }
}
package presentation.theme

import androidx.compose.runtime.Composable

@Composable
internal actual fun SystemAppearance(isDark: Boolean) {
}
```

```
import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.runtime.Composable

actual fun getPlatformName(): String = "Desktop"

@Composable fun MainView() = App()

@Preview
@Composable
fun AppPreview() {
    App()
}
package data.preferences

import platform.Foundation.NSUserDefaults

actual class PreferencesStorage {
    private val userDefaults =
NSUserDefaults.standardUserDefaults

    actual fun getString(key: String, defaultValue:
String): String =
        userDefaults.stringForKey(key) ?: defaultValue

    actual fun putString(key: String, value: String) {
        userDefaults.setObject(value, forKey = key)
    }
}
package presentation.theme


import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import platform.UIKit.UIApplication
import platform.UIKit.UIStatusBarStyleDarkContent
import platform.UIKit.UIStatusBarStyleLightContent
import platform.UIKit.setStatusBarStyle


@Composable
internal actual fun SystemAppearance(isDark: Boolean) {
    LaunchedEffect(isDark) {
        UIApplication.sharedApplication.setStatusBarStyle(
            if (isDark) UIStatusBarStyleDarkContent else
UIStatusBarStyleLightContent
        )
    }
}
import androidx.compose.ui.window.ComposeUIViewController


actual fun getPlatformName(): String = "iOS"


fun MainViewController() = ComposeUIViewController { App()
}
package data.preferences
```

```
import android.content.Context
import android.content.SharedPreferences
import data.preferences.PrefKey.GEMINI_TALKER_PREF

actual class PreferencesStorage(context: Context) {
    private val sharedPreferences: SharedPreferences =
        context.getSharedPreferences(GEMINI_TALKER_PREF,
Context.MODE_PRIVATE)

    actual fun getString(key: String, defaultValue:
String): String =
        sharedPreferences.getString(key, defaultValue) ?:
defaultValue

    actual fun putString(key: String, value: String) {
        sharedPreferences.edit().putString(key,
value).apply()
    }
}
package presentation.theme

import android.app.Activity
import android.graphics.Color
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.ui.platform.LocalView
import androidx.core.view.WindowCompat
```

```
@Composable
internal actual fun SystemAppearance(isDark: Boolean) {
    val view = LocalView.current
    val systemBarColor = Color.TRANSPARENT
    LaunchedEffect(isDark) {
        val window = (view.context as Activity).window
        WindowCompat.setDecorFitsSystemWindows(window,
false)
        window.statusBarColor = systemBarColor
        window.navigationBarColor = systemBarColor
        WindowCompat.getInsetsController(window,
window.decorView).apply {
            isAppearanceLightStatusBars = isDark
            isAppearanceLightNavigationBars = isDark
        }
    }
}
import androidx.compose.runtime.Composable

actual fun getPlatformName(): String = "Android"

@Composable fun MainView() = App()
```