**import** Foundation

**import** FirebaseFirestore

**import** FirebaseStorage

**class** FirestoreManager {

**static** **let** shared = FirestoreManager()

**private** **let** db = Firestore.firestore()

**private** **let** storage = Storage.storage()

**private** **init**() {}

// **MARK: - Klienten-Methoden**

**func** getClients(completion: **@escaping** (Result<[Client], Error>) -> Void) {

db.collection("clients").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** clients = documents.compactMap { **try**? $0.data(as: Client.**self**) }

completion(.success(clients))

}

}

**func** createClient(client: Client, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("clients").addDocument(from: client) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** updateClient(client: Client, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = client.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Client-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("clients").document(id).setData(from: client) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteClient(clientID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("clients").document(clientID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Funktionär-Methoden**

**func** getFunktionäre(completion: **@escaping** (Result<[Funktionär], Error>) -> Void) {

db.collection("funktionare").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** funktionäre = documents.compactMap { **try**? $0.data(as: Funktionär.**self**) }

completion(.success(funktionäre))

}

}

**func** createFunktionär(funktionär: Funktionär, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("funktionare").addDocument(from: funktionär) { error **in**

**if** **let** error = error {

print("Create failed with error: \(error.localizedDescription)")

completion(.failure(error))

} **else** {

print("Funktionär successfully created")

completion(.success(()))

}

}

} **catch** {

print("Encoding error: \(error.localizedDescription)")

completion(.failure(error))

}

}

**func** updateFunktionär(funktionär: Funktionär, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = funktionär.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Funktionär-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("funktionare").document(id).setData(from: funktionär) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteFunktionär(funktionärID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("funktionare").document(funktionärID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Vereins-Methoden**

**func** getClubs(completion: **@escaping** (Result<[Club], Error>) -> Void) {

db.collection("clubs").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** clubs = documents.compactMap { **try**? $0.data(as: Club.**self**) }

completion(.success(clubs))

}

}

**func** createClub(club: Club, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("clubs").addDocument(from: club) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** updateClub(club: Club, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = club.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Club-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("clubs").document(id).setData(from: club) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteClub(clubID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("clubs").document(clubID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Vertrag-Methoden**

**func** getContracts(completion: **@escaping** (Result<[Contract], Error>) -> Void) {

db.collection("contracts").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** contracts = documents.compactMap { **try**? $0.data(as: Contract.**self**) }

completion(.success(contracts))

}

}

**func** getContract(forClientID clientID: String, completion: **@escaping** (Result<Contract?, Error>) -> Void) {

db.collection("contracts")

.whereField("clientID", isEqualTo: clientID)

.limit(to: 1)

.getDocuments { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents, !documents.isEmpty **else** {

completion(.success(**nil**))

**return**

}

**if** **let** contract = **try**? documents[0].data(as: Contract.**self**) {

completion(.success(contract))

} **else** {

completion(.success(**nil**))

}

}

}

**func** createContract(contract: Contract, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("contracts").addDocument(from: contract) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** updateContract(contract: Contract, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = contract.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Contract-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("contracts").document(id).setData(from: contract) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteContract(contractID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("contracts").document(contractID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Transfer-Methoden**

**func** getTransfers(completion: **@escaping** (Result<[Transfer], Error>) -> Void) {

db.collection("transfers").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** transfers = documents.compactMap { **try**? $0.data(as: Transfer.**self**) }

completion(.success(transfers))

}

}

**func** createTransfer(transfer: Transfer, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("transfers").addDocument(from: transfer) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** updateTransfer(transfer: Transfer, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = transfer.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Transfer-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("transfers").document(id).setData(from: transfer) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteTransfer(transferID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("transfers").document(transferID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Match-Methoden**

**func** getMatches(completion: **@escaping** (Result<[Match], Error>) -> Void) {

db.collection("matches").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** matches = documents.compactMap { **try**? $0.data(as: Match.**self**) }

completion(.success(matches))

}

}

**func** createMatch(match: Match, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("matches").addDocument(from: match) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** updateMatch(match: Match, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = match.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Match-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("matches").document(id).setData(from: match) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteMatch(matchID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("matches").document(matchID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Sponsor-Methoden**

**func** getSponsors(completion: **@escaping** (Result<[Sponsor], Error>) -> Void) {

db.collection("sponsors").addSnapshotListener { querySnapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = querySnapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** sponsors = documents.compactMap { **try**? $0.data(as: Sponsor.**self**) }

completion(.success(sponsors))

}

}

**func** createSponsor(sponsor: Sponsor, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("sponsors").addDocument(from: sponsor) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** updateSponsor(sponsor: Sponsor, completion: **@escaping** (Result<Void, Error>) -> Void) {

**guard** **let** id = sponsor.id **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Kein Sponsor-ID verfügbar"])))

**return**

}

**do** {

**try** db.collection("sponsors").document(id).setData(from: sponsor) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** deleteSponsor(sponsorID: String, completion: **@escaping** (Result<Void, Error>) -> Void) {

db.collection("sponsors").document(sponsorID).delete { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

// **MARK: - Profilbild-Methoden**

**func** uploadProfileImage(documentID: String, image: UIImage, collection: String = "profile\_images", completion: **@escaping** (Result<String, Error>) -> Void) {

**guard** **let** imageData = image.jpegData(compressionQuality: 0.8) **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Bild konnte nicht komprimiert werden"])))

**return**

}

**let** storageRef = storage.reference().child("\(collection)/\(documentID).jpg")

storageRef.putData(imageData, metadata: **nil**) { metadata, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

storageRef.downloadURL { url, error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** **if** **let** url = url {

completion(.success(url.absoluteString))

} **else** {

completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "URL konnte nicht abgerufen werden"])))

}

}

}

}

// **MARK: - Aktivitäten-Methoden**

**func** createActivity(activity: Activity, completion: **@escaping** (Result<Void, Error>) -> Void) {

**do** {

**try** db.collection("activities").addDocument(from: activity) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** getActivities(forClientID clientID: String, completion: **@escaping** (Result<[Activity], Error>) -> Void) {

db.collection("activities")

.whereField("clientID", isEqualTo: clientID)

.order(by: "timestamp", descending: **true**)

.getDocuments { snapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** **if** **let** snapshot = snapshot {

**let** activities = snapshot.documents.compactMap { **try**? $0.data(as: Activity.**self**) }

completion(.success(activities))

}

}

}

// **MARK: - Chat-Methoden**

**func** createOrUpdateChat(chat: Chat, completion: **@escaping** (Result<String, Error>) -> Void) {

**let** ref: DocumentReference

**if** **let** id = chat.id {

ref = db.collection("chats").document(id)

} **else** {

ref = db.collection("chats").document()

}

**do** {

**try** ref.setData(from: chat) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(ref.documentID))

}

}

} **catch** {

completion(.failure(error))

}

}

**func** sendMessage(chatID: String, message: Message, completion: **@escaping** (Result<Void, Error>) -> Void) {

**let** chatRef = db.collection("chats").document(chatID)

**let** messagesRef = chatRef.collection("messages").document()

**do** {

**try** messagesRef.setData(from: message) { error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

chatRef.updateData([

"lastMessage": message.content,

"lastMessageTimestamp": message.timestamp

]) { error **in**

**if** **let** error = error {

completion(.failure(error))

} **else** {

completion(.success(()))

}

}

}

} **catch** {

completion(.failure(error))

}

}

**func** getChats(forUserID userID: String, completion: **@escaping** (Result<[Chat], Error>) -> Void) {

db.collection("chats")

.whereField("participantIDs", arrayContains: userID)

.order(by: "lastMessageTimestamp", descending: **true**)

.addSnapshotListener { snapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = snapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** chats = documents.compactMap { **try**? $0.data(as: Chat.**self**) }

completion(.success(chats))

}

}

**func** getMessages(forChatID chatID: String, completion: **@escaping** (Result<[Message], Error>) -> Void) {

db.collection("chats").document(chatID).collection("messages")

.order(by: "timestamp", descending: **false**)

.addSnapshotListener { snapshot, error **in**

**if** **let** error = error {

completion(.failure(error))

**return**

}

**guard** **let** documents = snapshot?.documents **else** {

completion(.success([]))

**return**

}

**let** messages = documents.compactMap { **try**? $0.data(as: Message.**self**) }

completion(.success(messages))

}

}

}