Integration Tests

1. Test: Verify user can login
    - Tap on the continue with telegram button button, a webview of telegram sign in should show up
    - Fill in your telegram credentails
    - Should be redirected to show your trips
2. Test: Verify user can create an event
    - Tap on the create event button
    - Fill in required details
    - Tap on the create button
    - Verify the event is created
3. Test: Verify user can delete an event
    - Navigate to the event details page
    - Tap on the delete button
    - Verify the event is deleted
4. Test: Verify user can view events
    - Navigate to the events page
    - Verify the events that the user has hosted or invited to are displayed
5. Test: Verify user can view their profile
    - Navigate to the profile page
    - Verify the user's profile information is displayed
6. Test: Verify user can update their profile
    - Navigate to the profile page
    - Tap on the edit button
    - Update the required details
    - Tap on the save button
    - Verify the changes are saved
7. Test: Verify user can view their friends
    - Navigate to the profile page
    - Navigate to the friends page by clicking "See All"
    - Verify the user's friends are displayed
8. Test: Verify user can search for events by date
    - Navigate to the events page
    - Tap on a date

    - Verify that events for the date are shown

9. Test: Verify user can view event details
    - Navigate to the events page
    - Tap on an event
    - Verify the event details are displayed
10. Test: Verify user can RSVP to an event
    - Navigate to the event details page
    - Tap on the "Are you going" button
    - Tap on the Going menu tiem
    - Verify that Going is shown on the event
11. Test: Verify user can chat with friends going to the same invite
    - Navigate to the chat page
    - Tap on a invite
    - Send a message
    - Verify the message is sent and displayed to all the friends going on the same invite
12. Test: Verify user can view their trip
    - Navigate to profile tab
    - Navigate to the trips page by scrolling down and clicking switch
    - Verify the user's trips are displayed
13. Test: Verify the realtime map displays all users in the trip
    - Navigate to the trip containing multiple users
    - Open the realtime map
    - Verify all users in the trip are displayed on the map with their respective profile images
14. Test: Verify user profile sheet appears when clicking on a user icon
    - Open the realtime map with multiple users
    - Click on a user's icon
    - Verify a sheet appears with the user's profile information
15. Test: Verify location sharing can be toggled on and off using the radar icon
    - Open the realtime map with multiple users
    - Click the radar icon to toggle location sharing on or off
    - Verify the user's location sharing status is updated accordingly
    - Repeat the process to ensure location sharing can be toggled on and off successfully

Unit tests

ChatRepositoryFirebaseTests

1. setUp Initialize ChatRepositoryFirebase instance, expect all properties to be initialized with correct values.
2. getBasicInfo Mock Database to return valid Chat snapshot, call method, expect completion to return correct Chat instance with basic information.
3. getChatMessageAM Mock Database to return valid ChatMessageApiModel snapshot, call method, expect completion to return correct ChatMessageApiModel instance.
4. getChat Mock Database to return valid ChatMessageApiModels snapshot, call method, expect completion to return correct Chat instance with chat messages.
5. sendChatMessage Create valid ChatMessage instance, call method, expect message to be added to the database and chat basic info to be updated.
6. setListenerForChatMessages Mock Database to trigger .childAdded event, call method, expect completion to return new ChatMessage instance on every new message.
7. removeListenerForChatMessages Call method, expect observers to be removed from the "messages" path in the database.
8. setListenerForChatBasicInfo Mock Database to trigger .value event, call method, expect completion to return updated Chat instance on every change in basic information.
9. removeListenerForChatBasicInfo Call method, expect observers to be removed from the "chats" path in the database.
10. getChatPreview Mock Database to return valid Chat snapshot with basic information and last 3 messages, call method, expect completion to return correct Chat instance with a preview of the last 3 messages.

UserServiceTests

1. setUp Initialize UserService instance, expect all properties to be initialized with correct values.
2. getUser Retrieve User from cache, call method, expect retrieved User to match stored User.
3. storeUserId Store a valid UUID, call method, expect UserDefaults to store the correct UUID string.
4. logout Call method, expect isLoggedIn to be false, observers to be empty, and UserDefaults to reset related values.
5. emailSignin Mock UserRepository to return valid User, call method, expect completion to return true with fetched User's UUID.
6. emailSigninFailed Mock UserRepository to throw an error, call method, expect completion to return false with nil UUID.
7. fetchUser Mock UserRepository to return valid User, call method, expect completion to return true and User to be initialized in cache.
8. fetchUserFailed Mock UserRepository to throw an error, call method, expect completion to return false and User to not be initialized in cache.

9. updateUser Prepare User instance with modified values, call method, expect UserRepository to update User and UserService to notify observers.
10. reloadUser Call method, expect UserRepository to fetch User and UserService to notify observers.
11. editUser Call method with new values, expect User instance to be updated with new values.
12. clear Call method, expect observers and cache to be cleared.
13. changeTrip Call method, expect observers to be cleared and cache to be cleared.
14. getProfileHeaderViewModel Call method, expect a valid ProfileHeaderViewModel instance to be returned.
15. getEditProfileViewModel Call method, expect a valid EditProfileViewModel instance to be returned.
16. getSocialMediaLinksViewModel Call method, expect a valid SocialMediaLinksViewModel instance to be returned.

MapStorageServiceTests

1. setUp Initialize MapStorageService instance with a mock MapRepository, expect all properties to be initialized with correct values.
2. startUserLocationUpdate Initialize LocationManagerService, call method with valid userId, expect LocationManagerService locationUpdateHandler to be set, and MapRepository.updateCurrentUserLocation to be called when the location is updated.
3. stopUserLocationUpdate Initialize LocationManagerService, call method with valid userId, expect LocationManagerService locationUpdateHandler to be set to nil, and MapRepository.removeCurrentUserLocation to be called with the provided userId.
4. fetchFriendsLocations Mock MapRepository to return a dictionary of friends locations, call method, expect friendsLocations to be updated with the fetched locations.

LocationManagerServiceTests

1. setUp Initialize LocationManagerService instance, expect all properties to be initialized with correct values, and locationManager to have correct configuration.
2. locationManager*didUpdateLocations Call locationManager(*:didUpdateLocations:), expect userLocation to be updated with the last location from the provided locations array, and locationUpdateHandler to be called with the same location.
3. startUpdatingLocation Call method, expect locationManager to start updating location, and allowsBackgroundLocationUpdates to be true.
4. stopUpdatingLocation Call method, expect locationManager to stop updating location, and allowsBackgroundLocationUpdates to be false.
5. configureLocationManagerForBackgroundUpdates Call method, expect locationManager to have correct configuration for background updates.

TripServiceTests

1. setUp Initialize TripService instance, expect all properties to be initialized with correct values.
2. getTripViewModel Call method with a valid tripId, expect a valid TripViewModel instance to be returned.
3. getCurrTrip Call method after setting selectedTrip, expect the correct Trip instance to be returned.
4. getCurrTripId Call method after setting selectedTrip, expect the correct tripId to be returned.
5. resetTapIndex Call method, expect selectedTapInCurrTrip to be reset to 0.
6. loadUserTrips Mock TripRepository to return a list of trips, call method with valid userId, expect cache to be initialized with fetched trips and completion handler to return true.
7. loadUserTripsFailed Mock TripRepository to throw an error, call method with valid userId, expect cache to remain empty and completion handler to return false.
8. reloadUserTrips Mock TripRepository to return a list of updated trips, call method with valid userId, expect cache and view models to be updated with fetched trips and completion handler to return true.
9. reloadUserTripsFailed Mock TripRepository to throw an error, call method with valid userId, expect cache and view models to remain unchanged and completion handler to return false.
10. selectTrip Call method with a valid Trip instance, expect selectedTrip to be set to the provided instance.
11. clear Call method, expect selectedTrip and lastSelectedTripId to be nil, observers and cache to be cleared.

FriendServiceTests

1. setUp Initialize FriendService instance, expect all properties to be initialized with correct values.
2. getProfileFriendsViewModel Call method, expect a valid ProfileFriendsViewModel instance to be returned.
3. getCreateInvitePageViewModel Call method, expect a valid CreateInvitePageViewModel instance to be returned.
4. fetchAllFriends Mock TripRepository to return a list of users, call method with valid tripId and userId, expect cache to be initialized with fetched friends excluding the current user.
5. fetchAllFriendsFailed Mock TripRepository to throw an error, call method with valid tripId and userId, expect cache to remain empty.
6. reloadFriends Mock TripRepository to return an updated list of users, call method with valid tripId and userId, expect cache to be updated with fetched friends excluding the current user and completion handler to return true.

7. reloadFriendsFailed Mock TripRepository to throw an error, call method with valid tripId and userId, expect cache to remain unchanged and completion handler to return false.
8. createTempFriendsSocialMediaLinkVM Call method with a valid User instance, expect a valid SocialMediaLinksViewModel instance to be returned with correct values.
9. clear Call method, expect tasks to be canceled, cache and observers to be cleared.

EventServiceTests

1. setUp Initialize EventService instance, expect all properties to be initialized with correct values.
2. getEventViewModel Call method with a valid eventId, expect a valid EventViewModel instance to be returned.
3. loadUserEvents Mock EventRepository to return a list of events, call method with valid tripId and userId, expect cache to be initialized with fetched events excluding cancelled events.
4. loadUserEventsFailed Mock EventRepository to throw an error, call method with valid tripId and userId, expect cache to remain empty.
5. reloadUserEvents Mock EventRepository to return an updated list of events, call method with valid tripId and userId, expect cache to be updated with fetched events excluding cancelled events.
6. createEvent Mock EventRepository to return a created event, call method with a valid event, expect cache to be updated and completion handler to return the created event.
7. cancelEvent Mock EventRepository to successfully cancel an event, call method with a valid eventId, expect the event to be removed from the cache, its observers to be removed, and completion handler to return true.
8. cancelEventFailed Mock EventRepository to throw an error, call method with a valid eventId, expect cache to remain unchanged and completion handler to return false.
9. rsvpToEvent Mock EventRepository to successfully RSVP to an event, call method with a valid eventId and userId, expect the event's attendingUsers or rejectedUsers to be updated accordingly.
10. findAttendingEventsId Call method with a valid user, expect a list of attending event IDs to be returned.
11. findAttendingEventsVM Call method with a valid user, expect a list of attending EventViewModel instances to be returned.
12. clear Call method, expect cache and observers to be cleared.

ChatServiceTests

1. setUp Initialize ChatService instance, expect all properties to be initialized with correct values.
2. setUserId Call method with a valid UUID, expect userId to be set properly.
3. fetchChatPageCells Mock ChatRepository to return a list of basic chat information, call method with valid chat IDs, expect chatPageCellVMs to be populated with fetched chat information.

4. fetchChatPageCell Call method with a valid chat ID, expect chatPageCellVMs to be updated with the fetched chat information.
5. removeChatPageCell Call method with a valid chat ID, expect chatPageCellVMs to have the corresponding chat removed and the listener for the chat to be removed.
6. fetchChat Mock ChatRepository to return a chat object, call method with a valid chat ID, expect chatHeaderVM, chatMessageVMs, and chat listeners to be initialized with fetched chat data.
7. removeChatListener Call method, expect chat listener to be removed.
8. sendChatMessage Mock ChatRepository to return a successful send operation, call method with a valid message text, expect a return value of true.
9. shouldShowTimestampAboveMessage Call method with a valid ChatMessageViewModel, expect a boolean indicating whether the timestamp should be shown above the message.
10. getEventPagePreview Mock ChatRepository to return a chat object, call method with a valid event ID, expect chatPreviewVM to be updated with fetched chat data.
11. clear Call method, expect all listeners to be removed and all stored view models to be reset.