Aim: To apply navigation, routing and gestures in Flutter application.

Theory:
Navigation: Navigation refers to the process of moving between different screens or "routes" within the app. Flutter provides the Navigator class, which manages a stack of routes and facilitates navigation between them. You can push new routes onto the stack using Navigator.push() and remove routes using Navigator.pop(). Named routes can be pre-defined in the app's route table, making it easier to navigate to specific screens by providing their names. Nested navigation allows for hierarchical navigation structures, such as tab-based navigation or modal dialogs.

Routing: Routing involves defining and managing the routes or paths that users can take through the app. Routes are logical representations of screens or pages within the app and are associated with unique identifiers (route names or route keys). Route management includes defining routes, specifying transitions between routes, passing data between routes, and handling route navigation events.

Gestures: Gestures enable users to interact with the app's UI elements through touch-based interactions. Flutter provides a wide range of gesture recognizers, such as GestureDetector, InkWell, Draggable, LongPressGestureDetector, etc., to detect and respond to user gestures. Gesture recognizers can detect taps, swipes, drags, pinches, and other touch-based actions, allowing for rich and intuitive user interactions. You can customize gesture behaviors, such as sensitivity, velocity, and directionality, to meet specific app requirements. By implementing navigation, routing, and gestures effectively in your Flutter app, you can create a smooth and engaging user experience, enabling users to navigate between screens, interact with UI elements, and perform actions with ease.

Code:
```
import 'dart:async';
import 'dart:convert';
import 'dart:io';

// Flutter imports:
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';

// Package imports:
import 'package:redux/redux.dart';
import 'package:redux_logging/redux_logging.dart';
import 'package:sentry_flutter/sentry_flutter.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:invoiceninja_flutter/utils/platforms.dart';
void main({bool isTesting = false}) async {
  WidgetsFlutterBinding.ensureInitialized();
```

```
 _registerErrorHandlers();

 try {
   SecurityContext.defaultContext.setTrustedCertificatesBytes(
     Uint8List.fromList(isrgRootX1.codeUnits),
   );
 } catch (e) {
   // Ignore CERT_ALREADY_IN_HASH_TABLE
 }

 final prefs = await SharedPreferences.getInstance();

 if (isDesktopOS()) {
   await windowManager.ensureInitialized();

   windowManager.waitUntilReadyToShow(
       WindowOptions(
         center: true,
         size: Size(
           prefs.getDouble(kSharedPrefWidth) ?? 800,
           prefs.getDouble(kSharedPrefHeight) ?? 600,
         ),
       ), () async {
     await windowManager.show();
     await windowManager.focus();
   });
 }

 final store = Store<AppState>(appReducer,
     initialState: await _initialState(isTesting, prefs),
     middleware: []
       ..addAll(createStoreAuthMiddleware())
       ..addAll(createStoreDocumentsMiddleware())
       ..addAll(createStoreDashboardMiddleware())
       ..addAll(createStoreProductsMiddleware())
       ..addAll(createStoreClientsMiddleware())
       ..addAll(createStoreInvoicesMiddleware())
       ..addAll(createStoreExpensesMiddleware())
       ..addAll(createStoreVendorsMiddleware())
       ..addAll(createStoreTasksMiddleware())
       ..addAll(createStoreProjectsMiddleware())
       ..addAll(createStorePaymentsMiddleware())
       ..addAll(createStoreQuotesMiddleware())
       ..addAll(createStoreSettingsMiddleware())
```

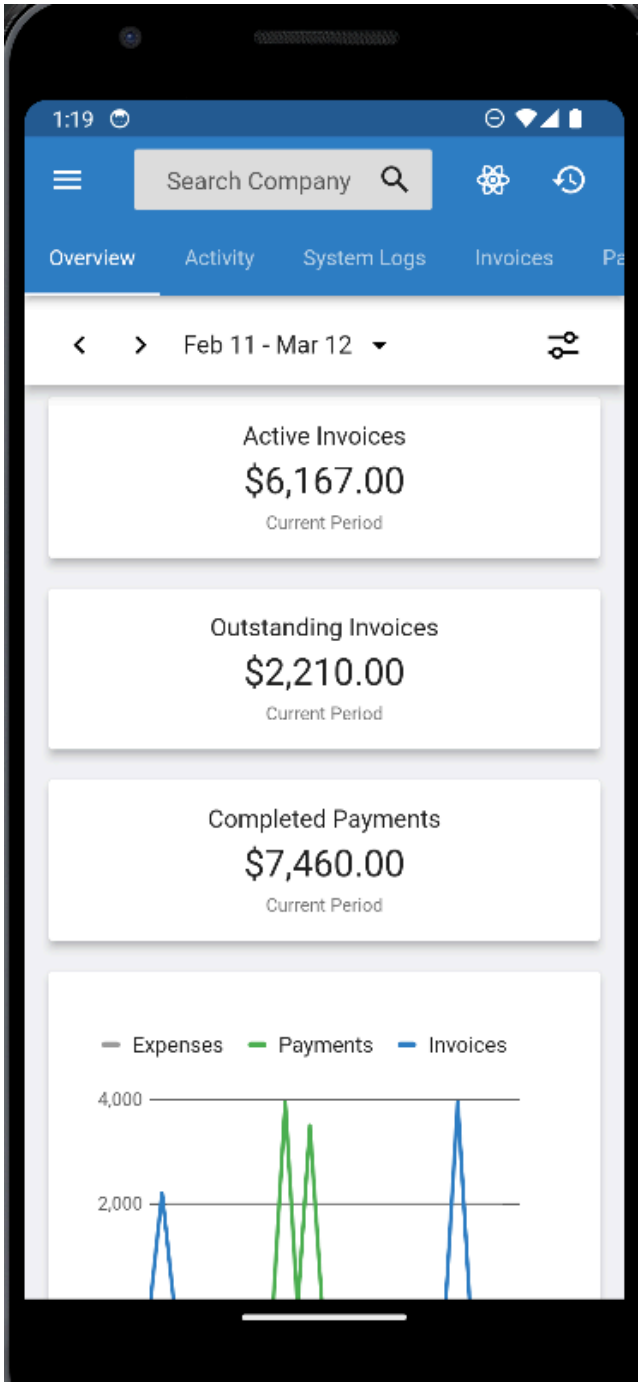
```
      ..addAll(createStoreReportsMiddleware())
      // STARTER: middleware - do not remove comment
      ..addAll(createStoreSchedulesMiddleware())
      ..addAll(createStoreTransactionRulesMiddleware())
      ..addAll(createStoreTransactionsMiddleware())
      ..addAll(createStoreBankAccountsMiddleware())
      ..addAll(createStorePurchaseOrdersMiddleware())
      ..addAll(createStoreRecurringExpensesMiddleware())
      ..addAll(createStoreSubscriptionsMiddleware())
      ..addAll(createStoreTaskStatusesMiddleware())
      ..addAll(createStoreExpenseCategoriesMiddleware())
      ..addAll(createStoreRecurringInvoicesMiddleware())
      ..addAll(createStoreWebhooksMiddleware())
      ..addAll(createStoreTokensMiddleware())
      ..addAll(createStorePaymentTermsMiddleware())
      ..addAll(createStoreDesignsMiddleware())
      ..addAll(createStoreCreditsMiddleware())
      ..addAll(createStoreUsersMiddleware())
      ..addAll(createStoreTaxRatesMiddleware())
      ..addAll(createStoreCompanyGatewaysMiddleware())
      ..addAll(createStoreGroupsMiddleware())
      ..addAll(createStorePersistenceMiddleware())
      ..addAll(isTesting || kReleaseMode || !Config.DEBUG_EVENTS
          ? []
          : [
              LoggingMiddleware<dynamic>.printer(
                formatter: LoggingMiddleware.multiLineFormatter,
              ),
            ]));

  if (!kReleaseMode) {
    runApp(InvoiceNinjaApp(store: store));
  } else {
    await SentryFlutter.init(
      (options) {
        options.dsn = Config.SENTRY_DNS;
        options.release = const String.fromEnvironment('SENTRY_RELEASE',
            defaultValue: kClientVersion);
        options.dist = kClientVersion;
        options.beforeSend = (SentryEvent event, {dynamic hint}) {
          final state = store.state;
          final account = state.account;
          final reportErrors = account.reportErrors;
```

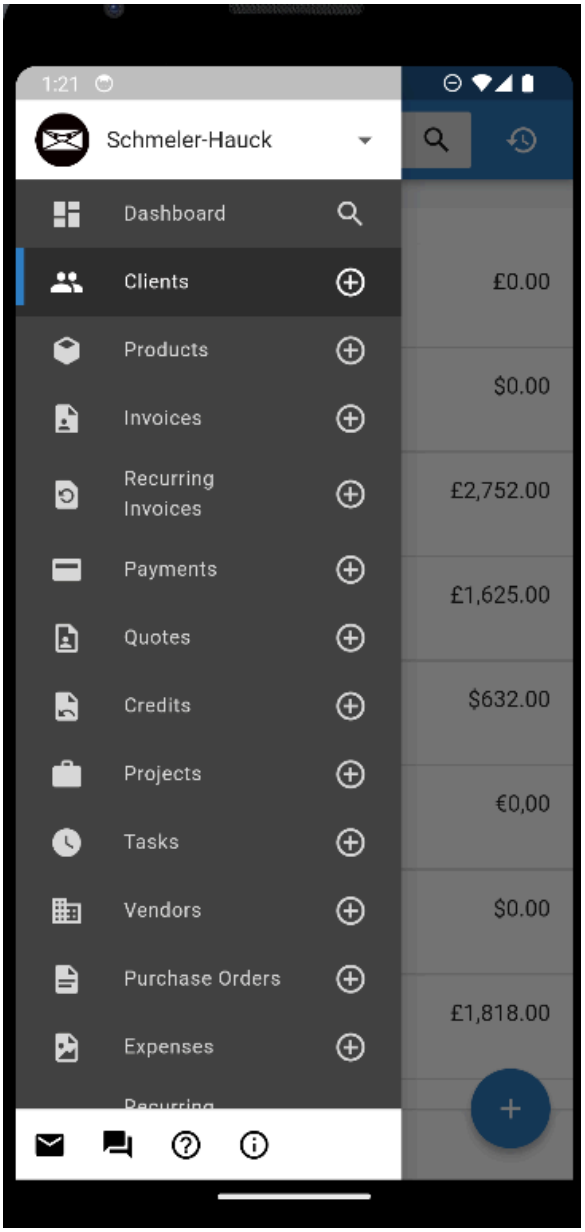
```
        if (!reportErrors) {
          return null;
        }

        event = event.copyWith(
          environment: '${store.state.environment}'.split('.').last,
          /*
          extra: <String, dynamic>{
            'server_version': account.currentVersion,
            'route': state.uiState.currentRoute,
          },
          */
        );

        return event;
      };
    },
    appRunner: () => runApp(InvoiceNinjaApp(store: store)),
  );
 }

 /*
 if (isWindows()) {
  doWhenWindowReady(() {
    final win = appWindow;
    win.title = 'Invoice Ninja';
    win.show();
  });
 }
 */
}

Future<AppState> _initialState(bool isTesting, SharedPreferences prefs) async {
 final prefString = prefs.getString(kSharedPrefs);

 final url = WebUtils.apiUrl ?? prefs.getString(kSharedPrefUrl) ?? '';
 if (!kReleaseMode) {
  //url = kAppStagingUrl;
  //url = kAppProductionUrl;
  //url = kAppDemoUrl;
 }

 PrefState? prefState = PrefState();
 if (prefString != null) {
```

```
    try {
      prefState = serializers.deserializeWith(
          PrefState.serializer, json.decode(prefString));
    } catch (e) {
      print('## Error: Failed to load prefs: $e');
    }
  }

  prefState = prefState!.rebuild((b) => b
    ..enableDarkModeSystem =
        PlatformDispatcher.instance.platformBrightness == Brightness.dark);

  String? browserRoute;
  if (kIsWeb && prefState.isDesktop) {
    browserRoute = WebUtils.browserRoute;
    if (browserRoute!.isNotEmpty && browserRoute.length > 4) {
      if (browserRoute == '/kanban') {
        browserRoute = '/task';
        prefState = prefState.rebuild((b) => b
          ..showKanban = true
          ..useSidebarEditor[EntityType.task] = true);
      }
    } else {
      browserRoute = null;
    }
  }

  bool reportErrors = false;
  bool whiteLabeled = false;
  String? referralCode = '';

  if (kIsWeb) {
    reportErrors = WebUtils.getHtmlValue('report-errors') == '1';
    whiteLabeled = WebUtils.getHtmlValue('white-label') == '1';
    referralCode = WebUtils.getHtmlValue('rc');
    if (reportErrors) {
      print('## Error reporting is enabled');
    }
  }

  return AppState(
    prefState: prefState,
    url: Config.DEMO_MODE ? '' : url,
    referralCode: referralCode,
```

```
    reportErrors: reportErrors,
    isWhiteLabeled: whiteLabeled,
    currentRoute: browserRoute,
  );
}

void _registerErrorHandlers() {
  /*
  // * Show some error UI if any uncaught exception happens
  FlutterError.onError = (FlutterErrorDetails details) {
    FlutterError.presentError(details);
    //errorLogger.logError(details.exception, details.stack);
  };
  // * Handle errors from the underlying platform/OS
  PlatformDispatcher.instance.onError = (Object error, StackTrace stack) {
    //errorLogger.logError(error, stack);
    return true;
  };
  */

  ErrorWidget.builder = (FlutterErrorDetails details) {
    return Material(
      color: Colors.grey.shade100,
      child: Center(
        child: Text(
          details.toString(),
          style: TextStyle(color: Colors.black),
        ),
      ),
    );
  };
}
```

Output:





Conclusion:

In conclusion, implementing navigation, routing, and gestures in a Flutter application enhances user experience by enabling seamless navigation between screens, defining clear routes for different app sections, and providing intuitive interactions through gestures. This experiment demonstrated the importance of these features in creating engaging and user-friendly Flutter apps.