Descrizione del progetto finale

Nome del progetto di laboratorio:

StudyBuddy

GRUPPO

Informazioni sul gruppo di laboratorio.

Nome del gruppo:	##Nome del gruppo##
Componenti:	* Merve Karacaoğlu
Github/Gitlab Repo Link:	• https://github.com/mervekaracaoglu/StudyApp

DATE

Le date principali del documento.

Data di sottomissione della proposta di progetto	14.05.2025
Data di accettazione della proposta di progetto	16.05.2025

DESCRIZIONE BREVE

Descrizione della App in una frase.

StudyBuddy is a mobile application designed to support students in managing their studies by allowing them to log study sessions, track progress with analytics and streaks, organize tasks with subjects and tags, set personalized reminders, and customize the app's appearance, leveraging modern Android architecture with MVVM, Room, StateFlow, and Jetpack Compose.

DEFINIZIONI

Di seguito la definizione dei termini, abbreviazioni e acronimi utilizzati.

Termine	Definizione
MVVM	Architecture pattern that separates the UI (Composables), business logic (StudyViewModel), and data (Room) layers.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Room	Used to persist study sessions locally in SQLite database via StudySession entity and SessionDao.
StateFlow	StateFlow in ViewModel to expose live-updating session and analytics data to the UI
Jetpack Compose	All UI screens like DashboardScreen, LogSessionScreen
DAO	SessionDao defines methods like getTodayStudyMinutes() to interact with the Room database
Navigation	Jetpack Navigation to switch between screens (NavHost and NavController)
DataStore	Used for storing user preferences such as theme mode (dark/light) and preferences for weekly goals
ViewModel	Holds UI state and business logic. Screens use a ViewModel to maintain state across configuration changes
ViewModelFactory	Used to provide the ViewModel with a repository and oter dependencies during app startup
AlarmManager	Triggers a custom BroadcastReceiver at the appropriate time
Coroutines	Asynchronous tasks like reading from database or network calls
Service	To perform long running operations in the background like the timer that continues even if the user switches screens
Repository	Abstracts the data access from ViewModel
LifeCycleService	Pomodoro timer runs as LifeCycleService in the foreground using coroutines.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

1. CONTESTO DEL PROGETTO

1.1. Situazione attuale

There are several study and productivity apps available on the Play Store, such as Forest, StudySmarter, and Pomodoro-based timers. These apps offer useful features like timers, reminders, or general productivity tracking, they often have key limitations for students:

- Many apps focus solely on time tracking without allowing detailed session logging (subject, notes, or study goals).
- Most solutions are not fully offline and rely heavily on cloud features, limiting usability in low-connectivity environments.
- Customization is often minimal. Users cannot personalize themes, tag sessions, or adapt the experience to their study style.
- Some platforms are overloaded with features unrelated to studying, creating cluttered and unfocused interfaces.
- Few apps provide subject-based analytics, daily/weekly statistics, or progress tracking tailored to academic contexts.

As a result, there is a gap for a lightweight, offline-capable, and customizable app focused specifically on helping students track and improve their study habits in a structured and intuitive way.

1.2. Benefici e creazione di valore

StudyBuddy offers a targeted, student centered solution to study management by combining a personalized planner with a timer. The main benefits and value include:

- **Improved productivity**: Helps users track and visualize their study patterns, motivating consistent habits through streaks and goals.
- **Offline usability**: StudyBuddy works entirely offline, making it accessible in classrooms, libraries, or low-connectivity environments.
- **Customizability**: Users can personalize the experience with theme settings, session tags, notes.
- **Clarity and simplicity**: Designed with an intuitive interface using Jetpack Compose, the app avoids clutter while still providing functionality.
- **Analytics & motivation**: Study statistics, such as daily time totals, longest sessions, and subject breakdowns, help users stay accountable and measure progress.
- **Reminders**: Users can schedule reminders to avoid forgetting important sessions or tasks.
- **Extendable foundation**: Built on MVVM and Jetpack libraries, the app is modular and can be extended with authentication, syncing, or collaborative features.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

1.3. Obiettivi del progetto

Elenco di aspetti specifici e verificabili che realizzerete nella vostra App e che la renderanno "appetibile".

Study Session Logging	Users can log sessions by subject, duration, tags, and notes with automatic timestamps. Furthermore, users can choose if the log is completed or not and set a due date, making it functionable as tasks.
Offline Local Storage	All data is stored locally using Room, ensuring full app functionality without internet access
Analytics	Includes visual summaries of total study time, streaks, goals, and subject breakdowns.
Reminder System	Users can set reminders and due dates for study-related tasks with support for notifications
Bottom Navigation UI	Easy access to core sections: Home, Reminders, Analytics, and Timer.
Theme Customization	Allows users to personalize the interface using selectable color themes (stored via DataStore).
Modern Android Architecture	Built with Jetpack Compose, MVVM, ViewModel, Room, Coroutines, Navigation, and StateFlow.
Future-proof design	Allows adding Firebase login for session sync and user-specific data.
Persistent Storage	Settings such as dark mode and weekly goals are stored with Jetpack DataStore.
Timer	Users can track their study sessions with a Pomodoro Timer that uses 25/5/15 method.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

2. PROFILO DEL PROGETTO

2.1. Ambito del progetto

The goal of the StudyBuddy app is to develop a mobile app for students to plan, log, and analyze their study habits in an organized and personalized way. The app will include features such as:

- Logging study sessions (with subject, duration, notes, tags, completed checkbox, and due dates)
- A dashboard with analytics (daily/weekly progress, streaks, most used tags, etc.)
- A reminder and task system with notification support
- A bottom navigation bar to access key areas like Home, Reminders, Timer, Analytics
- Theme customization and persistent user settings using DataStore
- Offline functionality using Room for local storage

The app is designed with clean MVVM architecture, Compose UI, ViewModel, Room, StateFlow, and Coroutines to ensure reusability and future expandability.

What is developed:

- Full offline-capable study session tracker
- Local Room database with DAO
- Jetpack Compose UI for all screens
- Dashboard with analytics
- Reminder system using BroadcastReceiver and AlarmManager
- Customizable UI with persistent settings via DataStore
- Navigation using Jetpack Navigation component
- Persistent storage for weekly goals via DataStore
- Timer developed with LocalBroadcastManager and LifeCycleService

What could be developed later:

- Firebase Authentication (email and Google login)
- Cloud sync using Firestore or Realtime Database
- Sharing progress or sessions with friends
- Exporting study logs or statistics (CSV/PDF)
- Integration with Google Calendar or Google Maps (for location-based study sessions)

External Services:

In its initial version, the app is fully offline. However, it is architected in a way that supports future integration with external services such as: Firebase Authentication, Firebase Firestore or Realtime database

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

2.2. Profilo della soluzione

The StudyBuddy app follows a modular and scalable architecture based on the MVVM (Model-View-ViewModel) pattern and uses the latest Jetpack libraries to ensure clean separation of concerns and maintainability.

Architecture Layers:

- **Model**: Includes the Room database, StudySession entity, SessionDao, and StudyRepository. This layer manages all data storage, queries, and persistence.
- **ViewModel**: The StudyViewModel exposes UI state using Kotlin StateFlow and handles business logic such as inserting sessions, calculating study totals, and managing reminders.
- **View (UI)**: Built with Jetpack Compose, the app's UI layer contains composable screens such as DashboardScreen, LogSessionScreen, ReminderScreen, and more.

Navigation:

The app uses Jetpack's Navigation Component to manage screen transitions between tabs like: Home (dashboard), Log a Session, Logged Sessions, Reminders, Timer, Analytics. These are connected via a bottom navigation bar for quick access.

State Management:

State is managed using StateFlow, which allows the UI to reactively update whenever the ViewModel data changes.

Persistent Preferences:

User settings such as dark mode are stored using Jetpack DataStore**,** which ensures settings are saved even when the app is closed or restarted.

Reminders & Timer:

A combination of BroadcastReceiver, AlarmManager and Services is used to deliver scheduled reminders and support timer functionality (e.g. Pomodoro).

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

3. VINCOLI E ASSUNTI (REQUIREMENTS)

3.1. Vincoli tecnologici

Functional requirements:

- The user must be able to **log a study session** including subject, duration, notes, completed checkbox, and tags.
- Sessions must be **stored locally** using the Room database and remain accessible offline.
- The app must provide a **dashboard** displaying total study time, streaks, goal progress and two buttons to navigate to log session and logged sessions screen.
- The user can **view a list of logged sessions**, with filters by completion and due date.
- The user can **set reminders** for study sessions or tasks, stored and triggered via AlarmManager.
- The user can set a Pomodoro timer for 25 minutes work, 5 minutes break and 15 minutes long break after 4 sessions via **LifecycleService** (a type of Foreground Service) and coroutines.
- Users must be able to **customize the app theme** and preferences (dark/light mode).
- The app must use **bottom navigation** to switch between Dashboard, Analytics, Reminders, and Timer.

Non-functional requirements:

- The app should function **entirely offline**, with persistent local storage.
- The app should be built using **Jetpack Compose** for a modern, reactive UI.
- Architecture must follow **MVVM** with clear separation between layers (UI, ViewModel, Repository, DB).
- UI components must **react to state changes** using Kotlin StateFlow and collectAsState().
- Code should be **modular and maintainable**, supporting future extension.
- All UI strings should be **externalized using string resources** for localization support.
- The app must follow **Material 3** design principles.
- Preferences must be **persisted** using Jetpack DataStore**.**
- The app should provide **integration and unit testing** to ensure reliability

3.2. Eventuali assunti

- Users will primarily interact with the app on Android 11+ devices with at least **API level 26.**
- The initial version will **not require internet access** or a backend server.
- If Firebase Auth is used, it will be **optional** and not mandatory for using core features.
- Background reminders are assumed to work reliably using Broadcast Receiver and the system AlarmManager, even if the app is closed.
- All study session data is **user-specific** and stored locally; no multi-user or collaborative features are required at this stage.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

4. DESIGN

MVVM Pattern in StudyBuddy

- **Model Layer**
  - Consists of data classes (e.g. StudySession), the Room database (StudyDatabase), and DAO interfaces (SessionDao).
  - Handles all data operations such as inserting, querying, and updating sessions.
  - Also includes the repository(StudyRepository), which acts as an abstraction layer between ViewModel and DAO.
- **ViewModel Layer**
  - Classes like StudyViewModel expose reactive data (StateFlow) to the UI.
  - They handle business logic such as calculating daily totals, loading upcoming sessions, and managing user preferences.
  - ViewModels survive configuration changes and are lifecycle-aware.
- **View Layer (Compose UI)**
  - Built entirely with Jetpack Compose for declarative UI.
  - Screens include DashboardScreen, LogSessionScreen, ReminderScreen, SettingsScreen, and LoggesSessionsScreen, Timer Screen.
  - UI observes ViewModel state using collectAsState() from StateFlow.

Jetpack Components Used:

Jetpack Compose: UI layer built entirely with Compose. Components are reactive and modular.

ViewModel: Maintains screen state and encapsulates logic

Room: Handles local data persistence for sessions and reminders.

StateFlow: Used instead of LiveData for observing state in the ViewModel.

Navigation: Jetpack Navigation Component manages screen transitions with NavHost and NavController.

DataStore: Stores persistent settings like dark/light mode and weekly goals settings

Coroutines: Enables background tasks (e.g., database access) without blocking the main thread.

AlarmManager: Schedules notifications for reminders in the background.

Services: LifeCycleService a Foreground service for long running tasks (Pomodoro Timer)

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

How the Architecture Satisfies the Requirements:

Offline Functionality: Data stored using room, no cloud dependency.

Modular and Scalable Design: MVVM pattern with repository abstraction

UI state updates reactively: StateFlow + collectAsState() in Compose

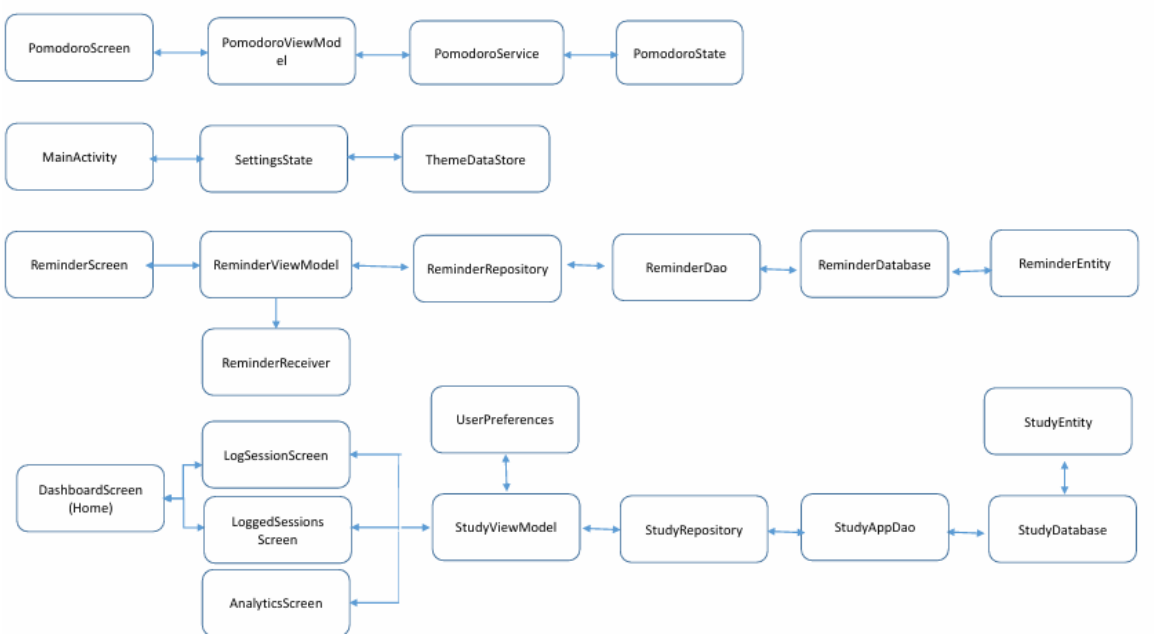
Persistent User Settings: Jetpack DataStore

Navigating Across Screens: Jetpack Navigation with NavHost, NavController, BottomBar.

Scheduling Reminders: AlarmManager for one-time and recurring background tasks

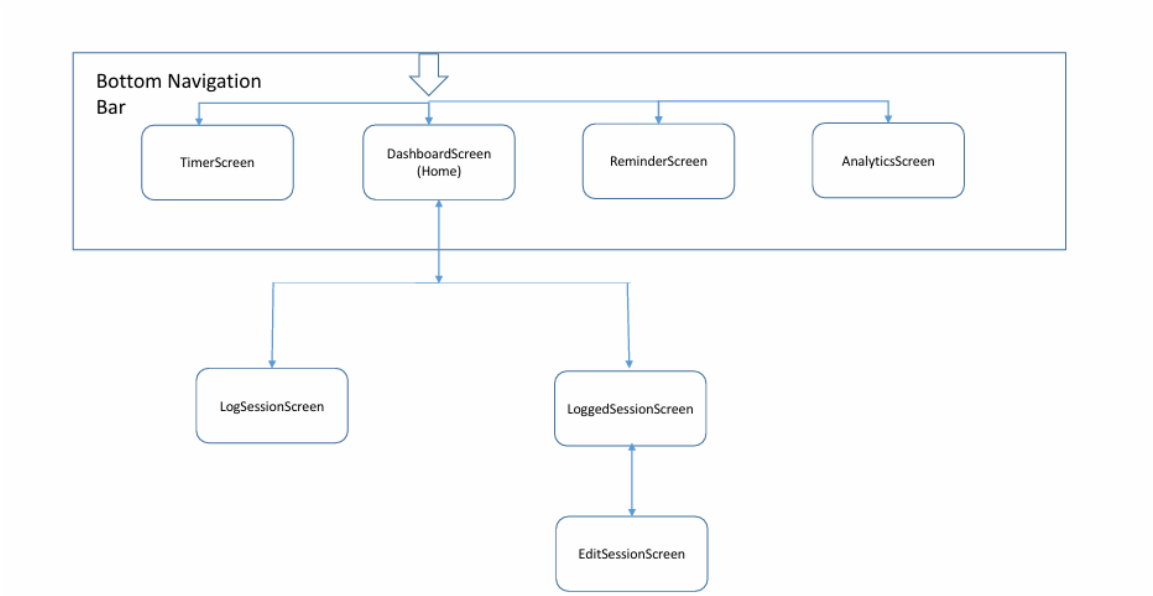
Code maintainability: Clear layer of separation, use of ViewModelFactory

Future Extensibility: Modular ViewModel-Repository setup and dependency injection ready.



5. IMPLEMENTAZIONE

MVVM Architecture
com.example.studyap/
database/ Room database entities: StudyAppDao, StudyDatabase, StudyEntity
reminders/ ReminderDao, ReminderDatabase, ReminderEntity, ReminderReceiver, ReminderScreen, ReminderViewModel, ReminderViewModelFactory
repository/ StudyRepository
screens/ AnalyticsScreen, DashboardScreen, EditSessionScreen, LoggedSessionsScreen, LogSessionScreen
datastore/ThemeDataStore, ThemeState, UserPreferences
timer/ PomodoroScreen, Pomodoro Service, PomodoroState, PomodoroViewModel, PomodoroViewModelFactory
viewModel/ StudyViewModel, StudyViewModelFactory
MainActivity



Session Logging Flow:

- LogSessionScreen (UI) calls StudyViewModel.addSession(...)
- StudyViewModel passes the session to StudyRepository
- StudyRepository executes the logic and uses StudyAppDao to insert the session
- StudyDatabase provides the singleton room instance
- The new session is saved locally in room and flows back to the UI through Flow -> State Flow

Reactive Data Flow (StateFlow)

- DAO methods like getAllSessions() return a Flow<List<StudySession>>

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

- Repository exposes these flows to the ViewModel
- ViewModel converts them into StateFlow and exposes them to the UI
- Compose screens use collectAsState() to observe and react to updates

Navigation Flow

- In MainActivity.kt, the app sets up NavHost() using routes defined in the same file.
- Screens navigate with navController.navigate("log_session") or "logged_sessions"
- Each screen is a destination: "dashboard", "log_session", "reminders", etc.

Reminders & Notifications

- In ReminderScreen.kt, the user picks a time and message
- This info is sent to a ReminderReceiver (a custom Broadcast Receiver) that uses AlarmManager to schedule an Intent
- It reads the intent data and shows a system notification using NotificationManager and NotificationCompat.

Persistent User Preferences (Theme)

- In DashboardScreen, users can toggle dark mode
- These preferences are stored via ThemeDataStore
- ThemeDataStore handles reading/writing preferences
- The app reads preferences at startup and applies them using MaterialTheme

Pomodoro Timer

- The Pomodoro timer runs as a LifecycleService in the foreground, using Kotlin coroutines for real-time countdown logic.
- Notifications are updated every second, and user controls like Pause, Reset, and Stop are exposed directly in the notification.

6. TEST

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

Unit Testing:

ReminderViewModelTest:

To verify that ReminderViewModel correctly loads reminders from the database.
ReminderDao was mocked using mockito-kotlin.
Test frameworks that are used are JUnit4 for assertions and turbine for testing Kotlin Flows**.**
reminders emits data from DAO: Confirms the reminder list is emitted properly.

StudyViewModelTest:

To validate the StudyViewModel logic that interacts with StudyRepository and exposes state to the UI.
StudyRepository was mocked to simulate database operations.
Test frameworks that are used: Junit4, Turbine for collecting flows, Mockito for mocking behavior, Robolectric for Android components without an emulator.

Tests that are written:

allSessions emits data from repository	Verifies sessions are loaded correctly
deleteSession should remove session	Checks if delete operation triggers on the repository
loadTodayStudyMinutes updates value	Validates computation of today's study time
setWeeklyGoal updates flow and saves value	Tests that goal-setting logic updates the expected Flow
weeklyStudyMinutes sums only current week	Ensures only sessions within this week are counted
currentStreak stops at non-consecutive day	Confirms streak is broken if there's a skipped day

Integration Testing:

StudyIntegrationTest:

The StudyIntegrationTest class verifies the full integration of: Room database (StudyDatabase), Study Repository, StudyViewModel. An in-memory database is used to simulate real database interactions without affecting production data.

Tests:

addAndReadSession	Verifies that sessions added via ViewModel are persisted and retrievable
updateSession_shouldModifySession	Confirms updates to sessions are reflected in storage
deleteSession_shouldRemoveSession	Ensures deleting sessions works end-to-end
emptyState_shouldReturnZeroValues	Validates the app's behavior with no data
currentStreak_shouldCountConsecutiveDays	Confirms correct streak calculation logic when days are consecutive

Room.inMemoryDatabaseBuilder(...) to simulate database. runBlocking for coroutine-based operations. Kotlinx.coroutines.flow.first for testing flow emissions.Junit4 for assertions.

User Testing

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603

To validate the usability and effectiveness of the app, informal user testing was conducted with a small group of target users.The goal was to assess the clarity of the user interface, discover any usability issues, and gather feedback on the app's core features.
Participants: 3 students with Android smartphones
Method: Remote testing
Tasks given: Log a study session for a subject, start and stop a pomodoro timer, set a custom study reminder, navigate to the analytcis screen and review session data, add tasks using the log screen
Outcomes: Participants were able to complete the tasks. No crashes or bugs reported. Feedback has been recorded for future improvements
Feedbacks: Clean, minimal interface, easy to navigate, helpful to track deadlines, useful timer for tracking time passed.
Suggestions: Customizable timer lengths, ability to snooze reminders, set task priority, color themes, notifications when a uncompleted log sessions due day is approaching.

7. CONCLUSIONI

The StudyBuddy app was developed with the goal of providing a complete, intuitive, and customizable tool to support students in planning and tracking their study activities. Throughout the development process, the app has successfully met all the functional and non-functional requirements initially defined, proving to be an effective and well-structured solution both architecturally and in terms of user experience.

Requirements Fulfilled

All the functional requirements outlined in Section 3 have been fully implemented and tested:

- Users can log study sessions with subject, duration, tags, notes, deadline, and completion status.
- Sessions are stored locally using Room database and are available offline.
- The reminder system allows users to schedule notifications for tasks or future sessions using AlarmManager.
- The user interface adheres to Material Design 3 guidelines and is built entirely with Jetpack Compose.
- The Pomodoro timer works in the background via a dedicated Service.
- The app features responsive navigation with a bottom navigation bar.
- Interface personalization is handled using Jetpack DataStore.
- The architecture is modular and adheres strictly to the MVVM pattern, with clear separation between UI, ViewModel, and Repository layers.

These implementations demonstrate strong alignment with the design choices described in Section 4, where Jetpack technologies (Room, ViewModel, StateFlow, Navigation, AlarmManager, Compose) were selected to meet the project's constraints and goals.

Validation Through Testing

To ensure the app's correctness and reliability, a multi-layer testing strategy was employed:

- **Unit tests** on ViewModels and Repositories using mocks, simulating insertions and queries.
- **Integration tests** on database, viewmodel and repository using in-memory database.
- **User testing** conducted with real students, who completed practical tasks without requiring assistance.

These tests confirmed that:

- The app behaves correctly in realistic scenarios and does not crash.
- State is managed consistently, and the UI updates properly based on data changes.
- Key features are stable and functional.
- The interface is intuitive even for non-technical users.

Final Thoughts

StudyBuddy demonstrates that it is possible to build a modern, stable, and helpful mobile app by starting from a clean architectural foundation and leveraging the latest Jetpack libraries. The project met all initial goals and now stands as a base for future development, such as cloud sync with Firebase (Auth and Firestore), progress sharing between users, or data export capabilities. The modularity and scalability of the codebase, and the attention to user experience make this app a practical example of mobile development focused on usability, performance, and long-term maintainability.

Dipartimento di Informatica, C.so Svizzera 185. Tel. 0116706711 Fax. 011751603