# Jarvis360 — System Documentation (Snapshot)

Last updated: 2025-10-16

This document summarizes the current state of the Jarvis360 project (backend, frontend), developer runbook, test instructions, and recent changes made to improve CSV intake, forecasting, async bootstrap, tests and CI.

## High-level architecture

- Backend: Django 5.2 application with Django REST Framework endpoints under `api/`.

- Frontend: React (Create React App) contained in `client/` with Recharts for charts and html2canvas for exports.

- Forecasting & analytics: in-browser utilities under `client/src/utils` (`analytics.js`, `forecast.js`) performing aggregation, linear and Holt forecasts and optional bootstrap CI.

## Recent changes (summary)

- Replaced the naive CSV parser with PapaParse (`client/src/utils/csv.js`) to correctly handle quoted fields and commas inside fields. Dates are normalized to `YYYY-MM-DD` where possible.

- Centralized forecast helper `client/src/utils/forecast.js` ensures consistent return shape. Holt forecast supports optional bootstrap confidence intervals and an async path with a cancellable worker-style promise.

- Frontend improvements to UX for async bootstrap: clearer status indicator and an explicit Cancel CI button which revokes the in-flight async promise and preserves the last visible forecast.

- Unit tests added/updated: CSV parse tests (including quoted field cases), forecast async path tests, and existing analytics tests all run under `client/` Jest harness.

- CI workflow added: `.github/workflows/ci.yml` runs backend Django tests and frontend tests on push/PR.

## CSV input expectations

- The client expects a CSV with a date-like column and an MRR/amount column. Common header names are detected automatically, or users can map headers in the UI.

- Supported: quoted fields, embedded commas, escaped quotes. Dates will be normalized to `YYYY-MM-DD` if parsable by the JS Date constructor.

- For robust, production usage consider using consistent ISO dates in the CSV to avoid locale parsing ambiguity.

## Forecasting behavior

- `client/src/utils/analytics.js` provides:

- `computeMonthlySeries(records)` — aggregates records into monthly totals and computes new/expansion/churn heuristics.

- `linearForecast(series, monthsOut)` — OLS regression on time index with approximate 95% CI.

- `holtLinearForecast(series, monthsOut, options)` — Holt's linear double-exponential smoothing with optional auto-tune, bootstrap CI, and an async bootstrap path that supports cancellation.

- `client/src/utils/forecast.js` provides `computeForecastFromRecords(records, options)` which normalizes inputs, runs the selected method and always returns a predictable structure. If bootstrapAsync is requested it returns a Promise which has an attached `revoke()` method (when possible) so callers can cancel long-running CI computations.

## Async bootstrap & cancellation

- When the user enables Holt bootstrap with `bootstrapAsync`, a worker-like async computation is used. The promise returned by the helper can include a `revoke()` method which terminates the underlying worker or abandons the computation when run in non-browser environments.

- The UI exposes a Cancel CI button which calls `revoke()` and stops the computing indicator while keeping the last known forecast visuals.

## How to run tests locally (developer runbook)

1. Backend (Django):

- Create and activate a virtual environment (Windows PowerShell):

```powershell

python -m venv venv

.\venv\Scripts\Activate

pip install -r requirements.txt

```

- Run Django tests:

```powershell

python manage.py test

```

Notes: The `requirements.txt` pins `numpy` and `pandas` versions; if pip fails to install a pinned `numpy` wheel for your Python version, either use a Python version compatible with the pinned wheel or update the pinned numpy in `requirements.txt`.

2. Frontend (client):

- Install and run tests from the project root:

```powershell

npm --prefix .\client install

npm --prefix .\client test -- --watchAll=false

```

- Start dev server:

```powershell

npm --prefix .\client start

```

3. CI (GitHub Actions):

- Commits to `master` and Pull Requests trigger the workflow defined at `.github/workflows/ci.yml`. The workflow runs backend Django tests (Python 3.11) and frontend tests (Node 18).

## System / environment caveats

- numpy wheel availability: some pinned numpy versions may not have wheels for your Python minor version. If pip fails for a pinned numpy, either install a compatible Python interpreter or loosen the pin in `requirements.txt`.

- Plotly is optional — if not installed interactive plots in the admin may not render, but tests should still run.

## Developer notes & next steps

- Consider adding end-to-end tests for the file upload + forecasting pipeline (Cypress or Playwright) to validate the entire UX across browsers.

- Replace the in-repo worker script with a formal Web Worker build step to improve reliability when performing heavy bootstrap sampling in browsers.

- Consider adding a graceful fallback UI for very large CSV uploads (progress indicator, chunked parse with PapaParse).

## Changelog (short)

- 2025-10-16: Replaced CSV parser (PapaParse), added async forecast tests, improved async bootstrap UX, added GitHub Actions CI, added this documentation.

----

If you want this `docs/System\_Documentation.md` exported into the existing `docs/System\_Documentation.docx`, tell me and I will (a) convert to a simple .docx using a Python script, or (b) paste its contents into the `.docx` if you prefer manual editing in Word.

Jarvis360 — System Documentation

Version: 1.0

Last updated: 2025-10-15

Maintainer: (project repository)

Overview

--------

Jarvis360 is a lightweight local-memory SaaS analytics sandbox for exploring customer churn, forecasting MRR, and modeling retention scenarios. It runs as a React single-page application (SPA) in `client/` with a small Django backend skeleton in the repository root (most work in this sprint is frontend-focused). The UI is intentionally local-first: uploaded datasets stay in-browser memory and localStorage; no external network is contacted unless you add integrations.

Goals of this document

----------------------

- Describe how the system is structured and how the main components interact.

- Explain features and user flows (Data Intake, Overview, Forecasting, Scenarios, Risk & Actions, Settings).

- Document important implementation details (files, utilities, autosave keys, export formats).

- Provide instructions for common tasks: running tests, adding features, exporting scenarios.

- Note limitations and future improvement ideas.

High-level architecture

-----------------------

- Frontend (client/): React app created with Create React App.

- Entry point: `client/src/App.jsx` — main application container and view router.

- Components: `DataDashboard`, `DataOverview`, `TimeSeriesForecast`, `WhatIfSimulation`, `ChurnPredictor`, `Settings`.

- Utilities: `client/src/utils/analytics.js` (data aggregation + forecast helpers), `client/src/utils/summarizer.js` (local scenario summarizer).

- UI helpers: `client/src/Toast.jsx`, `client/src/ChurnChart.jsx`.

- Backend (Django): present, but not required for local usage. `requirements.txt` has pinned server libraries if you later enable backend features.

- Note: The separate `analysis` Django app was removed and its CSV analysis endpoints were consolidated into the `api` app (`api/views.py`). The API endpoints for overview/simulation are reachable under `/api/overview/` and `/api/simulation/` (previously available under `/api/analysis/...`). Update any external callers accordingly.

Key data flows

--------------

1. CSV Import (Data Dashboard)

- User uploads a CSV via `DataDashboard`.

- `parseCSV` (in `App.jsx` DataDashboard component) maps headers to canonical fields: `id`, `name`, `MRR`, `date`, `churnProbability`, `supportTickets`, `lastActivityDays`, `contractLengthMonths`.

- Normalized records are returned to the main `App` via `onDataUpload`, which stores them in-memory in `customers` state.

2. Aggregation & Overview

- `overviewData` is computed from `customers` using `computeMonthlySeries(customers)` (in `client/src/utils/analytics.js`) to produce a monthly series of totals and deltas (new/expansion/churn).

- `DataOverview` renders key metrics (Total Customers, Total MRR, Average MRR, Est. Annual Revenue) and uses `ChurnChart` to show New / Expansion / Churn stacked bars by month.

3. Forecasting

- `TimeSeriesForecast` receives `overviewData.monthlySeries` and builds a numeric series of monthly totals.

- `linearForecast` logic (inlined and in `utils/analytics.js`) computes an OLS linear projection across the historical period and extends N months with a 95% CI band computed from residual standard deviation.

- The UI uses Recharts (LineChart/Area/Brush) to show Actual, Predicted, and CI band; users can download CSV, download PNG (html2canvas), or copy a PNG to clipboard.

4. What-If Simulation

- `WhatIfSimulation` accepts the enhanced customer list (customers with `riskScore` and `riskLevel` computed in App).

- Users tune lever sliders (discount, support, campaign) and select risk-target levels (All/High/Medium).

- A memoized simulation calculates: `potentialMRRLoss`, `simulatedMRRLoss`, `projectedMRRSaved`, and `targetCustomerCount`.

- Features:

- Save/load/delete scenarios (localStorage key: `jarvis\_saved\_scenarios\_v1`).

- Autosave draft of the what-if picklist to `jarvis\_autosave\_whatif\_v1` on every change.

- Export scenario CSV of targeted customers.

- Export scenario JSON (added recently) — includes meta, parameters, and a snapshot of results.

- Import scenario JSON to restore parameters locally.

- Local summarizer: `client/src/utils/summarizer.js` produces a short human-readable summary (no external API calls by default).

5. Churn Predictor (Action List)

- `ChurnPredictor` lists high and medium risk customers, visualizes riskScore bars, and allows marking customers as contacted (local flag update).

Important files and purpose

--------------------------

- `client/src/App.jsx`: Main app, view routing, component definitions for DataDashboard, DataOverview, Forecast, Simulation, ChurnPredictor, and Settings. Contains higher-level glue code for state, toasts, and exporters.

- `client/src/utils/analytics.js`: computeMonthlySeries(records) and linearForecast(series, monthsOut) — used by Overview and Forecast.

- `client/src/utils/summarizer.js`: generateScenarioSummary(simulationResults, whatIfData) — local heuristic for short scenario summaries.

- `client/src/Toast.jsx`: non-blocking toast component used across the app.

- `client/src/ChurnChart.jsx`: Recharts-based stacked bar chart for New/Expansion/Churn.

- `client/package.json` and `client/PINNED\_VERSIONS.md`: pinned frontend dependency versions for reproducible installs.

- `requirements.txt`: pinned Python dependencies for backend if you enable backend functionality.

How to run locally (frontend)

-----------------------------

1. Install dependencies (from `client/`):

cd client  
npm ci

2. Start dev server:

npm start

3. Run tests:

$env:CI='true'; npm test -- --watchAll=false

Notes: We pin dependencies (see `PINNED\_VERSIONS.md`) to avoid unexpected upgrades. Use `npm ci` in CI to replicate exact installs.

Export & sharing

-----------------

- Forecast CSV: "Download CSV" button in the Forecast view.

- Forecast image: "Download Image" — rasterized with html2canvas for a full-container capture.

- Copy Image: attempts to write PNG to clipboard (browser security may block this; download fallback is available).

- Scenario CSV: export targeted customer rows from the Simulation view.

- Scenario JSON: export/import full scenario parameters and results.

Autosave and persistence keys

-----------------------------

- Scenarios saved to localStorage: `jarvis\_saved\_scenarios\_v1` (array of saved scenario objects).

- Current what-if draft autosave: `jarvis\_autosave\_whatif\_v1` (single object with current parameters).

Testing

-------

- Unit tests are under `client/src/\_\_tests\_\_/` and run with Jest (react-scripts test). Current tests include:

- `computeMonthlySeries` happy path & edge cases

- `linearForecast` basic checks

- `summarizer` produces string output

Extensibility notes (how to add features)

-----------------------------------------

- Forecast algorithm:

- `client/src/utils/analytics.js` is the right place to add new forecasting algorithms (exponential smoothing, Prophet, ARIMA wrappers, etc.). If you add heavy dependencies (pandas, statsmodels), prefer server-side or WebAssembly approaches.

- LLM summaries:

- The current summarizer is local to respect privacy. If you want LLM-backed summaries, add an integration point in the Simulation UI that calls an API behind a backend endpoint (so you do not expose API keys in the browser). There is a pre-existing Python backend scaffold you can extend.

- Export improvements:

- For perfect vector-quality exports, instrument charting libs to produce SVGs and serialize those; html2canvas is used for reliable full-container rasterization today.

Known limitations & caveats

--------------------------

- All data is stored in-browser unless you explicitly add persistence to a server.

- Clipboard image writes depend on browser support and secure contexts (HTTPS).

- html2canvas cannot capture cross-origin images unless served with CORS headers. If your imported CSV references remote images, they may not render in the exported PNG.

- The forecasting model is intentionally simple (linear OLS + residual-based CI). For production workloads use a more robust modeling approach and validate assumptions.

Change log & maintenance

------------------------

- 2025-10-15: Initial documentation created. Features implemented in this sprint:

- Monthly aggregation & dedupe

- Recharts-based Forecast + CI band + Brush

- Robust exports: html2canvas based PNG export & clipboard copy

- Scenario persistence + JSON export/import + autosave draft

- Local summarizer heuristic

- Churn chart replaced with Recharts stacked bar

- Pinning frontend dependency versions

This file will be updated as the project evolves. The canonical source is `docs/System\_Documentation.md` in the repository.

Contact & contribution

----------------------

- Make pull requests against the repo. Add tests where possible and run `npm test`.

- For API integrations (LLMs, analytics), prefer adding server-side endpoints that keep keys out of client bundles.

Appendix: Quick references

-------------------------

- Autosave key: `jarvis\_autosave\_whatif\_v1`

- Scenarios key: `jarvis\_saved\_scenarios\_v1`

- Summarizer location: `client/src/utils/summarizer.js`

- Aggregation: `client/src/utils/analytics.js` (computeMonthlySeries)

- Main app & views: `client/src/App.jsx`

--- End of Document